

# Agent-Based Peer-to-Peer Service Networks: A Study of Effectiveness and Structure Evolution\*

Yathiraj B. Udipi  
Dept of Computer Science  
North Carolina State U  
Raleigh, NC 27695, USA  
ybudupi@ncsu.edu

Pinar Yolum  
Dept of Artificial Intelligence  
Vrije U, Amsterdam  
The Netherlands  
pyolum@few.vu.nl

Munindar P. Singh  
Dept of Computer Science  
North Carolina State U  
Raleigh, NC 27695, USA  
singh@ncsu.edu

## Abstract

*We study peer-to-peer service networks consisting of autonomous agents who seek and provide services. To fulfill its local needs, an agent attempts to discover and select information services by contacting others. Other agents can potentially help it by giving referrals to guide its search. Each agent autonomously decides whom to contact for a service and whom to provide a service or a referral. Service networks evolve as the agents change their neighbors to improve how their needs are fulfilled. If an agent autonomously decides to, it may cache some responses from other (information) services. Thus our service networks differ significantly from traditional peer-to-peer networks.*

*We study the behavior and evolution of agent-based service networks in the presence of caching. We observe that even a small cache improves agents' success in discovering needed services and enables a few initial service providers to serve the information needs of many. Caching induces clustering of agents based on interest. Caching prematurely, however, can cause the propagation of poor quality information. The above study supports design rules to help decide when and in what situations to apply caching to maximize success while discovering services.*

## 1. Introduction

The purpose of information access is to fulfill a need. In conventional practical systems, information access is mapped to database lookup and an emphasis is placed on looking for correct (as from relations) or relevant (as from text) results. By contrast, in open settings, correctness and

relevance may not be as clearly independent of users, so we must look for authoritative sources that can provide correct and relevant results. The results may be available from multiple sources and hence are not necessarily unique. The above distinction corresponds to the difference between requesting a copy of Abraham Lincoln's Gettysburg Address (of which, say, there is one definitive version) and a commentary on the American Civil War. Or, more prosaically, a copy of the first draft WSDL standard versus an analysis of Web Services. Importantly, in an open environment, we cannot rely on traditional mechanisms using regulatory restrictions for ensuring the quality of the services obtained or the trustworthiness of the peers found over the network.

Thus, the need for finding authoritative sources leads us to develop a decentralized approach wherein peers can find peers who are trustworthy and offer high quality services. These peers could control and exploit private knowledge bases which were unreachable by traditional search engines.

Some of the peers would be service providers and they can offer different types of services, different ways of carrying out services, and different levels of quality. Other peers who are service consumers would look for these providers and use them. Service consumers differ in their service needs and in the ways in which they evaluate service providers and the services provided. When a peer is requested for a service, it may offer the service or it may provide a *referral* by recommending other peers.

Some information services can be cached so that they can be reused. Caching aids the search for information since a peer that is looking for information can find it in some cache—its own cache or the cache of another peer. Traditional distributed system approaches to caching view information access primarily as looking up specific objects and hence are not sufficient to fulfill the information needs flexibly. We develop a flexible caching scheme for service networks and study it in depth. Our main results are that even with small caches and fewer number of service providers, the agents can find answers successfully. The success rate

---

\* The first author is a full-time doctoral student. This research was supported by the National Science Foundation under grant ITR-0081742. This is an revised version of a paper appearing in the AAMAS 2003 Workshop on Agent Oriented Information Systems.

partially depends on the agents’ evaluations of answers and their cache size. We conclude with design rules that can help apply caching appropriately.

**Organization.** Section 2 introduces key elements of our referrals-based model and describes the search with the help of an example system. Section 3 describes the evaluation scheme, discusses some of the important control variables, and gives our results. Section 4 discusses some related work and concludes with our design rules.

## 2. Trustworthy Service Networks

We provide some background on our model of trustworthy service networks. The system consists of *principals* that are computationally represented via *agents*. We model two types of agents: service consumers who are looking for services and service providers who are offering services themselves.

