Web Intelligence and Agent Systems: An International Journal 6 (2008) 1–28
IOS Press

Considering operational issues for multiagent conceptual inferencing in a distributed information retrieval application

Leen-Kiat Soh *
Computer Science and Engineering Department, University of Nebraska, 256 Avery Hall, Lincoln, NE 68588-0115, USA

Abstract. Our system, based on a multiagent framework called collaborative understanding of distributed knowledge (CUDK), is designed with the overall goal of balancing agents' conceptual learning and task accomplishment. The tradeoff between the two is that while conceptual learning allows an agent to improve its own concept base, it could be counter-productive: conceptual learning is time consuming and requires processing resources necessary for the agent to accomplish its tasks. In our current phase of research, we investigate the roles of resource and knowledge constraints, environmental factors (such as the frequency of queries), and learning mechanisms in a CUDK-based distributed information retrieval (DIR) application. In this application, an agent is motivated to learn about its neighbors' concept base so it can collaborate to satisfy queries that it cannot satisfy alone. Similarly, to conserve resources, an agent is motivated *not* to learn from neighbors that have been unhelpful in the past. As a result, it is possible for an agent to learn from a helpful neighbor that is not the authoritative expert in the system. The agents use neighborhood profiling to learn about other agents' helpfulness and conceptual inferencing to learn about other agents' known concepts. The helpfulness measure defines a metric called collaboration utility, and the inferencing results are stored in a translation table in which each entry is a mapping between two concepts plus an associated credibility score. The experiments investigate how operational and conceptual factors impact the DIR application's performance.

Keywords: Collaboration, distributed conceptual learning, dynamic profiling, resource description

1. Introduction

We first proposed and outlined the CUDK (originally called CUDO) framework in [19] with the following objectives: (1) to promote understanding among agents of a community, thus reducing communication costs and inter-agent traffic, (2) to improve cooperation among neighbors of a community, thus enhancing the strength (productivity, effectiveness, efficiency) of a neighborhood and supporting the distributed effort of the community, (3) to encourage pluralism and decentralization within a multi-agent community; i.e., the specialization of agents of a community so that each agent can rely on its neighbors for tasks not covered by its own capabilities, and (4) to enable collaborative learning to improve throughput of the community, intelligence in communication and task allocation, self-organization within the community, and integrity of the community.

Our approach to designing and building this framework is two-tiered. Our first task is to investigate the roles of operational and conceptual factors in system performance and how agents make decisions in balancing the task accomplishment and ontological learning goals under constraints. Given the experience and understanding gained from that first step, we intend to devise a set of policies for multiagent collaborative learning and local conceptual designs. Our objectives in the first tier include identifying (1) whether and how agents can function effectively without needing to understand all other agents, and (2) how agents can iden-

*Tel: +1 402 472 6738; Fax: +1 402 472 7767; E-mail: lksoh@cse.unl.edu.

tify a specific subset of neighbors whose knowledge would be valuable to learn about in terms of concepts. Our objectives in the second tier will then extend the insights and considerations gained or devised in the first tier to a formal framework for a general-purpose multiagent system that manages and builds sufficiently effective distributed, local concept bases. This paper is concerned with the first tier.

In our multiagent system, agents can have different topical terms or keywords describing a *concept*, and the semantics of each topical term that an agent knows is captured in the associated documents/links that the agent keeps for that term. For example, a topical term of "sports" may have the following set of associated documents: {www.espn.com, www.nba.com, www.atptour.com}. Different agents may know different concepts: different topical terms for the same concept, and/or different documents associated with the same topical term.

The current phase of our CUDK research focuses on understanding the interplay between conceptual knowledge and operational factors, exemplified through a distributed information retrieval (DIR) application. We have previously reported on our studies in neighborhood profiling and how knowledge of concepts and resources affect the quality of information retrieval in [20], emphasizing the incorporation of operational factors in conceptual learning. This paper extends that work with further experiments on the impact of query tasks, neighborhood profiling, and conceptual inferencing on the quality of query satisfaction. Specifically, the experiments reported here are (1) to investigate and identify how agents collaborate to understand each other under different operational constraints and setups, (2) to investigate how agents' inherent knowledge or concept bases affect their collaborations, and (3) to examine how multiagent collaborative learning affects overall performance. We have also previously reported our results on devising policies for tradeoff between conceptual inferencing and query satisfaction in [21].

In our DIR application, agents work as a team to accept and process queries and to learn about the relationships (1) among their individual knowledge of concepts, and (2) among their individual operational capabilities and characteristics in collaborative activities. Each agent maintains a *concept base* equipped with a repository of documents (or web page links), a translation table, and a neighborhood profile of other agents (i.e., neighbors) that it interacts with. The agent accepts a query from a user, then it (1) interprets that query and obtains the relevant documents, and/or (2) approaches credible or helpful neighbors to gather additional relevant documents. While an agent may always ask an authoritative expert neighbor for help on a particular query in a traditional DIR application, ours takes into account operational issues such that an agent may approach a lesser but more helpful neighbor for help. To identify such neighbors, an agent considers two values that it monitors: (1) a collaboration utility measure of each neighbor in its neighborhood, and (2) a credibility score between each pair of concepts, based on its translation table.

Our work is important to support the diversity in concepts that always exists among agents of a heterogeneous community due to different utilities [8]. It encourages the growth of such a community not by requiring the agents to conform to a standard set of concepts, but by promoting the uniqueness and freedom of expression of each member through cooperative learning in a multiagent framework. On-going research has focused on using a pre-defined, common ontology to share knowledge between agents by using a common set of ontology description primitives such as KIF [9] and Ontolingua [12]. However, the approach of using global ontologies has problems due to the multiple and diverse needs of agents and the evolving nature of ontologies [15]. Further, agents may have disparate references, which lead them to refer to the same object or concept using different terms and viewpoints; i.e., diverse ontologies [5]. Our CUDK framework allows members or agents to learn and identify what these disparate references mean. Furthermore, from the viewpoint of DIR applications, as described in [27], information resources are essentially passive since each source *delivers specific pages when requested*. There is a need for *active* information sources or modules acting on behalf of these information sources that are able to identify other information sources to help with satisfying a query, in order to improve the efficiency and effectiveness of the retrieval tasks. Thus, agents such as ours in the CUDK framework have the potential to add intelligence and autonomy to information sources to improve DIR applications.

Note also that in the following DIR application, we assume that a concept can be described by a set of relevant documents. This assumption, though not necessarily valid in many conceptual learning situations, allows us to proceed with our research design in investigating the feasibility of the proposed CUDK approach. It provides us with a DIR environment for the multiagent system and a conceptual inferencing mechanism that motivates the agents to learn from each other.

In the rest of this paper, we first describe the current CUDK framework and design in Section 2. Then we present our agent implementations in Section 3. Subsequently, we discuss our experiments and results in Section 4. In Section 5, we report on research and systems related to CUDK. We then address future work for our research and present our conclusions.

2. Framework and design

Our current research focuses on integrating conceptual and operational components of the multiagent CUDK framework with a DIR application. The key to collaboration in the multiagent system is the neighborhood profiling and reasoning process that determines which neighbors to approach and how to allocate the query tasks among the neighbors. That hinges upon the aforementioned two measures: *collaboration utility* and *credibility score*. Both measures are subjective—that is, they are computed from the viewpoint of the agent of one of its neighboring agents. Though our CUDK framework is a general one [19] we use information retrieval strategies in designing the CUDK modules and agents for our discussions here.

2.1. The distributed information retrieval (DIR) application

We apply our CUDK framework to DIR. In our multiagent system, an agent is motivated to collaborate to ultimately improve its own performance in satisfying queries that it receives from its users. A query, q, is a tuple of $\langle c_q, \#_q, o_q, s_q \rangle$, where c_q is the topical term or keyword, $\#_q$ is the number of documents or links desired, o_q is the originator of the query, and s_q is the current sender of the query. A query may be relayed multiple times such that $o_q \neq s_q$. The designation of o_q informs an agent who the originator of a query is such that it can return the documents to the originator.

Figure 1 shows the behavior of an agent that receives a query. Given a query q, an agent first decides whether to entertain the query. If the query comes directly from a user, then the agent will always entertain the query. If the query comes from one of its neighbors and the agent is presently busy, it may decide to decline the query. If the agent decides to entertain the query, then it first checks c_q against its own concept base. If it finds a match and it has enough links to satisfy $\#_q$, then it simply returns the results to s_q, without having to ask for help from other neighbors. If it finds a match but it does not have enough links to satisfy $\#_q$, then the agent needs to contact its neighbors to help satisfy the query. This is called *collaboration*. If the agent cannot find a match, i.e., c_q is not in its vocabulary, then it checks its translation table and sees whether c_q matches some keywords or terms that other neighbors know. If a keyword match is found, then the agent relays the query to the corresponding neighbor. This is called a *targeted relay*. Otherwise, the agent distributes the query to all neighbors. This is called *exploration* or a *generic relay*.

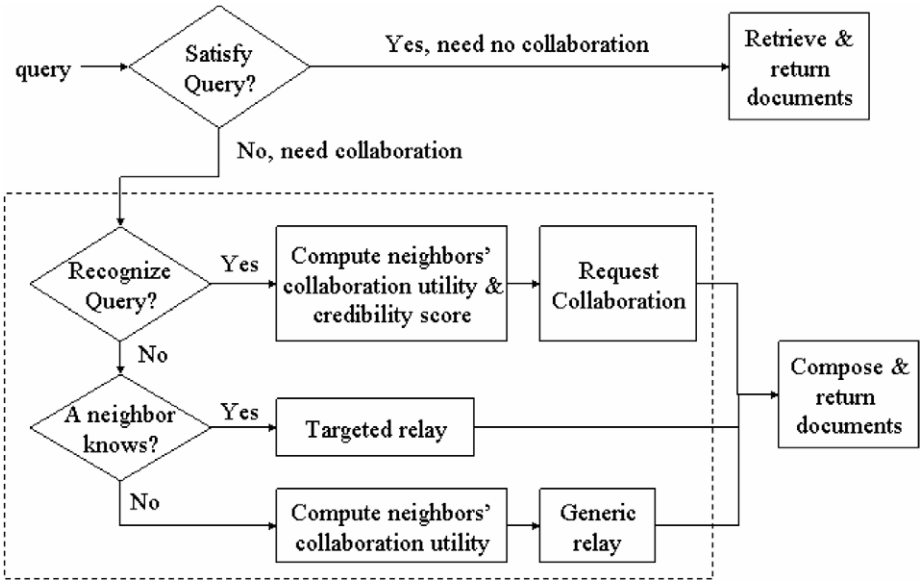

Fig. 1. The behavior of an agent when it receives a query.

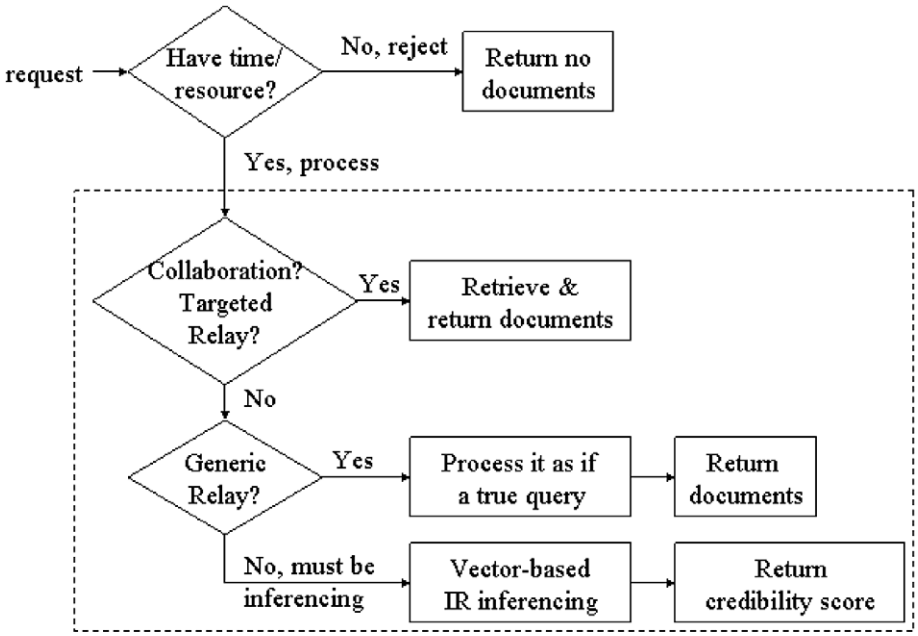

Fig. 2. The behavior of an agent when it receives a request from another agent.

Whenever agent a_i realizes that it has not been able to successfully satisfy a particular query, it checks its translation table. The quality of each mapping entry in the table reflects how credible the mapping is between a concept that a_i knows and a concept that a neighbor j of a_i, $n_{a_i,j}$, knows. If the credibility is low or NIL, then a_i sends an *inferencing* request to $n_{a_i,j}$ in an attempt to update the credibility score of the mapping. Further, agent a_i keeps a profile of each of its neighbors. Each interaction results in a change in the profile of the neighbor involved. Agent a_i uses the profile to compute the *collaboration utility* of a neighbor when a_i decides whether and how to request for help from its neighbors.

Figure 2 shows the behavior of an agent when it receives a request from an agent a_i. When a neighbor $n_{a_i,j}$ receives a request, it immediately rejects the request if it is busy or it does not have the resources to perform the query. Otherwise, if the request is a collaboration or a targeted relay, then it retrieves as many links as required and returns them to agent a_i. If the request is a generic relay, then $n_{a_i,j}$ performs the reasoning steps as outlined in Fig. 1. If the request is for inferencing, then it conducts a vector-based similarity match, to be discussed later.

Due to the resource competition between the need to improve concept bases for future collaborative activities and the need to satisfy current queries, an agent will have to strike a balance among the above actions. In the following, we describe the factors that agents consider when making such decisions.

2.2. Neighborhood, neighborhood profile, and collaboration utility

We define an agent's neighborhood as follows. An agent a_i has a neighborhood $N_{a_i} = \{n_{a_i,1}, n_{a_i,2}, \ldots, n_{a_i,N}\}$ such that it can contact and ask for help from each of the agents in the neighborhood. Agents in a_i's neighborhood are a_i's neighbors.

Neighborhood profile. Agent a_i keeps track of its interactions with its neighbors based on the interactions between a_i and the neighbors. The profile of a neighbor is a vector of 5 parameters, based on [22]: (a) $\_helpRate$, the ratio of successful collaborations when the agent a_i receives a request from the neighbor $n_{a_i,j}$ over the total number of requests from $n_{a_i,j}$ to a_i, indicating how helpful or useful a_i has been to $n_{a_i,j}$, (b) $\_successRate$, the ratio of successful collaborations when the agent a_i initiates a request to the neighbor $n_{a_i,j}$ over the number of total requests from a_i to $n_{a_i,j}$, indicating how helpful or useful $n_{a_i,j}$ has been to a_i, (c) $\_nowCollaborating$, a Boolean indicator as to whether the agent a_i and the neighbor $n_{a_i,j}$ are currently collaborating on another task, (d) $\_requestToRate$, the ratio of the total number of re-

quests from the agent to the neighbor $n_{a_i,j}$ over the total number of all requests from the agent a_i, indicating the reliance of a_i on $n_{a_i,j}$, and (e) $\_requestFromRate$, the ratio of the total number of requests from the neighbor to the agent a_i over the total number of all requests initiated by $n_{a_i,j}$, indicating the reliance of $n_{a_i,j}$ on a_i. We have chosen these five parameters, as they seemed to us a set of parameters that are easy to compute and still sufficiently capture the collaborative relationship between an agent and one of its neighbors.

Collaboration utility. The collaboration utility is an agent a_i's perception of how useful a neighbor $n_{a_i,j}$ has been with respect to its requests. We define the collaboration utility of a neighbor, $n_{a_i,j}$, as perceived by a_i as:

$$CU_{n_{a_i,j}} = \frac{\_helpRate + \_successRate}{5} + \frac{\_requestToRate + \_requestFromRate}{5} + \frac{(1 - \_nowCollaborating)}{5}. \quad (1)$$

With the above score, if an agent has been in close relationship with a neighbor—having high values for the above rates) then that neighbor's collaboration utility is high. The fact that the agent is *not* currently collaborating with the neighbor adds to the utility as well. This is to prevent the agent from overloading a particular neighbor with too many requests.

In our current implementation, we define a successful collaboration in terms of the ratio of what is requested of a neighbor $n_{a_i,j}$ by a_i over what is supplied by $n_{a_i,j}$ to a_i. Take our DIR application as an example. Suppose that a_i requests that $n_{a_i,j}$ provide k links (or documents) to satisfy a particular query task, and $n_{a_i,j}$ supplies a_i with k' links. Then the degree of success of that collaboration is k'/k.

2.3. Concept base, translation table, credibility score, and inferencing

An agent a_i's concept base, Γ_{a_i}, consists of a set of concepts. Each concept is composed of a topical term (or keyword) and a set of documents categorized under that topic. In our framework, we assume that each agent is given a concept base to begin with.

Translation table. An agent a_i keeps track of the mappings between the topical terms it knows in its concept base with those of its neighbors in a translation table, Ψ_{a_i}. Each entry in the table records a mapping between a topical term c of agent a_i's and a topical term c_{map} of a neighbor, $n_{a_i,j}$, if such a mapping exists. Each mapping is also associated with a credibility value of the mapping: CV_{map}.

Table 1
An example of a translation table

Concepts/Neighbors	$n_{a_i,1}$	$n_{a_i,2}$	$n_{a_i,3}$	$n_{a_i,4}$
basketball	NBA 0.9	Bball 0.1	NIL	Basketball 0.4
car	NIL	Auto 0.8	Car 0.7	Move 0.2
...				

In our application, we use a single phrase to represent a topical term and use WWW addresses (URLs) as the supporting documents or links. We build the initial concept bases by gathering several students' WWW bookmarks based on a similar technique outlined in [25]. Each bookmark has a title (i.e., a topical term) and a set of links.

Table 1 shows an example of a translation table. In the example, agent a_i has four neighbors, $n_{a_i,1}, n_{a_i,2}, n_{a_i,3}$, and $n_{a_i,4}$. It knows of topical terms such as "basketball" and "car". For "basketball", it is similar to $n_{a_i,1}$'s "NBA" with a credibility of 0.9, $n_{a_i,2}$'s "Bball" with a credibility of 0.3, and $n_{a_i,4}$'s "Basketball" with a credibility of 0.4. However, it does not have a translation for "basketball" between itself and $n_{a_i,3}$.

Credibility score and inferencing. In the beginning, the mapping entries in the translation table are set to NIL and are learned through inference. When an agent a_i realizes that it has not been able to respond to queries regarding a particular concept in a satisfactory manner, it may decide to identify and repair the weak mappings for the concept (e.g., in Table 1, the mapping between the "Basketball" of the agent and "BBall" of neighbor $n_{a_i,2}$ has a credibility value of only 0.1). To do so, a_i sends an inferencing request to that particular neighbor. This request includes the concept that a_i knows (the topical term and the associated documents or links). Since the process is costly in terms of time and resources, a_i only does so carefully. First, it decides to perform an inference when it has failed to satisfy a frequently-encountered query in the past. Second, it employs a stepwise approach. When a_i identifies a problematic query, it does not approach all neighbors simultaneously to ask for an update on each mapping. Instead, it first selects the neighbor with the best collaboration utility and the worst credibility value, indicating a potentially very helpful neighbor with possibly poor, outdated mapping. We assume that

two agents that have collaborated successfully in the past are more likely to have a strong mapping after the inferencing process, and are also more likely to utilize that mapping.

In general, the inferencing process to find a match may be based on induction, clustering, or latent semantic analysis [24]. Our inferencing process is based on an information retrieval approach. Suppose that agent a_i sends an inferencing request to neighbor $n_{a_i,j}$. If the neighbor $n_{a_i,j}$ decides to help (only if it is not overly busy and has an idle thread), then $n_{a_i,j}$ first sets up a connection with the WWW server of each associated document or link provided in the request. It then requests and collects the documents pointed to by these links. We denote this collection the *target* set. In $n_{a_i,j}$'s concept base, each topical term it knows also has an associated collection. The goal of the inference process is to find the concept in $n_{a_i,j}$'s concept base that has the most similar collection of documents to the target set, and then use that similarity to compute the credibility value for the mapping.

The similarity is based on the term frequency and inverse document frequency method, $tf_{i,j} \bullet idf_i$—popular in the area of information retrieval [3]—where $tf_{i,j}$ stands for the *term frequency* of the i-th keyword in the j-th document, and idf_i stands for the *inverse document frequency* of i-th keyword in the entire set of documents. In general, a keyword that occurs only in a few documents is given more weight as it is deemed to be more discriminative. A keyword that occurs more frequently in a document than another keyword is also given a higher weight.

Briefly, to compute the document similarity between two documents, we first perform *stopword filtering* and *stemming*, both standard procedures in information retrieval. Stopword filtering removes common words such as articles and conjunctives from the document. Stemming reduces the remaining words in the document to their root or base forms. Words that remain become the document's list of keywords, and the $tf_{i,j}$ of each keyword is computed. Doing this over all documents in the set, we also obtain the number of hits (i.e., the number of documents that contain a particular keyword) for each keyword. We equate the idf_i of a keyword to the inverse of the number of hits. Thus, multiplying $tf_{i,j}$ and idf_i gives the weight of the i-th keyword, $w_{i,j}$. With this, the j-th document is represented with a vector of keyword weights, $\vec{w}_{i,j} = \langle w_{1,j}, w_{2,j}, \ldots, w_{N,j} \rangle$, where N is the total number of unique keywords in the set of documents.

To compute the similarity of two documents j and k, the cosine product formula is used:

$$sim_{j,k} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2 \sum_{i=1}^{N} w_{i,k}^2}}. \quad (2)$$

The similarity score between two collections of documents, Γ_a and Γ_b, is thus:

$$sim_{\Gamma_a,\Gamma_b} = \underset{r}{avg}\Big(\max_s sim_{\tau_{a,r},\tau_{b,s}}\Big), \quad (3)$$

where Γ_a has R documents and each document is indexed with r, and Γ_b has S documents and each document is indexed with s. To find the correct mapping between a target set specified by agent a_i and the collection of repository sets of neighbor $n_{a_i,j}$, we simply find the $\Gamma_{c_{n_{a_i,j}},m}$ that yields the highest similarity with the target set, $\Gamma_{c_{ir a_i \to n_{a_i,j}}}$. Thus we have $sim_{\Gamma_a,\Gamma_b} = avg_r(\max_s sim_{\tau_{a,r},\tau_{b,s}})$ and $c_{map} = \arg\max_m sim_{\Gamma_{c_{ir a_i \to n_{a_i,j}}},\Gamma_{c_{n_{a_i,j}},m}}$ and the credibility value is:

$$CV_{map} = \max_m sim_{\Gamma_{c_{ir a_i \to n_{a_i,j}}},\Gamma_{c_{n_{a_i,j}},m}}. \quad (4)$$

The neighbor $n_{a_i,j}$ then sends over the mapping such that a_i updates the entry in its translation table accordingly.

The above design thus does not specifically deal with synonyms *per se*; instead, it deals with relevance between two topical terms based on the amount of shared keywords in their respective associated documents.

In our current design, a neighbor that receives an inferencing request will agree to perform the task if it has time or resources to do so. However, since inferencing is expensive, it could be cost-effective for the responding neighbor to negotiate with the requesting agent to reduce the task load. Negotiation issues could include the accuracy needed for the credibility score (if low accuracy is sufficient, then the responding neighbors could examine only a few documents) and the rewards (the requesting agent could offer guaranteed future services in return).

2.4. Interpretation, collaboration, and relays

After an agent a_i decides to entertain a query, q, it compares c_q against its own concept base, Γ_{a_i}. In our current implementation, the interpretation process is simply matching the string c_q to the concepts in Γ_{a_i}. In Section 6, we discuss a key item of our future work—using hierarchical ontologies and partial and relevant matching, as originally proposed in [19].

Collaboration. Suppose that a_i receives a query q with $\langle c_q, \#_q, o_q, s_q \rangle$ and that a_i is only able to satisfy the query partially, providing only $\#'_q$ links or documents matching c_q. Now, a_i needs to find additional $\#^c_q = \#_q - \#'_q$ links to satisfy the query. This collaboration consists of two parts: (1) the identification of specific neighbors from which to ask for help and (2) the allocation of requests to these neighbors.

First, each CUDK agent a_i has N_{a_i} collaboration threads. When an agent asks for help from one of its neighbors, it activates one of its collaboration threads so that such interaction is handled in a thread while the main agent process carries out its other tasks. Hence, the number of neighbors to approach for help is limited by the number of idle collaboration threads, $N^{idle}_{a_i}$, that an agent a_i has at the time of the collaboration.

Given $N^{idle}_{a_i}$, a_i identifies the potential help by examining its translation table, looking for mappings of c_q. First, each neighbor $n_{a_i,j}$ with a non-NIL mapping is a potential source of help. Second, each of these potential help sources is ranked based on the credibility of the mapping and the collaboration utility. If the number of potential help sources is greater than $N^{idle}_{a_i}$, then only the top $N^{idle}_{a_i} - N_{insurance,a_i}$ neighbors are selected to form the collaboration, where $N_{insurance,a_i}$ is the number of threads that each agent a_i reserves to handle requests from other agents. In general, an agent with a higher combined value of collaboration utility and credibility score will have a higher $N_{insurance,a_i}$. After this stage, agent a_i has determined a subset of its neighbors to approach for help.

The second task involves distributing the number of desired links, $\#^c_q$, among the neighbors. Proportionally, agent a_i assigns the number of desired links to request from a neighbor $n_{a_i,j}$, $\#^c_{q,n_{a_i,j}}$, based on $n_{a_i,j}$'s ranking. The higher the ranking, the larger $\#^c_{q,n_{a_i,j}}$ is. This design encourages an agent to prefer the same neighbors for help as long as those neighbors have been useful and credible in the past.

Relays. A relay occurs when agent a_i cannot find a match for a query q, i.e., c_q is not in its concept base. There are two types of relays in CUDK: *targeted* and *generic*.

A targeted relay occurs when agent a_i matches c_q to one of the entries in its translation table. Suppose the entry is $\psi_{c_q,a_i \rightarrow n_{a_i,j}}$. A match occurs when $c_q = c_{map}$. When such a match is found, a_i relays the query to $n_{a_i,j}$. It is possible that $n_{a_i,j}$'s understanding of c_{map} does not match what the query's originator has in mind for c_q. But in view of ignorance on a_i's part, for our current design, the agent simply assumes that $n_{a_i,j}$ would likely return relevant links to the query.

A generic relay occurs when an agent has absolutely no idea what c_q is. In this case, it initiates an exploration with the following principles. First, it starts the exploration conservatively, approaching only a small number of neighbors, thus conserving the collaboration threads that it has. Second, it allocates the number of desired links in the same way as in the collaboration requests to the neighbors. So, if neighbor $n_{a_i,j}$ has been useful and credible, agent a_i will count on that neighbor more for exploration. Consequently, within the general exploration process, there is still a touch of targeted strategies.