**Basic Communication Protocol** When an agent is in need of a service, it begins to look for a trustworthy provider for the specified service. The agent queries some other agents from among its *neighbors*, which are typically a small subset of the agent’s acquaintances. A queried agent may perform the specified service or may give referrals to other agents. The querying agent may accept a service offer or may follow referrals that it receives. Hence an agent may receive the required service from its neighbors or through a series of referrals.

**Trust** We understand trust as a property defined for an agent with respect to a particular service. For example, we may trust a hotel agent for all our accommodation needs but not for our travel needs. Because of different evidence or different evaluations of the same evidence, two agents can have different assessments of trust for a particular agent. When an agent is evaluating its trust in another, it considers the information it received directly (useful answers and referrals) and indirectly (referrals from others). Agents thus keep track of the trustworthiness of other agents by querying services or by flexibly taking referrals.

**Representation** Each agent maintains models of its acquaintances, which describe their *expertise* (i.e., the quality of the services they provide), and *sociability* (i.e., the quality of the referrals they provide). An agent may have varying levels of interest in multiple domains. Similarly, an agent can have different grades of expertise in multiple domains. This motivates us to represent the interests and expertise of the agents as term vectors from the vector space model (VSM) [8], each term corresponding to a different domain. Each of the dimensions of the vector can have values ranging from zero to one. The higher the value, the closer is the match to that particular domain. Sociability is captured as a scalar. A query is also modeled as a vector. An agent can

specify the keywords of search which can be transformed to a vector by assigning different values for the different dimensions of the vector.

**Strategies for Querying and Adaptation** The agent sends the query to some of its neighbors that are likely to provide an answer. These neighbors are chosen through the capability metric, which measures how sufficient the expertise vector is for a given query vector [9]. This metric is based on cosine similarity but it also takes into account the magnitude of the expertise vector. This means that expertise vectors with greater magnitude indicate higher capability for the given query vector. In Equation 1,  $Q (\langle q_1 \dots q_n \rangle)$  is a query vector,  $E (\langle e_1 \dots e_n \rangle)$  is an expertise vector, each with  $n$  dimensions.

$$Q \otimes E = \frac{\sum_{t=1}^n (q_t e_t)}{\sqrt{n \sum_{t=1}^n q_t^2}} \quad (1)$$

If  $Q \otimes E$  is greater than a threshold, for two agents  $A$  and  $B$ , then  $B$  is capable of answering  $A$ ’s query. Keeping the threshold high results in choosing only the most capable agents. When caching, if the querying agent receives multiple good answers, then only the best answer is cached.

The expertise and sociability of agents as modeled by an agent are adapted based on the service ratings given by its principal. When an answer comes in, the modeled expertise of the answering agent and the sociability of the agents that helped locate the answerer (through referrals) are increased or decreased based on the quality of the received answer. At certain intervals during the simulation, each agent can modify its choice of neighbors based on these acquaintance models.

**Service Caching** With the basic communication protocol, the agents need not always get an answer to their query, but whenever they get an answer it is generated from scratch by some provider. Not all agents are service providers and hence all the agents depend on the few service providers present. This causes a potential bottleneck for three reasons. One, because of the incomplete connectivity of the agents in the network, many times not all consumers can reach the service providers [10]. Two, if the providers are reachable by all, then the providers can become overloaded with queries. Three, if the agents do not learn the services offered and if their principal requests that service again, they will have to repeat the process of service location.

These shortcomings motivate us to find a way by which we can reuse some of the services already offered by some providers. Information services can be easily cached by the agent, so that the providers need not generate them again. The caching of a service by an agent takes a dual role: first to satisfy its principal’s needs and second to serve the other agents who are in need of that service. Caching thus aids information retrieval in service networks and makes them more efficient.

Agents can maintain individual caches to store the services. A cache consists of a set of entries that contain the answer as well as some information about the quality or appropriateness of the given answer. A cache entry is hence a  $\langle query, answer, quality \rangle$  triple. The quality of the cache entry is modeled based on how appropriate it is to the owner of the cache based on its interest.

When the agents cache information, they store answers in their individual caches based on their own interests. The sizes of the caches are usually limited. Hence, a cache entry must be replaced with a new entry based on the cache replacement policy. An agent can also delete an entry due to a lack of interest in the item. Even after the provider of a cached item deletes the original item and stops providing that item, the peers that have a cached copy of the item are free to keep and serve the item from their caches.