To prevent circular relays—i.e., a query going back to its originator—agents have a provision in place such that a neighbor that is also the originator of the query cannot be a potential source of help.

Relay score. To keep track of how well a neighbor handles a relay, we use a metric similar to collaboration utility. Suppose that a_i relays a query to $n_{a_i,j}$ and the query requests k links (or documents), and after the interaction, $n_{a_i,j}$ returns to a_i with k' links. Then the degree of success of the relay is k'/k. The relay score of $n_{a_i,j}$ from the viewpoint of a_i is the average of k'/k for all relays from a_i to $n_{a_i,j}$.

3. Implementation

In this section, we present briefly the agent implementation for the application of CUDK to DIR. As shown in Fig. 3, a CUDK agent has eight key modules. Together with these eight modules are three dynamic

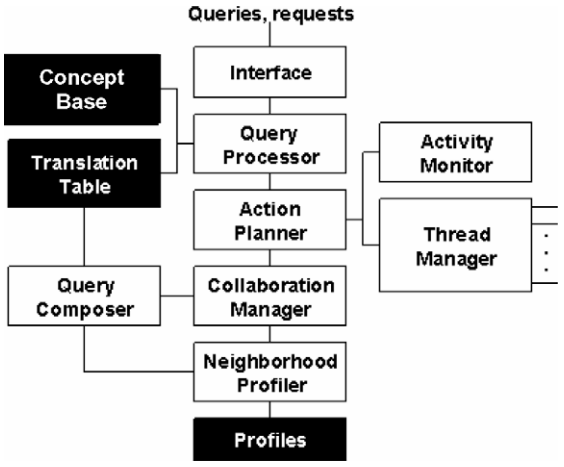

Fig. 3. The current design of the operational components of an agent in our framework.

knowledge bases or databases: a concept base, a translation table, and a set of neighbor profiles.

(1) **Interface**: This module interacts with the user to obtain queries and to provide query 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.
(2) **Query Processor**: This module receives a query from the Interface module and processes it. It first checks the agent's concept base. If the query matches one of the topical terms in the concept base, the module retrieves the number of links available. If the query does not find a match in the concept base, the module examines its translation table. If there are available translations, then that means a collaboration is possible.
(3) **Action Planner**: This 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; otherwise, if (b) the agent understands the query and finds available translations, it initiates its collaborative activities (as discussed in Section 2.3); or if (c) the agent does not understand the query, it will relay the query to another agent (as discussed in Section 2.4). Whether a 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 a collaboration, the query satisfaction process terminates. If it receives an inferencing request, it also decides whether to help as discussed in Section 2.3. If it helps, it carries out the inferencing using Eqs. (2) and (3) as discussed.
(4) **Collaboration Manager**: When the action planner calls for a collaboration, this module takes over. The objective of this module is to select a group of neighbors to approach and distribute the query demands (link allocations) among them accordingly. To design such a collaboration plan, this module relies on the neighborhood profile and the translation table. Each neighbor is tagged with a collaboration utility and a translation credibility score (Eq. (1)). The collaboration manager ranks these neighbors based on the two measures and composes the query demands accordingly, with the help of the Query Composer. The manager assigns more links to neighbors with higher ranking proportionally to maximize the chance of retrieval success, as discussed in Section 2.2. It also collects the query results and filters out low-credibility links when it has more links than desired.
(5) **Query Composer**: Based on the allocation of query demands, this module composes a specific query for each neighbor to be approached. 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.
(6) **Neighborhood Profiler**: Each time a collaboration is completed, this module updates its profile of the neighbor. For example, if it was a successful collaboration, this module increments the number of successful collaborations between the agent and the particular neighbor by one.
(7) **Activity Monitor**: This module keeps track of the activities in a *job vector*—whether the agent is processing a query on its own, or collaborating with other neighbors for more links, or entertaining a request from a neighbor. Each *job* is described with a list of attributes such as the originator, the executor, the task description, the current status, and so on. Also, if the agent encounters a particular query that it has frequently failed to satisfy, it triggers an inferencing request to its neighbors, as discussed in Section 2.3.
(8) **Thread Manager**: This module manages the threads of the agent. It is a *low*-level module that activates the threads and updates and monitors the thread activity.

We have implemented all eight modules of our agent as depicted in Fig. 3 in C++. Each agent receives its user queries from a software user through a socket connection and communicates with other agents using a central communication module through socket connections as well.

4. Experiments and results

The following experiments were designed to answer the following questions:

(1) How do operational and conceptual constraints together impact the query results in our multiagent DIR application? The operational con-

straints considered are time and collaboration threads. The conceptual constraints considered are the concept bases and the translation tables. The query results are measured in terms of content quality and time taken to satisfy a query.

(2) How do query tasks affect the query results in our multiagent information retrieval system? Here, we look at different segments of query tasks, designed to incur different environmental stresses on the agents.

(3) Does the profiling module (one of the two learning mechanisms) help improve the query satisfaction task? Answering this question will allow us to refine our profiling module, which could lead to a better design of our credibility score and collaboration utility.

(4) Does the inferencing mechanism (one of the two learning mechanisms) help improve the query satisfaction task? Answering this question will give us insights to build a better decision making process that balances costly inferencing acts with services to user queries.

4.1. Experimental setup

There are five agents supporting one software user each. All agents are neighbors and can communicate among themselves. Every agent has a unique set of nine concepts in its repository. Each concept has five supporting links. Each agent has an initial 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 mapping is not available, we use the symbol NIL.

Each software user has a query configuration file. Thus, instead of us manually submitting these queries, each software user automatically retrieves a query at a time from the configuration file and sends it to its agent. For each query in a configuration file there are (a) a cycle number, (b) the queried concept name, and (c) the number of links desired. The cycle number indicates when the query is to be submitted to the agent. Figure 4 gives an overview of the first batch of query segments (Table 4 in Section 4.4 describes additional attributes of these query segments):

(1) Cycles 0–10: Every software user queries about all different concepts its agent has in the concept base. Each agent is also able to satisfy the query demand on its own. During this segment, each agent does not need to collaborate. All queries across the users are submitted at the same cycles.

(2) Cycles 11–40: Every software user queries about all different concepts its agent has in the concept

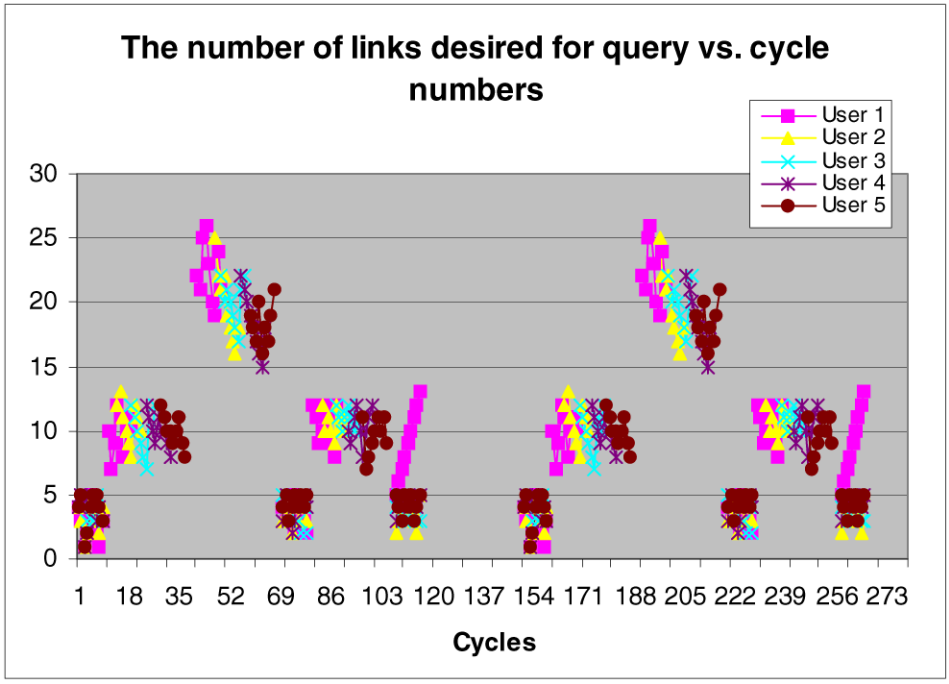

Fig. 4. The number of links for the queries submitted by the software users to the agents for each cycle.

base. However, each agent is not able to fulfill all queries on its own. During this segment, each agent needs to collaborate. All queries across the users are submitted in a staggered manner. User 1 submits all its nine queries first; user 2 submits its queries after 3 cycles; and so on.

(3) Cycles 41–70: Every software user queries about all different concepts its agent has in the concept base and each agent is not able to satisfy the queries on its own. Also, the number of links desired for every query is twice that in the second segment. *Extensive* collaborations are needed. Queries are also staggered in this segment.

(4) Cycles 71–80: Every software user queries about different concepts its agent does *not* have in its concept base. This forces the agent to relay the queries to its neighbors. Queries are packed and not staggered in this segment.

(5) Cycles 81–110: The setup of this segment is similar to that during cycles 11–40, but with concepts that each agent does not have in its concept base. Queries are staggered.

(6) Cycles 111–120: During this segment, two users query about concepts that their agents do not have in their respective concept bases, two software users query about only some concepts that their agents do not have in their respective concept bases, and one software user queries about concepts that its agent has in its concept base. The queried number of links is small and no collaborations are needed.

Our query segments are staggered (e.g., the third segment) and packed (e.g., the first segment) to investigate the response behaviors of the agents. Since the number of collaboration threads is limited for each agent, packed queries with high link demands may lead to only partial link retrievals. Our query segments also come with low and high link demands. Low link demands do not require any or require fewer collaborations, while high link demands prompt the agents to collaborate more. Finally, an agent may or may not know some of the queried concepts. The agent's concept base specifies this knowledge. When an agent knows the queried concept, it has more options, approaching different neighbors for help. When it does not know the queried concept, then it shifts the responsibility to one of the neighbors, essentially making itself a relay station.

Given the above query segments, we further vary two sets of parameters: (1) *operational constraint*: the number of collaboration threads, and (2) *conceptual constraint*: the credibility values in the translation tables. When the number of collaboration threads is small, an agent cannot afford to contact many neighbors simultaneously. Thus, this limits the opportunities to perform inferencing and entertain requests. In addition, the agents are supplied with different sets of translation tables for different experiments. For example, 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 is termed as a "*narrow translation*." That is, its translation table contains many NIL mappings, above 50%. In the third set, two agents have narrow translations. 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 agents with poor conceptual mappings learn to cope with query satisfaction.

We also collect the following parameters from our agents:

(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 collaboration utility measure, as described in Section 2. Table 2 documents the definitions of these parameters.

(2) **Query Result Parameters**: For each query, an agent collects parameters documenting the characteristics of the query and the query outcome. Table 3 documents the definitions of these parameters.

4.2. Analysis 1: Impact of operational constraints

We analyze the impact of operational constraints on how CUDK agents collaborate in our DIR application. The operational constraints considered are time and collaboration threads. The query results are measured in terms of content quality and time required to satisfy a query.

Figure 5 shows the average $\_successQuality$ of the queries (averaged over all queries) vs. the number of threads, for each software user.

Here are some observations:

- The average $\_successQuality$ of a user's queries increases as expected when the number of threads increases. This is because for high-demand queries

Table 2

Neighborhood profile parameters

Parameters	Definitions
$\_numSuccess$	The number of successful collaborations that the agent has initiated to neighbor i
$\_numHelp$	The number of successful collaborations that the agent has received from the neighbor i
$\_numRequestTo$	The total number of collaborations that the agent has initiated to the neighbor i
$\_numRequestFrom$	The total number of collaboration requests that the agent has received from neighbor i
$\_successRate$	$\_numSuccess/\_numRequestTo$
$\_helpRate$	$\_numHelp/\_numRequestFrom$
$\_requestToRate$	$\_numRequestTo/\_totalRequestTo$ where $\_totalRequestTo$ is the sum of all collaborations that the agent has initiated
$\_requestFromRate$	This number tells the agent how much neighbor i relies on the agent

Table 3

Query result parameters

Parameters	Definitions
$\_originator$	The originator of the query, either from a software user (ID) or another agent
$\_cycle$	The cycle ID when the query is first generated
$\_numLinksDesired$	The number of links desired by the query
$\_numLinksRetrieved$	The number of links retrieved at the end of the retrieval process and presented to the user, always smaller than or equal to $\_numLinksDesired$
$\_conceptName$	The query keyword
$\_successQuality$	$numLinksRetrieved/numLinksDesired$
$\_duration$	The actual elapsed time between the receipt of a query and the presentation of the query results to the user
$\_listLinks$	The list of links retrieved and presented to the user at the end of the retrieval process

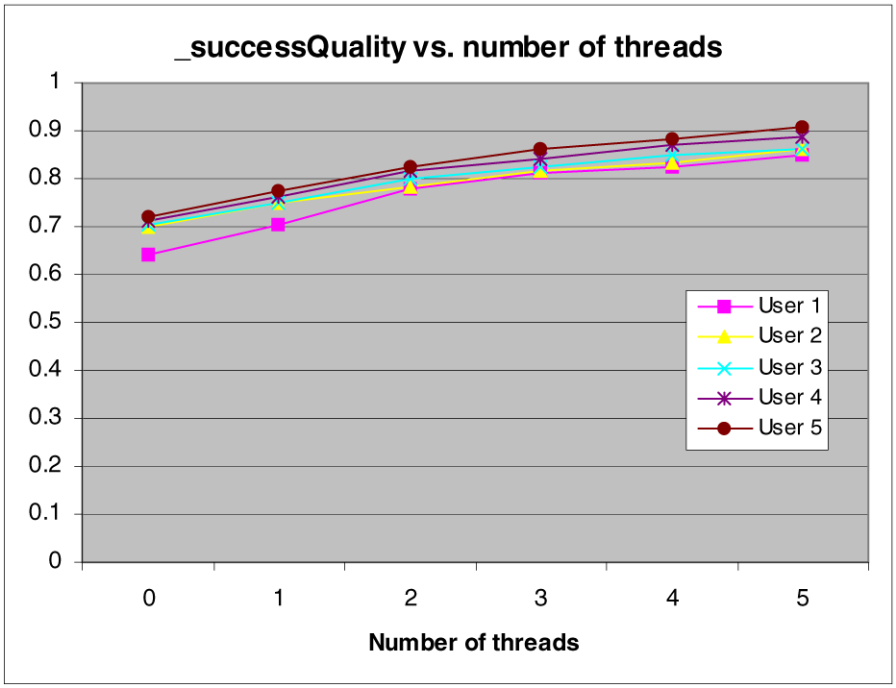

Fig. 5. The average $\_successQuality$ value of each user's queries vs. the number of threads that the agents have.

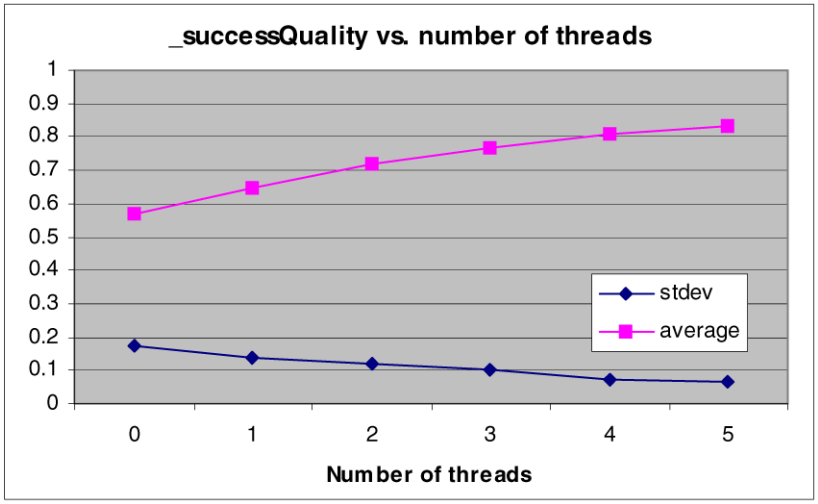

Fig. 6. The average and standard deviation of the $\_successQuality$ for all agents vs. the number of threads.

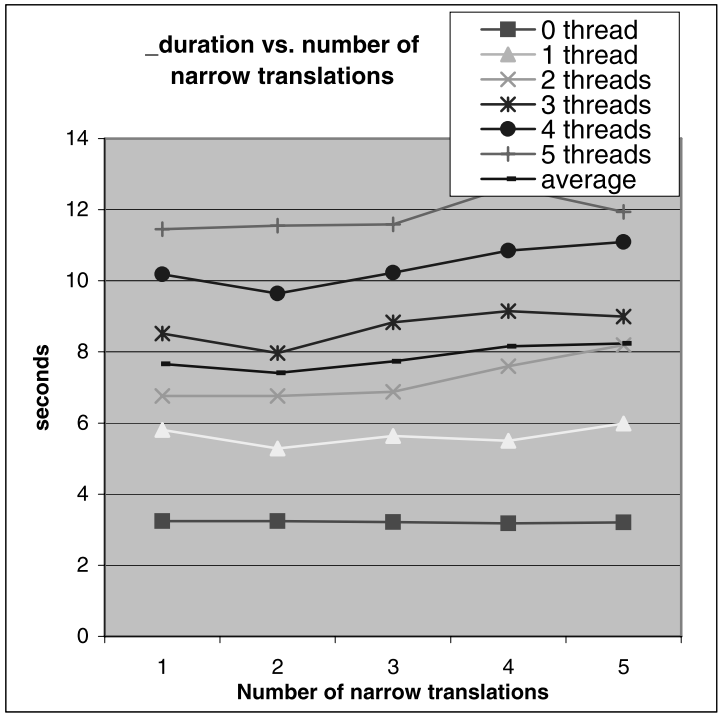

Fig. 7. The average $\_duration$, for different numbers of threads, vs. the number of narrow translations.

that call for collaborations, the agent has more resources (i.e., collaboration threads) to use.

- Figure 6 shows the average $\_successQuality$ and standard deviation of all queries for each number of threads. As we can see, with a higher number of negotiation threads, queries are satisfied more successfully (high average values), and also more consistently (low standard deviation values).
- Figure 7 shows the average $\_duration$ (in seconds) for each query to be processed and presented back to the user, for different numbers of collaboration threads. As observed, when the

number of threads increases, it takes longer for a query to be satisfied. Though this observation was not anticipated initially, 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 entertains 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.

Based on the above, we conclude the following. Though an increase in the number of threads improves query satisfaction in terms of retrieved documents and consistency, the query satisfaction performance in terms of time spent for retrieval process deteriorates. This has several implications. First, when the number of threads is high, the system performs better, and thus the agents have less motivation to improve learning of their concept bases (i.e., the translation tables). Second, when the number of threads is high, the agents slow down. To address the slowdown in query satisfaction, we realize that the agents should be conservative in their collaborations—instead of asking many neighbors for help, the agents should ask only a few top-ranked neighbors for help. This will allow an agent to complete a query task more quickly. Further, if an agent views the slowdown as partial failure, then the agent will indeed have motivation to learn to improve its translation table. Therefore, having more threads is both a liability and an advantage. How an agent manages the thread resources will have a significant impact on the way the agents learn about each other's concepts. This also implies that the role of an accurate neighborhood profile will be important since an agent has to be sure that the quality of help it receives from a reduced number of neighbors is good.

4.3. Analysis 2: Impact of conceptual constraints

In this analysis, we focus on the conceptual constraints imposed by the "narrow translations" defined in Section 4.1. We aim to investigate how poor initial mappings impact how agents collaborate in the system, and how that leads to the need for neighborhood profiling and conceptual inferencing.

- From Fig. 7, the average $\_duration$ values for the different numbers of narrow translations are 7.66, 7.41, 7.73, 8.15, and 8.24 seconds, respectively. We see that there is an increasing trend in the time spent to satisfy queries as the number of narrow translations increases. This is to be expected as agents are required to collaborate more often, incurring more time cost as the number of narrow translations increases
- Figure 8 shows the $\_successQuality$, for different numbers of narrow translations and threads, over the different sets of queries in terms of the numbers of desired links. As expected, the $\_successQuality$ drops as more links are desired. However, we see that the conceptual constraint is offset by an increase in agent resources (i.e., the number of collaboration threads).

Comparing Fig. 8 with the figures in Section 4.2, we see that operational constraints impact the system more significantly than do conceptual constraints: the number of narrow translations impacts the success quality insignificantly compared to what we have found in Section 4.2 about the number of threads. This was unexpected. Upon closer analysis, we see that the conceptual disadvantage in some agents can be compensated with neighborhood profiling and collaboration rather successfully. Thus, we see that the *motivation for agents to learn each other's concepts is likely to be more resource-related than concept-related*, at least in our CUDK framework and our DIR application. This also hints that with good neighborhood profiling and collaboration, agents with poorer initial concept bases do not necessarily perform more poorly than agents with better initial concept bases.

4.4. Analysis 3: Impact of query tasks

In this analysis, we investigate the impact of query tasks. Our objective is to find out how various combinations of query tasks stress the collaboration activities. For example, if the queries are packed and presented all at once to the agents, will the agents be able to still collaborate successfully?

Particularly, we want to investigate how the agents handle the different *segments* of queries. Briefly, there are six segments, as described in Section 4.1, in each batch of queries. Segment 1 is the least demanding in terms of the number of links or documents desired for each query. Segment 2 consists of queries that lead to every agent having to collaborate with its neighbors. Also, the queries are submitted in a staggered manner. Thus, the agents are not flooded with all their queries at the same time. Segment 3 is similar to Segment 2, but with far more demanding queries in terms of the number of links desired. In Segment 4, the queries

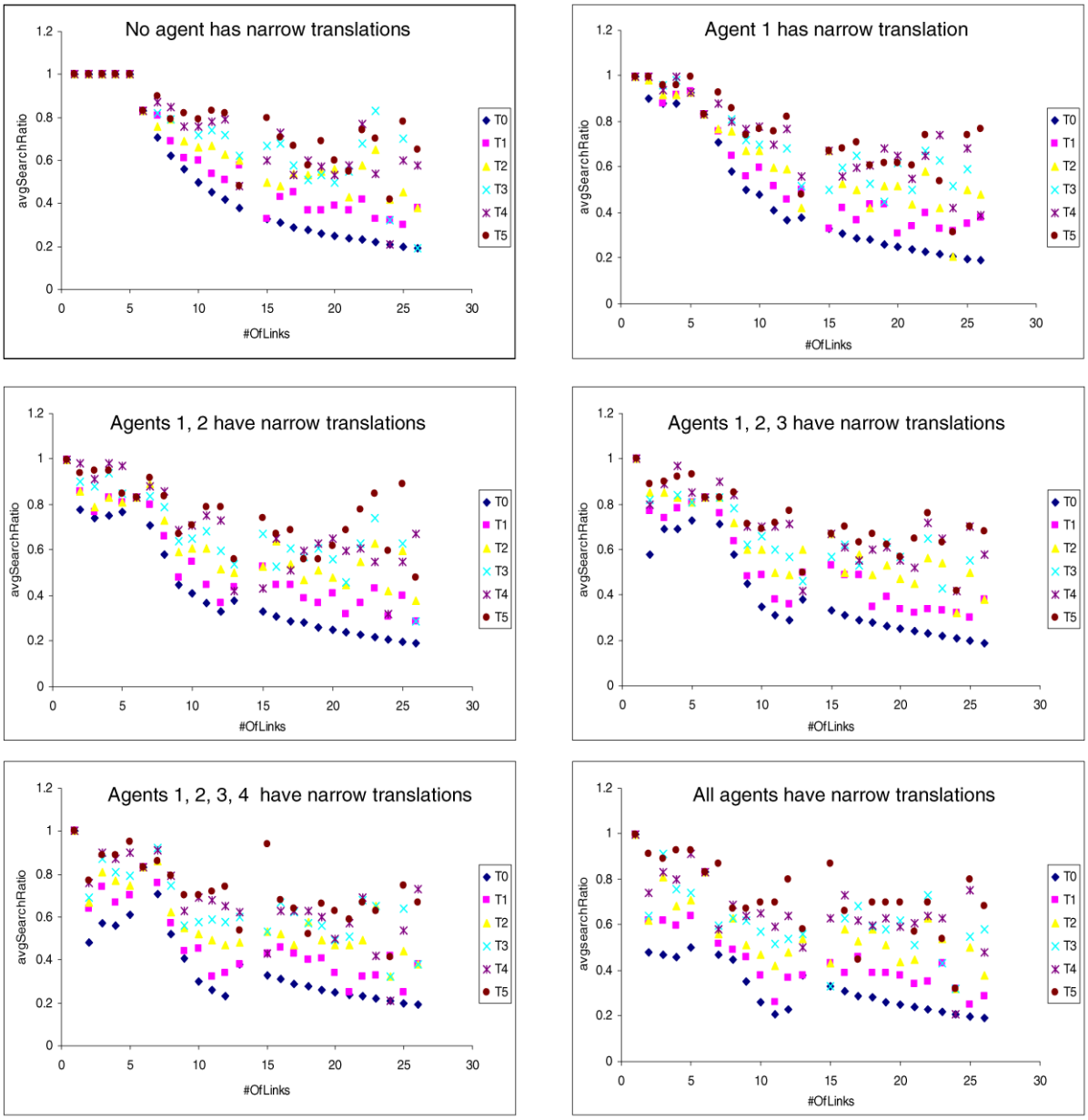

Fig. 8. *_successQuality* for queries of different link demands, for systems with different numbers of narrow translations.

are intentionally submitted to the agents that do not have links or documents to satisfy the queried concepts. These queries are also packed to induce communication congestion in the system as well as resource contention for negotiation threads within each agent. Segment 5 is similar to Segment 4 but with staggered submissions and thus is less constrained. Segment 6 is a mixture of all the above characteristics.

In addition, we identify eight attributes to describe each segment (see Table 4):

(1) *_cooperationNeeded*: indicating whether an agent needs to collaborate with its neighbors to satisfy the queries in the segment.
(2) *_numCycles*: the duration of the segment.
(3) *_queryCompactness*: the ratio of the number of queries occurring at the same cycles to the total number of queries in the segment.
(4) *_queryDensity*: *_queryCompactness* normalized by *_numCycles*.

Table 4

Description of the six different segments. Abbreviations: cN = *_cooperationNeeded*, nC = *_numCycles*, qC = *_queryCompactness*, qD = *_queryDensity*, aL = *_aveNumLinks*, maL = *_maxNumLinks*, miL = *_minNumLinks*, and kR = *_knowledgeRatio*

Segment	cN	nC	qC	qD	aL	maL	miL	kR
1	N	9	1	0.11	3	5	1	1
2	Y	27	0.8	0.03	10	12	7	1
3	Y	27	0.8	0.03	20	26	15	1
4	Y/N	9	1	0.11	4	5	2	0.02
5	Y	26	0.8	0.03	11	12	7	0.00
6	Y/N	9	1	0.11	5	13	2	0.54

(5) *_aveNumLinks*: the average number of links desired per query in the segment.
(6) *_maxNumLinks*: the maximum number of desired links of a query in the segment.
(7) *_minNumLinks*: the minimum number of desired links of a query in the segment.
(8) *_knowledgeRatio*: the ratio of the number of queries submitted to the agents that know the requested concepts over the number of total queries in the segment.