**An Example Search** An agent generating a query follows Algorithm 1, and an agent that receives a query acts in accordance with Algorithm 2.

---

#### Algorithm 1 Ask-Query()

---

```

1: if (cachingImplemented and hasCachedAnswer) then
2:   Evaluate cached answer
3:   Determine if it has to stay in cache and then return
4: else
5:   while (more matching neighbors to ask) do
6:     Send query to matching agents
7:     Receive message
8:     if (message.type == referral) then
9:       Send query to referred agent
10:    else
11:      Add answer to answerset
12:    end if
13:  end while
14:  for  $i = 1$  to  $|answerset|$  do
15:    Evaluate answer( $i$ )
16:    Cache answers
17:    Update agent models
18:  end for
19: end if

```

---

Let us describe the protocol using the example network of Figure 1. Here,  $A_1, A_2, A_3, A_4, A_5,$  and  $A_6$  are agents. An edge from agent  $A_i$  to agent  $A_j$  means that  $A_j$  is a neighbor of  $A_i$ . For each agent, the vectors  $I$  and  $E$  represent interest and expertise respectively. Agent  $A_1$  is interested in an item that is cast into a query vector of  $[0.2, 0.8, 0.3]$ . Agents  $A_4$  and  $A_6$  have the desired item.  $A_1$  sends the query to both  $A_2$  and  $A_3$ .

*Case without Caching* Let us first discuss the case when the agents do not have caches. Now when  $A_2$  gets the query from  $A_1$ , it does not have the necessary expertise to provide an answer. Notice that among  $A_2$ 's neighbors ( $A_4, A_5$ ),  $A_4$

---

#### Algorithm 2 Answer-Query()

---

```

1: if (cachingImplemented and hasCachedAnswer) then
2:   Return cached answer
3: else if (hasEnoughExpertise) then
4:   Generate answer
5: else if (cachingImplemented and interestedInQuery) then
6:   Ask-Query()
7:   Return answer
8: else
9:   Refer neighbors
10: end if

```

---

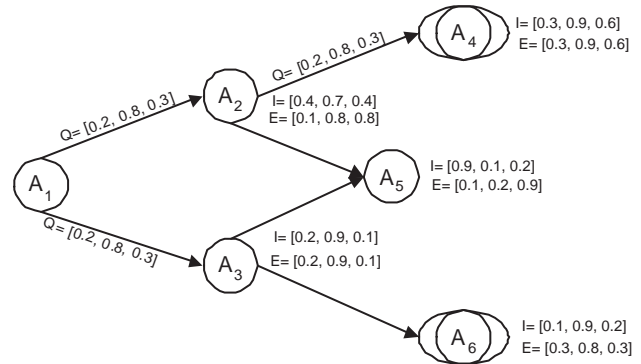


Figure 1. An example search

---

is the only one whose modeled expertise vector matches the query, so  $A_2$  gives a referral ( $A_4$ ) to  $A_1$ .  $A_3$  also gives a referral ( $A_6$  and  $A_5$ ) back to  $A_1$ . When  $A_1$  receives the referrals to  $A_4, A_6,$  and  $A_5$ , it sends its query to them. After  $A_1$  receives answers from  $A_4$  and  $A_6$ , it evaluates the answers by computing how much the answers match the query. The best answer is chosen from the two. In this example,  $A_1$  does not receive any answer from  $A_5$ .

*Case with Caching* Now consider the case when the agents may cache information. Before sending out the query to its matching neighbors,  $A_1$  checks if it has the answer to the query in its cache. When it does not find an answer in its cache, it sends, as in the above case, the query to  $A_2$  and  $A_3$ .  $A_2$  receives the query and checks its cache to see if any matching answer exists there. Not finding the answer in its cache,  $A_2$  decides to send the query to its neighbors using the Ask-Query() method of Algorithm 1, since its interest is similar to the query. That is,  $A_2$  is itself curious about the answer. After  $A_2$  obtains an answer itself, it can forward it to  $A_1$ .  $A_2$  sends the query only to  $A_4$  because  $A_4$ 's expertise as modeled by  $A_2$  matches the query. After  $A_2$  receives the item from  $A_4$ , it evaluates the answer and if useful, puts the item into its cache and forwards it to  $A_1$ . Meanwhile, since  $A_3$  is not interested in the query, it just sends a referral ( $A_5$  and  $A_6$ ) to  $A_1$ .  $A_1$  sends out queries to agents

for which it received referrals and the procedure is repeated. After evaluation,  $A_1$  caches only the useful (good) answers.