In general, if a segment requires collaboration, with a larger number of queries for an agent, higher compactness, lower number of cycles, higher query density, and higher number of links per query, and lower knowledge ratio, then we expect the system to perform less successfully. From the experiments, we observe the following:

- Figure 9 shows the average *_successQuality* values for all segments, for different numbers of threads and narrow translations. We see similar observations as those drawn from Fig. 8.
- Table 5 shows aggregate results of two sets of query segments grouped based on the levels of their *_knowledgeRatio*. The *_knowledgeRatio* value impacts the average *_successQuality* value: agents with higher *_knowledgeRatio* values achieve higher *_successQuality* values, especially when the number of threads is small (0 or 1). As the number of threads increases, the impact of *_knowledgeRatio* decreases. Also, as the number of agents with narrow translations increases, the number of threads factors more significantly into the *_successQuality* values of the agents with low *_knowledgeRatio* values. In general, when the knowledge ratio is low, a high number of threads—increased resources for collaborations and relays—can maintain a level of system performance very similar to that achieved when the knowledge ratio is high.
- Table 6 shows aggregate results of two sets of query segments grouped based on the levels of their *_queryDensity*. The impact of the *_queryDensity* value of the segments on the query results was not expected. We expected that the *_successQuality* would be high when the *_queryDensity* is low. However, this is not the case. Actually, the segments with a high query density (1, 4, and 6) have significantly higher *_successQuality* than do the other segments. Following this observation, we ran another experiment and the results are shown in Figs. 10 and 11. This new experiment shows that the agents learned to respond to queries faster when *_queryDensity* is higher, and they contact fewer neighbors. This is an interesting observation as *the higher query demands forced the agents to learn more quickly and improve their use of resources*. A faster response time frees up threads for collaborations; contacting fewer neighbors also frees up more threads for other collaborations and frees up the neighbors' threads. As a result, agents with handling query segments of high *_queryDensity* are able to produce better performance.
- Table 7 shows the standard deviation values of the average *_successQuality* for different numbers of narrow translations and threads. We see that the system performance is slightly less consistent—with larger standard deviations—when the agents have fewer threads (this coincides with Fig. 6) and also when the agents have more narrow translations. Table 8 shows the standard deviation values of the average response time for different numbers of narrow translations and threads. We see that the system's performance in terms

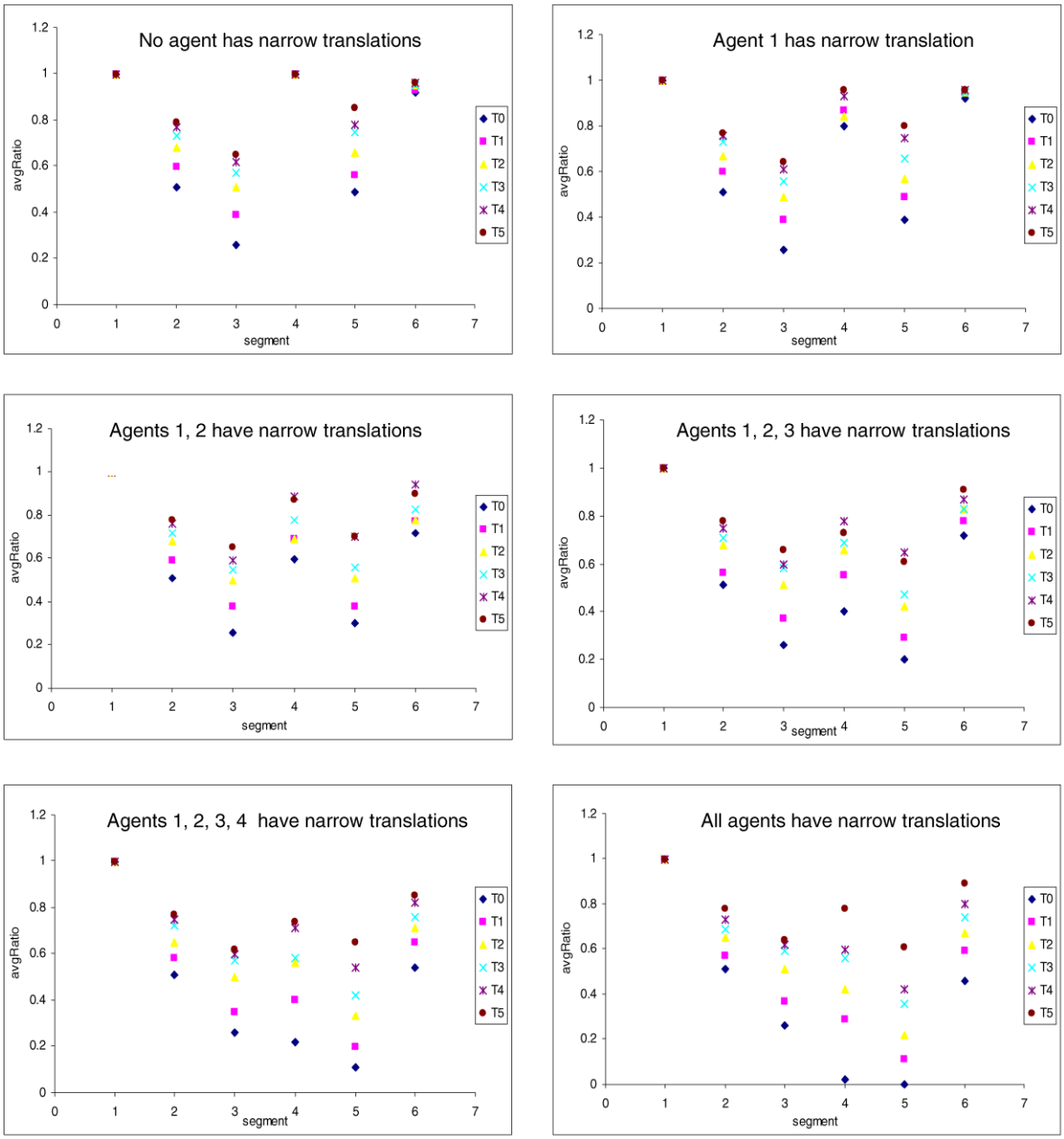

Fig. 9. Average $\_successQuality$ vs. Segment.

of response time becomes significantly more inconsistent—with larger standard deviations—when the agents have more threads. Combining the two observations, we see a key tradeoff. If we want to have a *reliable* and *predictable* system in terms of both the goal achievement (i.e., query satisfaction in this case) and the time it takes to achieve the goal, then having more threads does not help. Further, we find that collaborations are more consistent than targeted relays; and targeted relays are more consistent than generic relays. This is because agents responding to relays are more persistent since they have more threads to approach their neighbors. This implies that having good collaboration and relay mechanisms are not sufficient. Though these mechanisms help improve system performance, they do not help stabilize the system.

Table 5

Average $\_successQuality$, comparing segments 1, 2, 3 (high $\_knowledgeRatio$) with segments 4, 5, 6 (low $\_knowledgeRatio$) to analyze the impact of $\_knowledgeRatio$

# Narrow Translations	segment	T0	T1	T2	T3	T4	T5
0	1,2,3	0.59	0.66	0.73	0.77	0.8	0.81
0	4,5,6	0.80	0.83	0.87	0.9	0.91	0.94
1	1,2,3	0.59	0.66	0.72	0.76	0.79	0.80
1	4,5,6	0.70	0.77	0.79	0.85	0.88	0.91
2	1,2,3	0.59	0.66	0.73	0.76	0.78	0.81
2	4,5,6	0.54	0.61	0.66	0.72	0.84	0.82
3	1,2,3	0.59	0.64	0.73	0.76	0.78	0.81
3	4,5,6	0.44	0.54	0.64	0.66	0.77	0.75
4	1,2,3	0.59	0.64	0.72	0.76	0.78	0.8
4	4,5,6	0.29	0.42	0.53	0.59	0.69	0.75
5	1,2,3	0.59	0.65	0.72	0.76	0.78	0.81
5	4,5,6	0.16	0.33	0.44	0.55	0.61	0.76

Table 6

Average $\_successQuality$, comparing segments 2, 3, 5 (low $\_queryDensity$) with segments 1, 4, 6 (high $\_queryDensity$) to analyze the impact of $\_queryDensity$

# Narrow Translations	segment	T0	T1	T2	T3	T4	T5
0	2,3,5	0.42	0.52	0.62	0.68	0.72	0.76
0	1,4,6	0.97	0.98	0.98	0.98	0.99	0.99
1	2,3,5	0.39	0.49	0.58	0.65	0.71	0.74
1	1,4,6	0.91	0.94	0.93	0.96	0.96	0.97
2	2,3,5	0.36	0.45	0.56	0.61	0.68	0.71
2	1,4,6	0.77	0.82	0.82	0.87	0.94	0.92
3	2,3,5	0.32	0.41	0.54	0.59	0.67	0.68
3	1,4,6	0.71	0.78	0.83	0.84	0.88	0.88
4	2,3,5	0.29	0.38	0.49	0.57	0.63	0.68
4	1,4,6	0.59	0.68	0.76	0.78	0.84	0.86
5	2,3,5	0.26	0.35	0.46	0.55	0.59	0.68
5	1,4,6	0.49	0.63	0.70	0.77	0.80	0.89

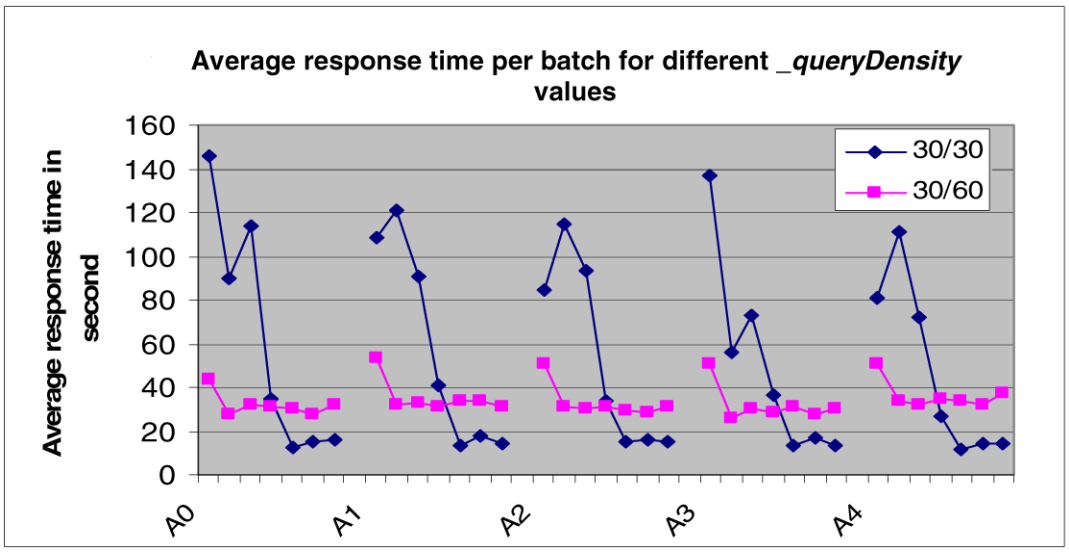

Fig. 10. Average response time per batch of segments for different $\_queryDensity$ values. $30/30 = 30$ queries in 30 cycles, high density; $30/60 = 30$ queries in 60 cycles, low density.

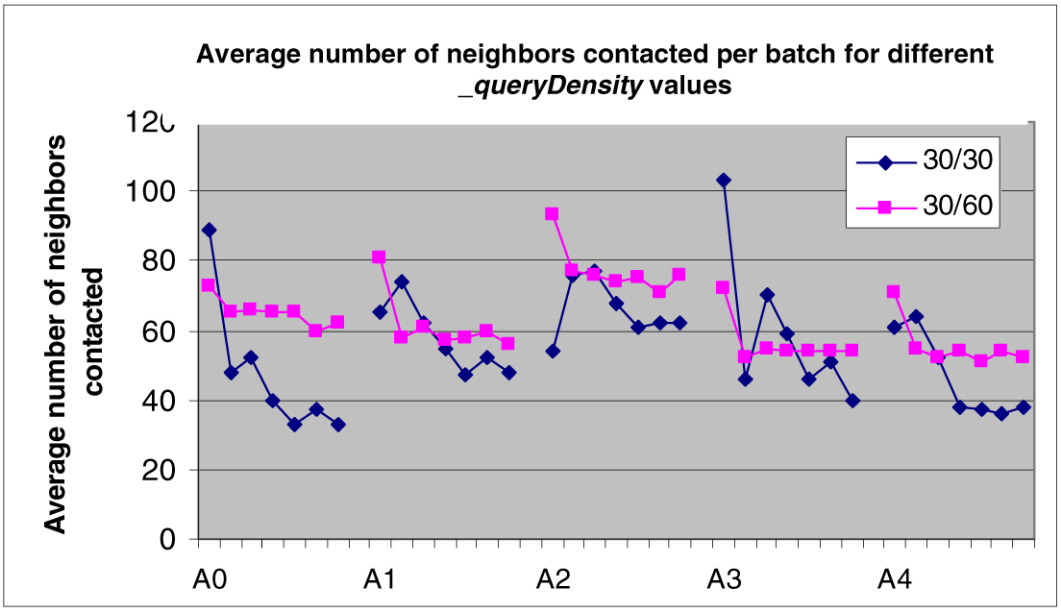

Fig. 11. Average number of neighbors contacted per batch of segments for different $\_queryDensity$ values. $30/30 = 30$ queries in 30 cycles, high density; $30/60 = 30$ queries in 60 cycles, low density.

Table 7

Standard deviation for average $\_successQuality$ by combining all six segments, for different numbers of threads and narrow translations

# Narrow Translations	T0	T1	T2	T3	T4	T5
0	1.94	1.92	1.90	1.88	1.87	1.87
1	1.96	1.93	1.92	1.89	1.88	1.87
2	1.99	1.96	1.94	1.92	1.89	1.89
3	2.02	1.98	1.94	1.93	1.90	1.90
4	2.05	2.01	1.97	1.95	1.92	1.91
5	2.08	2.03	1.99	1.95	1.94	1.90

Table 8

Standard deviation for average response time by combining all six segments, for different numbers of threads and narrow translations

# Narrow Translations	T0	T1	T2	T3	T4	T5
0	2.10	2.38	2.87	3.45	4.36	4.89
1	1.67	2.21	2.54	3.45	4.20	5.17
2	1.38	1.85	2.42	3.16	3.90	4.88
3	1.33	1.8	2.33	3.31	4.05	4.73
4	1.28	1.79	2.79	3.62	4.16	5.30
5	1.27	1.96	2.75	3.31	4.30	5.07

Based on the above, we conclude the following. Out of the eight attributes that we use to describe the various segments, only $\_knowledgeRatio$ plays a significant role on the system's performance in query satisfaction. We also see that agents equipped with more resources (i.e., threads) are able to address the conceptual constraints through collaboration and relay mechanisms. However, more resources also create a less predictable system in terms of the time spent on each query. Reducing relays could help since they contribute most significantly to the inconsistency. To reduce relays, conceptual inferencing is a very viable approach. We also observe that (1) transfers of conceptual knowledge may improve the system's performance, and (2) referrals of queries may improve the system's performance. When we transfer conceptual knowledge from an agent a_i to another agent a_j, a_j becomes knowledgeable. This is particularly useful when a_j has few resources available. However, a_i's uniqueness will decrease, as will the diversity of the system as a whole. When we refer queries from an agent a_i to another agent a_j, a_i basically transfers one of its users to another agent. It is possible that a_i eventually becomes a *relay* station for a_j and thus loses its autonomy. Therefore, combining the results from Sections 4.2 and 4.3, we see that conceptual constraints play a very important role on our CUDK agents if the agents do not have enough resources to collaborate, or if the resources are both disadvantageous and advantages at the same time. This could serve as the underlying motivation for agents to learn conceptually for our tier-2 research and design.

4.5. Analysis 4: Impact of neighborhood profiling

The objective of this analysis is to investigate whether and how the profiling module helps improve the query satisfaction task. We want to find out whether the profiling is able to help an agent build better collaborations faster and achieve better query results. Knowing how to profile more accurately also leads to a better design of collaboration utility.

Here are the typical observations, showing the results of one agent, a_1:

- Figure 12 shows the average neighbor profile of agent a_1 of its neighbors: $\_numSuccess$, $\_numHelp$, $\_numRequestTo$, and $\_numRequestFrom$. For a_1, the number of times it has requested help is smaller than the number of times it has entertained other agents' requests. This indicates that the query segments tend to induce collaborations, causing the originating agents to ask for help from many different neighbors. From the graph, we see that the agent approaches more neighbors for help when it has more collaboration threads.
- Figure 13 shows the average $\_successRate$ vs. the number of threads available. As observed, the agent is able to collaborate more successfully when the number of threads increases. This is expected since with more threads available, the agent's neighbors are able to entertain more requests. Coupling this with Fig. 9, we see that a_1 is able to conduct *more* collaborations *more* successfully when the number of threads increases—and to do so more effectively and more efficiently.
- Figure 14 shows the $\_requestToRate$ vs. the number of threads available. As observed, when the number of threads is 1, agent a_1 relies on agent a_2 (or $n_{a_1,1}$) almost completely. This is due to the fact that in the beginning of an agent, all neighbors are weighted very similarly; as a result, the agent will approach the first neighbor that it knows. However, as the number of threads increases, the agent is able to collaborate more with other neighbors. As a result, the reliance on $n_{a_1,1}$ greatly decreases. Meanwhile, the reliance on the other three neighbors steadily increases.

Based on the above, we conclude the following. More collaboration threads mean more collaborations and more successes. We also see that an agent collaborates more successfully (with a higher success rate) as it has more threads. Further, the reliance of an agent on its neighbors is distributed more evenly as it has more threads. These three observations indicate that our CUDK agents are able to profile their neighbors, learn about the good neighbors, and seek them out for subsequent collaborations. The implicit reinforcement learning takes place here: an agent will request help from a neighbor that has been helpful in the past. This gives us a mechanism to identify neighbors

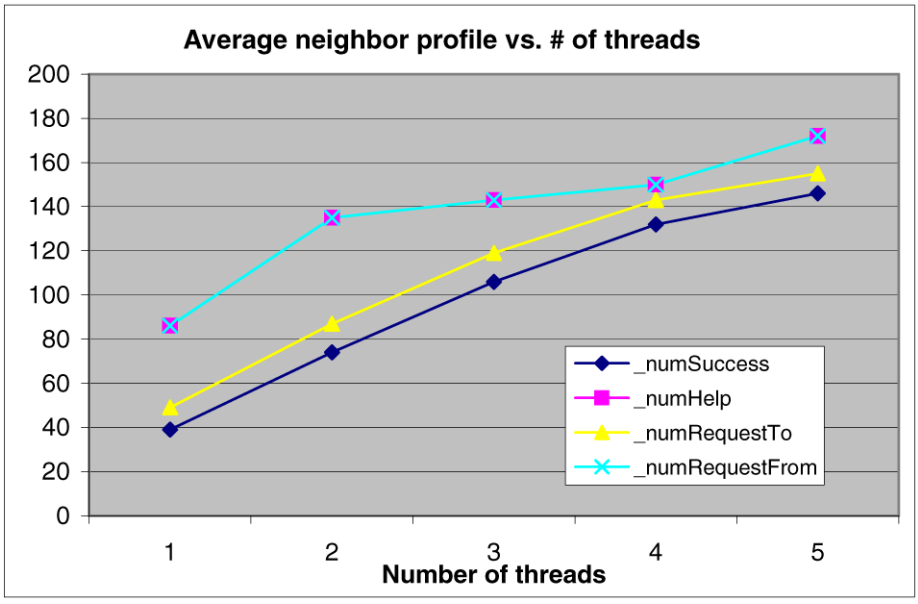

Fig. 12. The average neighbor profile for agent a_1 of its neighbors vs. the number of threads.

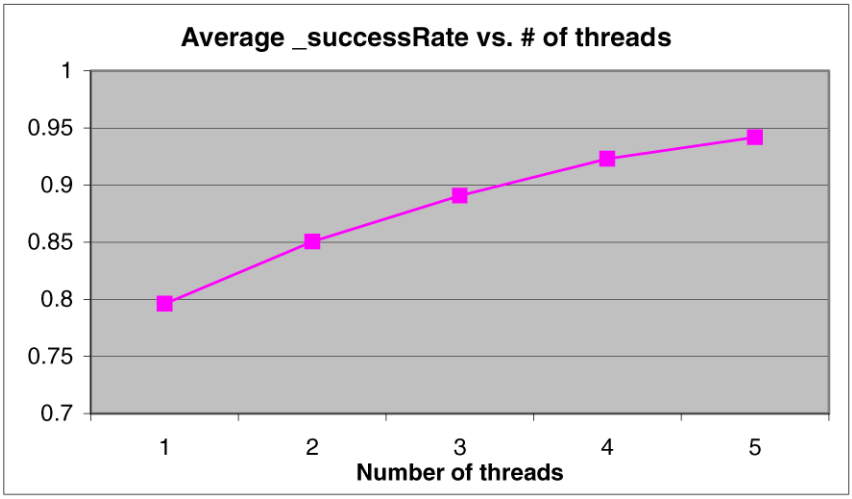

Fig. 13. The average neighbor profile for agent a_1 of its neighbors vs. the number of threads.

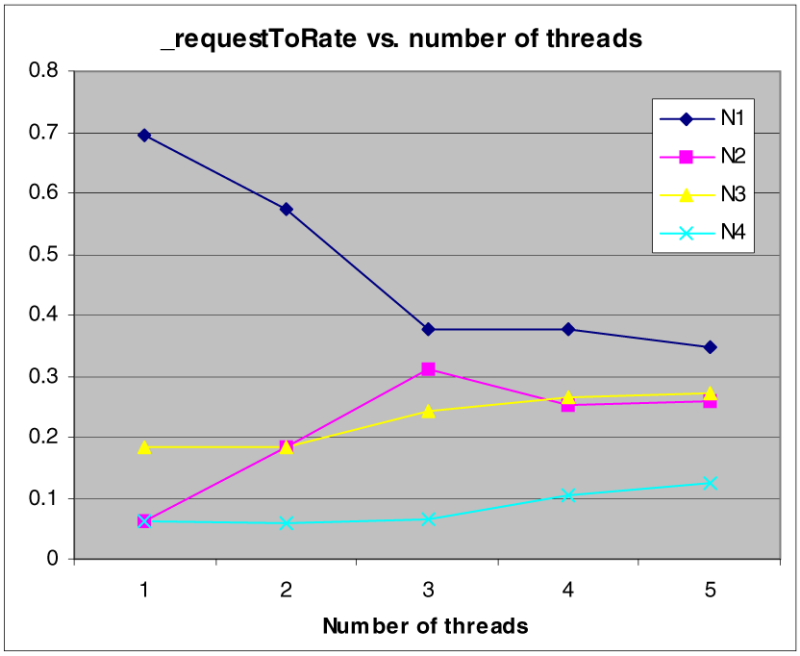

Fig. 14. The $\_requestToRate$ from agent a_1 to its neighbors, N1 ($n_{a_1,1}$), N2 ($n_{a_1,2}$), N3 ($n_{a_1,3}$), and N4 ($n_{a_1,4}$) vs. the number of threads.

whose concept bases are more important for an agent to understand—for our future tier-2 work, we can utilize this relationship to perform cost-effective conceptual inferencing. The neighborhood profile empowers an agent to strategically select a subset of its neighbors to perform conceptual inferencing, thus improving the overall system performance.

Query-triggered collaborations. The queries that an agent encounters trigger collaboration requests, including the targeted and generic relays. Since queries trigger different types of collaborations, an agent learns differently as well. We identify six collaboration types that an agent might encounter during its query satisfaction process as shown in Table 9.

Type-3 and -4 collaborations are situations in which the agent cannot approach potentially helpful neighbors for help because it does not have available collaboration threads. Further, Type-2, -5, and -6 col-

Table 9

Types of collaborations triggered by queries based on what an agent knows and what resources it has available

Type	Knows the queried concept?	Has enough documents/links?	Has idle threads?	Has entry in translation table?	Actions
1	Yes	Yes	Don't Care	Don't Care	No collaboration; return documents/links
2	Yes	No	Yes	Don't Care	Collaboration; compose and return documents
3	Yes	No	No	Don't Care	No collaboration; return documents
4	No	No	No	Don't Care	No collaboration; return nothing
5	No	Don't Care	Yes	Yes	Targeted relay
6	No	Don't Care	Yes	No	Generic relay

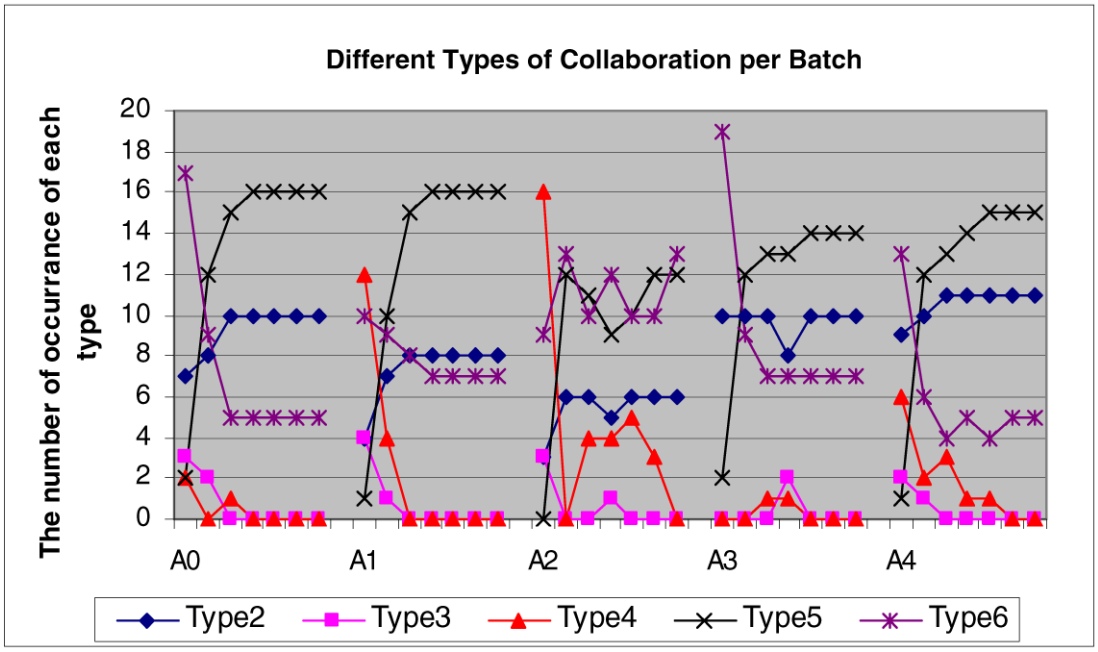

Fig. 15. The number of occurrences of different types of collaborations over time.

laborations are situations where the agent has the resources to carry out query collaborations, indicating that it is operationally capable. A good multiagent system should reduce Type-3 and -4 collaborations and increase Type-2, -5, and -6 collaborations. Reducing the numbers of Type-3 and -4 collaborations indicates that the agents are able to better utilize their resources and avoid fruitless requests for collaboration. Increasing the numbers of Type-2, -5, and -6 collaborations, on the other hand, indicate that the agents are able to identify helpful and useful neighbors.

Figure 15 shows the numbers of different types of collaborations in each batch for the five agents. As learning progressed over time, the number of Type-5 collaborations (targeted relays) increased because the agents gradually learned what the other agents knew and what they themselves did not know through conceptual inferencing. Further, the number of Type-6 collaborations (generic relays) decreased because the agents became more knowledgeable about the other agents' concept bases. Thus, the agents became more responsible in asking for help—in essence, they engaged in less "spamming". The number of Type-2 collaborations remained the same as the local concept base of each agent did not change. Best of all, the numbers of Type-3 and -4 collaborations (situations where no idle threads were available for collaborations) significantly decreased. This indicates that the agents were able to learn to use their resources effectively.

The performance of Type-2 and -5 collaborations were significantly improved by the profile-based reinforcement learning. In Type-5 collaborations (targeted relays), we observe that the agents were able to identify unknown queries and relay those queries to appropriate neighbors such that the $\_successQuality$ improved. However, in Type-6 collaborations (generic relays), the agents needed the relay score in addition to the collaboration utility to obtain improved performance. This indicates that even when an agent had absolutely no idea which neighbor knew about a particular queried concept, it was still able to improve its performance by looking at two operational factors: the collaboration profile and the relay score, with the latter keeping track of the response of a neighbor to a relay request.

4.6. Analysis 5: Impact of multiagent inferencing

In this analysis, we investigate the impact of agents performing conceptual inferencing on query satisfaction. For this analysis, we use the following experimental setup. We distinguish three phases of activities: In Phase I, agents do not have the ability to perform conceptual inferencing and each agent has an empty translation table. Phase I shows the baseline system performance and the quality of service when the agents do not have inferencing ability. In Phase II, each agent has an empty initial translation table and is able to perform inferencing every 30 cycles and when a certain percentage of idle threads are available. Phase II shows how agents handle queries, collaborate, and distribute resources to perform inferencing. In Phase III, each agent has a full initial translation table but has no inferencing capability. Phase III shows the baseline system performance when all agents are given full translation tables.

Figure 16 shows the system performance in terms of the total number of links returned for the three phases, respectively, with different numbers of threads per agent. We see that conceptual inferencing improves the overall system performance significantly. Not recorded in the graph are occurrences of "panicky" collaborations: when an agent realizes that it has an initially empty translation table, it invokes conceptual inferencing repeatedly. Since this process is resource- and time-consuming, the agent does not have enough resources to satisfy queries, resulting in lowered system performance. Thus, we see that there is a delicate balance between how much conceptual inferencing is appropriate to ensure improved performance. Trying to learn too much or trying to help too often renders query satisfaction inefficient. Therefore, the strategy of an agent's decision and design of conceptual inferencing should be *gradual* and *selective*. This could also be a self-regulating rule for every agent in the system. For example, if a query for a concept has been well satisfied, the motivation to ask for a translation should be low even if the translation is empty or NIL.

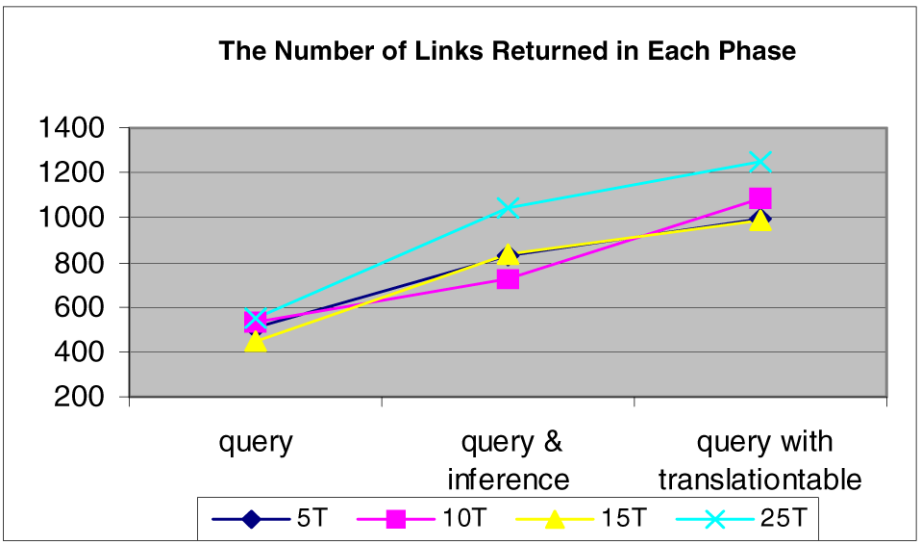

Fig. 16. The #links returned vs. phases. From left to right, Phase I: no inferencing capability with empty translation tables, Phase II: inferencing capability with initially empty translation tables, Phase III: no inferencing with full translation tables.

5. Related work

Our research is related to information matchmaking, cooperative learning, and resource description. When an agent selects and approaches a subset of its neighbors to ask for help, it is attempting to find a match in its neighbors. In our case, this happens whenever the incoming query demands a number of documents greater than what an agent has in its concept base. After selecting the appropriate neighbors, the agent assigns particular tasks or subtasks to each neighbor. Ideally, the agent matches the tasks to a neighbor's expertise. In our case, a neighbor's expertise corresponds to the credibility of a translation and the collaboration utility of that neighbor to the agent. Our agents learn about each other's concepts through collaboration in satisfying queries. Each agent performs such learning only when necessary—when it needs help from its neighbors. Thus, the learning is problem- or event-driven and only occurs when the agents collaborate. Our research work, through its DIR application, is related to resource description and resource selection. Resource description is the profiling of what a resource has—similar to what an agent profiles concerning its neighbors. Resource selection is the selection of resources per query—similar to an agent's decision making during the coalition formation and task allocation stages.

5.1. Information mediation and matchmaking

SIMS [1,2,13] is an information mediator that provides access to and integration of multiple sources of information. The mediator determines which data sources to use, how to obtain the desired information, how and where to temporarily store and manipulate data, and how to efficiently retrieve information. First, it selects the appropriate information sources, given an incoming query. This is done by translating a query expressed in terms of the concepts in the domain model into a query expressed in the information source models. In general, the choice is made so as to minimize the overall costs of the retrieval. For example, the cost can be minimized by using as few information sources as possible. Next, the mediator generates a query plan for retrieving and processing the data. The query plan specifies the precise operations that need to be performed, as well as the order in which they are to be performed. The uniqueness of the system lends itself to the semantic query optimization where rules are used to search for the least expensive query in the space of semantically equivalent queries. The goal here is to transform the original query into an inferred set of optimized subqueries—leading to fewer processes within the system. The idea of minimizing the overall costs of the retrieval by using as few information sources as possible is akin to the objective of our research. The mediator approach would allow a mediator agent to perform modeling on n other agents and share the modeling information with the agents such that only one (or a few) of the mediator agents does the modeling work. This would reduce the complexity of the multiagent system. However, this is assuming that how the mediator agent perceives an agent A is the same as how all other agents perceive A. Otherwise, the mediator agent would end up having to model agent A from multiple perspectives, one for each agent that interacts with the mediator agent. In that case, the mediator agent now would have to model roughly $n \times n$ relationships. In our framework, we assume that the modeling of an agent A will yield different results by different agents, due to the different query needs, operational constraints, and collaboration utility values. Thus, a mediator in our framework would have to deal with the $n \times n$ relationships. We have chosen to do away with the mediator approach to increase (1) flexibility: so that if a mediator agent becomes non-operational, the other agents can still operate; and (2) scalability: so that a mediator agent would not have to handle all agents. On the other hand, from a different viewpoint, we see that each agent in the CUDK framework behaves like a mediator agent, mediating between *itself* and its neighbors. That could be viewed as fundamentally similar to the SIMS approach.

Kuokka and Harada [14] described two matchmaking algorithms—SHADE and COINS—to support a cooperation partnership between information providers and consumers. Information providers take an active role in finding specific consumers by advertising their information capabilities to a matchmaker. Conversely, consumers send requests for desired information to the matchmaker. The matchmaker attempts to identify any advertisements that are relevant to the requests and notifies the providers and consumers accordingly. SHADE supports many modes of operation over formal, logic-based representations (recruiting, advertising, subscribing, brokering). COINS operates over free-text information, supporting fewer modes. Compared to our design, SHADE and COINS matchmake based on advertisements and requests, without taking the operational issues into account. For example, a producer that has advertised about its resources

at time t_1 may no longer have the resources available when the matchmaker approaches the producer at time t_2. This failure, which may be due to the dynamic characteristics of a resource, or to the communication bandwidth between the producer and the matchmaker, would not be captured by the matchmaker in SHADE and COINS. Essentially, our system considers matchmaking in terms of both conceptual and operational competitiveness.

Bayardo et al. [4] described a system called InfoSleuth where a broker accepts advertisements from new resources and notifications of resource unavailability at any time, leading to dynamic binding of resources. These brokers that serve the information sources interact with each other to accomplish query-answering goals. Compared to our design, InfoSleuth does not have the ability to predict when it decides whether to approach a particular agent for help—it assumes that if agent A_i does not hear from a particular agent A_j, then A_i will proceed with its assignment of sub-queries, for example, based on its current knowledge of resource binding. Thus, the responsibility for updating the binding actually lies with A_j. Cognitively, this requires A_j to be willing to update other agents about its current resources. In our approach, however, A_i keeps a conceptual profile as well as an operational profile. Given the two profiles, when A_i needs to decide its assignment of sub-queries, it is able to predict to a certain degree how useful other agents have been to its queries and assigns accordingly. As a result, this design rests the responsibility on A_i to keep track of its neighbors or other information resources. This has two advantages in a system with dynamic information resources. First, the updating of information resources is event-driven (triggered by a query) and consequently the number of messages due to advertisements and notifications is reduced. Second, cognitively, it is more sensible to have an agent shouldering the responsibility of keeping track other agents, since the agent is motivated to satisfy its queries.

5.2. Cooperative learning

Sen and Weiss [16] established that multiagent systems can bring out different types of learning. For example, agents may learn organizational roles, learn to benefit from market conditions, and learn to play better against an opponent. Coupled tightly with multiagent learning is communication. This relationship is mainly focused on the requirements on the agents' ability to effectively exchange useful information. According to Sen and Weiss [16], agents may learn to communicate, in which learning is viewed as a method for reducing the load of communication among individual agents. In this situation, the agent learns what to communicate, when to communicate, with whom to communicate, and how to communicate. Alternatively, agents may use communication as learning, where communication is viewed as a method for exchanging information that allows agents to continue or refine their learning activities. In our CUDK framework, the agents communicate to learn how to satisfy queries better and to learn about each other's concept bases. As a side effect, a CUDK agent, due to better profiling of its neighbors, also reduces the number of messages that it sends out to other neighbors. In our framework, we see that agents communicate to learn, leading to better communications, which in turn leads to better learning, and so on.

Distributed Ontology Gathering Group Integration Environment (DOGGIE) [25,26] deployed an ontology learning methodology that is similar to our work. The distributed ontology understanding among agents is carried out in three steps: locating similar semantic concepts, translating semantic concepts, and learning key missing attributes. To locate similar semantic concepts, an agent sends other agents the name of the concept and a sample of semantic objects of that concept. The receiving agent interprets the semantics by comparing the concept and objects and then sends back the result. In essence, DOGGIE agents are able to teach each other what their concepts mean using their own conceptualization. Our work uses the same principle that allows agents to exchange conceptual understanding by multiple 1-to-1 collaborations. However, our framework combines both operational and conceptual aspects. Not only does it allow the agents to initiate collaboration by considering the knowledge expertise of other agents, but it also equally emphasizes the operational issues using neighbor profiling. Each agent takes into account that in a dynamic multiagent system, an agent that is very capable may not have the resources (e.g., communication threads) to be helpful.

Further, in DOGGIE, there are several key assumptions [24]: (1) agents live in a closed world represented by the distributed collective memory, (2) the identity of the objects in this world are accessible to all the agents and can be known by the agents, (3) agents use a knowledge structure that can be learned using objects in the distributed collective memory, and (4) the agents do not have any errors in their perception of the world even though their perceptions may differ. Our assump-

tions are different. Our agents live in an open world. The collective memory expands and changes dynamically. The identity of the objects is not accessible to all the agents and may not be known by the agents if deemed not useful. Agents use a knowledge structure. The agents, though they do not have any errors in their perception of the world, may have incomplete modeling or profiling of their perception of the world due to lack of data and evidence, changing environments, and noise.

Wiesman and Roos [23] proposed a concept mapping measure based on the ontological knowledge or capacity of the agents. This measure indicates the odds that a query instance (utterance) from an agent A and an existing instance in an agent B denote the same entity in the world given the corresponding words of the two utterances. They identify a number of factors that influence the success of learning a mapping: (1) increasing the number of labels (keywords or descriptors) in an utterance makes the mapping problem easier, (2) increasing the number of words in the vocabulary set and the occurrence of sub- and super-concepts makes the mapping problem harder, (3) splitting and concatenating label values makes the mapping problem harder, and (4) labels in one ontology that do not occur in the other ontology make the mapping problem harder. In Section 6, we touch upon addressing the second and third factors. We see that these are important factors that will help improve our CUDK framework.

5.3. Resource description and selection

The research of information retrieval has progressed from the single database model to the multi-database model as the latter is often more suitable due to proprietary information, costs (e.g., access, storage, management, duplication, and transmission), and distribution of data [6]. In this paper, we report on our experiments and analyses of a *multiagent* DIR system. In the system, each agent, safeguarding its database and processing queries, learns from its experience through its interactions with other agents. The unique characteristic of our methodology is the agent treatment of resource description and selection.

There are three key stages of the multi-database model [6]: (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 description, 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.

The resource description problem arises as databases (or resources) with diverse specialties may not be known to the distributed query systems. Usually, each resource has a *guardian* to handle queries, publish the expertise of the resource, and interact with other resources. A guardian is very similar to an agent in our CUDK framework. To interact, a guardian must find out what other resources exist. When resources are dynamic, large, or myriad, finding out about other resources is non-trivial. If resources are dynamic, then a guardian has to ping these resources periodically, update its knowledge of these resources, and believe in its knowledge of these resources with reservation. If each resource is large (i.e., consists of a large number of documents), then a guardian has to decide how to cost-effectively provide the most representative documents for its list of expertise. Likewise, a guardian of another resource querying into this large resource has to believe that its knowledge of this large resource is incomplete or inaccurate. To simplify the description, a guardian may assume that what it knows of such a large resource is the best of what the large resource can offer. When the resources in the system are myriad, a guardian trying to complete its description of these resources may face diverse resources with overlapping expertise. A guardian will have to believe that what it knows may be sufficient but not optimal. That is, if agent a_i receives a query q for a concept c_k, and it knows of another agent (or resource), a_j, that has documents for c_k, then should a_i be satisfied with asking for help from only a_j, or should it explore the system to see whether there are other agents with more relevant documents for c_k? These are questions that research in resource description and selection investigates.

In general, resource descriptions can be created in a distributed fashion through a technique called query-based sampling [7]. In this strategy, each resource provider cooperates by publishing resource descriptions for its document databases. The sampling requires minimal cooperation and makes no assumptions about how each provider operates internally. In a way, 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 maintained dynamically on a per-

demand basis. 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. As a result, our resource description is subjective, instead of objective as in traditional DIR.

One of the key areas in the resource selection problem is ranking resources by how likely they are to satisfy the information need [6]. Conventionally, the desired database ranking is one in which databases are ordered by the number of relevant documents they contain for a query [10]. Techniques proposed include a Bayesian inference network [7], the Kullback–Leibler divergence [18], and a relevant document distribution estimation taking the database size into account [17]. In particular, Wu and Crestani [28] proposed a model that considers four aspects simultaneously when choosing a resource: a document's relevance to the given query, time, monetary cost, and similarity between resources. Though similar, 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. The first feature is an important change in strategy in resource selection as it also takes into account the "operational capabilities" of a resource.

6. Future work

As alluded to earlier in Section 1, most concept bases are too complex and cannot always be specified by a set of relevant documents. To allow for a hierarchy of concepts with relationships such as is-a and has-a links, the current designs of our concept base and the translation table have to be extended. First, each agent's concept base should be a concept hierarchy, with each node a concept with a set of relevant documents. Second, each entry in the translation table is a mapping between a node from an agent A's hierarchy to another node from another agent B's hierarchy. As a result, agent A also inherits what agent B inherits, and the confidence in such inheritance depends on how similar the two nodes are. With the hierarchical concepts, the conceptual inferencing is more complicated. To illustrate, say that there is a concept $C1$ that A knows, in a hierarchy such that $C1$ is related to n other concepts. Likewise, there is a similar concept $C2$ that B knows, related to m other concepts in B's hierarchy. The motivation for A to learn or discover the mapping between $C1$ and $C2$ could now also depend on the values of m and n. If n is large, then this mapping could allow A to find relevant documents for many of its known concepts in the hierarchy from B. If m is large, then this mapping could allow A to find more relevant documents for its known concepts from B. Further, with a hierarchical concept, that means the mapping between a concept $C1$ in A and a concept $C2$ in B could also be inferred as long as there is a node in A that maps into a node in B, and every concept that an agent knows is organized into one hierarchy. How should one decide which mappings to keep and which ones to infer? Factors that one could consider include the size of the hierarchies, the cost of storing the mappings, the cost of inferences, and the conflicts between direct mappings and inferred mappings in terms of credibility values. As discussed earlier in Section 5.3, factors and issues pointed out in [23] will also be considered.

Another key issue concerning our experiments and design is the scalability issue: how will the system behave when there are many agents (100's, 1000's), each responsible for an information resource? Will the agents behave similarly to what has been reported in this paper? To address the scalability issue, we have employed the notion of neighborhood in our design—each agent has a neighborhood where it can approach all the agents in the neighborhood for help, and the neighborhoods can overlap. With a neighborhood, the overall system may still be scalable since regardless of the size of the system, the size of a neighborhood could remain the same. Adopting this notion, we then expect to observe similar results in a larger system since the CUDK design does not have a bottleneck such as a centralized mediator. For example, if the agent is constrained with a fixed number of communication threads, then it will still perform the same tradeoffs in order to select the best neighbors to approach for help. Likewise, because of the resource constraints, even when the system is large, the size of an agent's neighborhood will still remain constrained by the resources. And with the relay capabilities, agents from different neighborhoods can still help each other out, thereby reducing the need for expanding an agent's neighborhood.

There is also a concern about duplicated query results collected from the neighbors of an agent. In general, duplicates are not desirable. However, one may make use of the duplicates by giving duplicates a higher rating since multiple neighbors *think* the same link matches the query. On the other hand, an agent could make use of the duplicates to measure the uniqueness of its neighbors. An agent should avoid approaching a pair of neighbors that tend to return the same links for the same query task. In addition to the translation credibility score and the collaboration utility, the novelty factor of each neighbor should play a role in multiagent collaboration. How two agents differ in their understanding of a concept could be of key importance and could motivate the mapping between the two agents to identify the differences between their concept bases.

7. Conclusions

We have implemented the first tier of a multiagent framework for collaborative understanding of distributed knowledge (CUDK) and evaluated the design in distributed information retrieval. Through our experiments, we have shown that CUDK-based agents work as a team to accept and process queries and to learn about (1) the relationships among their individual concept bases, and (2) the relationships among their individual operational capabilities and characteristics in such collaborative understanding. We have drawn several conclusions based on our experiments.

We have identified key factors that are important to consider when designing a multiagent system dealing with operational and knowledge constraints. First, an agent should ask only a few top-ranked neighbors for help, indicating that neighborhood profiling is important. Second, operational constraints impact the system more significantly than conceptual constraints. Based on our DIR application and experiments, we have also realized that the *motivation* for agents to learn each other's concepts is likely to be more resource-related than concept-related; that is, agents with poorer initial concept bases do not necessarily perform more poorly than agents with better initial concept bases if the agents collaborate. Third, though more resources improve the overall system performance, they could also lead to a less predictable system. Fourth, in multiagent tasks involving conceptual understanding, it is wise to perform conceptual inferencing instead of transferring jobs or tasks to those who know how to accomplish those tasks in order to have more consistent results. In systems where resources are so constrained that agents do not have viable options to solve a concept-related problem, accurate inferencing is also critical. Fifth, simple neighborhood profiles can effectively identify neighbors that are capable and helpful. This profiling mechanism facilitates strategic neighbor selection for conceptual inferencing. Sixth, agents are able to reduce the number of generic relays (spamming) by keeping track of the quality of relays to each particular neighbor. This further suggests that profiling can reduce the need for agents to learn concepts. Seventh, and most importantly, there is a delicate balance between how much conceptual inferencing is appropriate when operational factors are considered. The design of conceptual inferencing should be gradual and selective. It should be balanced with the tasks at hand, allowing the agents to learn about collaborating with others and subsequently identify the appropriate neighbors whose concepts they should learn to improve system performance.

Acknowledgments

The author would like to thank Jingfei Xu and Chao Chen for their programming work and experiments. The author would like to thank the anonymous reviewers and Dr. Donna Haverkamp whose comments significantly improved this paper. This project was partially supported by a University of Nebraska Layman Grant.

References

[1] Y. Arens, C.Y. Chee, C. Hsu and C. Knoblock, Retrieving and integrating data from multiple information sources, *Int. Intelligent & Cooperative Information Systems* **2**(2) (1993), 127–158.

[2] Y. Arens, C.-N. Hsu and C.A. Knoblock, Query processing in the SIMS information mediator, in: *Readings in Agents*, M.N. Huns and M.P. Singh, eds, Morgan Kaufmann, San Francisco, CA, 1998, pp. 82–90.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, ACM Press and Addison-Wesley, 1999.

[4] R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh and D. Woelk, InfoSleuth: agent-based semantic integration of information in open and dynamic environments, in: *Readings in Agents*, M. Huhns and M. Singh, eds, Morgan Kaufmann, San Francisco, 1998, pp. 205–216.

[5] A.H. Bond and L. Gasser, eds, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1988.

[6] J. Callan, Distributed information retrieval, in: *Advances in Information Retrieval*, W.B. Croft, ed., Chapter 5, Kluwer Academic Publishers, 2000, pp. 127–150.

[7] J. Callan and M. Connell, Query-based sampling of text databases, *ACM Transactions on Information Systems* (2001), 97–130.

[8] M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, Palo Alto, CA, 1987.

[9] M. Genesereth and R. Fikes, Knowledge Interchange Format Manual Version 3.0, Technical Report Logic-92-1, Stanford Logic Group, Stanford University, 1992.

[10] L. Gravano and H. García-Molina, *Generalizing GIOSS to vector-space databases and broker hierarchies*, in: Proceedings of the 21st International Conference on Very Large Databases (VLDB'95), 1995, pp. 78–89.

[11] T.R. Gruber, A translation approach to portable ontologies, *Knowledge Acquisition* **5**(2), 199–220.

[12] T.R. Gruber and G.R. Olsen, *An ontology for engineering mathematics*, in: Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94), Bonn, Germany, May 24–27. Morgan Kaufmann, 1994, pp. 258–269.

[13] C. Knoblock, Y. Arens and C. Hsu, *Cooperating agents for information retrieval*, in: Proceedings of the 2nd International Conference on Cooperative Information Systems, Univ. Toronto Press, Toronto, Ontario, Canada, 1994.

[14] D. Kuokka and L. Harada, Matchmaking for information agents, in: *Readings in Agents*, M.N. Huns and M.P. Singh, eds, Morgan Kaufmann, San Francisco, CA, 1998, pp. 91–97.

[15] G. Mineau, Sharing knowledge: starting with the integration of vocabularies, in: *Conceptual Structure: Theory and Implementation*, H. Pfeiffer and T. Nagle, eds, Proceedings of the Seventh Annual Workshop, Las Cruces, NM, July 8–10, Springer-Verlag, 1992, pp. 34–45.

[16] S. Sen and G. Weiss, Learning in multiagent systems, in: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, ed., MIT Press, 2000.

[17] L. Si and J. Callan, *Relevant document distribution estimation method for resource selection*, in: Proceedings of the 25th Annual Int. ACM SIGIR Conference on Research and Development in Information Retrieval, 2003.

[18] L. Si, R. Jin, J. Callan and P. Ogilvie, *A language model framework for resource selection and results merging*, in: Proceedings of the 11th CIKM, 2002.

[19] L.-K. Soh, *Multiagent distributed ontology learning*, in: Working Notes of AAMAS2002 Workshop on Ontologies in Agent System (OAS), Bologna, Italy, July 15–19, 2002.

[20] L.-K. Soh, *Collaborative understanding of distributed ontologies in a multiagent framework: design and experiments*, in: Proceedings of AAMAS 2003 Workshop on Ontology in Agent Systems (OAS), Melbourne, Australia, 2003, pp. 47–54.

[21] L.-K. Soh and C. Chen, *Balancing ontological and operational factors in refining multiagent neighborhoods*, in: Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'2005), July 25–29, Utrecht, the Netherlands, pp. 745–752.

[22] L.-K. Soh and C. Tsatsoulis, *Reflective negotiating agents for real-time multisensor target tracking*, in: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, WA, Aug 6–11, 2001, pp. 1121–1127.

[23] F. Wiesman and N. Roos, *Domain independent learning of ontology mappings*, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'2004), New York, NY, July 19–23, 2004, pp. 846–853.

[24] A.B. Williams, Learning to share meaning in a multi-agent system, in: *Autonomous Agents and Multiagent Systems*, vol. 8, 2004, pp. 165–193.

[25] A.B. Williams and Z. Ren, *Agents teaching agents to share meaning*, in: Proceedings of ICMAS'2001, ACM Press, Montreal, Canada, 2001, pp. 465–472.

[26] A. Williams, A. Padmanabhan and M.B. Blake, *Local consensus ontologies for B2B-oriented service composition*, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'2003), Melbourne, Australia, July 14–18, 2003, pp. 647–654.

[27] M. Wooldridge, *Reasoning about Rational Agents*, The MIT Press, Cambridge, MA, 2000.

[28] S. Wu and F. Crestani, *Multi-objective resource selection in distributed information retrieval*, in: Proceedings of IPMU'02, Annecy, France, July, 2002.