In the case when agents do not cache information,  $A_1$  increases the modeled expertise of  $A_4$  and  $A_6$  for having given good answers and also increases the sociability of  $A_2$  and  $A_3$  for having given good referrals. But, in the case when agents cache information,  $A_1$  increases the modeled expertise of  $A_2$ , as  $A_2$  provides the answer (returned from  $A_4$ ) and increases the sociability of  $A_3$  for having given a good referral ( $A_6$ ). In both the cases,  $A_1$  decreases the sociability of  $A_3$  for having given a wrong referral of  $A_5$  and the expertise of  $A_5$  is also decreased for not having given any answer. During a neighbor change,  $A_1$  may decide to drop  $A_3$  in favor of  $A_4$  or  $A_6$ .

### 3. Evaluation

Enabling such a flexible caching scheme poses interesting research questions.

- How does the success rate of the service network improve with caching, and how does it correlate with the cache size of the agents?
- How does the initial number of providers affect the success rate of the system with caching?
- Does the caching influence who the agents become neighbors with?

We study these questions with simulations over an agent platform, which enables the agents to query each other, obtain responses or referrals, and cache useful answers.

#### 3.1. Experimental Setup

Our simulation contains 400 agents and 4 domains in the vector model. There are 20 or 40 *experts* (5 or 10 in each domain) who provide services—these are the agents who originate the information that others may cache. Experts give good answers in their domains of expertise. The remaining agents are service consumers, whose interests may span several domains. Service providers do not generate queries and therefore do not have any neighbors. All the service consumers have four randomly chosen neighbors. During the course of the simulation, after every three queries each agent has a chance of modifying its choice of neighbors, but the number of neighbors for each agent remains the same.

Each agent ranks its acquaintances based on an equal weight of both sociability and expertise. The top-ranked acquaintances become the neighbors. As the agents change neighbors, they find neighbors that better suit their interests. The simulations are run for a duration corresponding to 10 neighbor changes.

The queries and the answers are randomly generated in the simulations as there are no actual principals (i.e., humans) and no actual user feedback involved. More precisely, an agent generates a query by slightly perturbing its interest vector, which means that the agent asks a question similar to its interests. Similarly, the answers are generated by slightly perturbing the expertise vector. Thus, implicitly, the agents with high expertise end up giving the correct answers.

#### 3.2. Control Variables

We describe here some of the variable parameters used in our study and discuss how they affect the functioning of the caching system.

**Threshold for forwarding query:** When an agent receives a query that it is also interested in, it can search for the answer itself. We compute the necessary level of interest by using the capability metric (defined in Section 2) between the query and the interest vector (Algorithm 2, line 5). If this similarity is greater than the threshold for forwarding query, the agent performs the search itself. Setting high values for this threshold results in a situation when the agent will search only for queries that are highly similar to its own interests, whereas setting low values results in a situation where the cache would have answers to a broad area of interests. A low value for the threshold reduces its usefulness to the agent’s own principal. Therefore, we set the threshold high so that most of the forwarded queries will closely match the agent’s own interest.

**Cache size:** This is the number of entries that can be stored by an agent in its cache. Having unlimited or a high cache size results in every good answer being cached. This results in a majority of answers being directly retrieved from the agent’s own cache, leading to a reduced exploration of the service network. Limiting the cache size means that we will encounter cache overflows to be addressed according to a cache replacement policy. We initially experimented with a *random replacement policy*, which randomly replaces any entry in the cache whenever there is an overflow. However, we improved the performance using a *quality based replacement policy*, which replaces the entry with the least quality if the quality of the new answer is higher than the least one. We consider caches of size 1, 15, 20, or 30.

**Selectivity:** This threshold is used when an agent evaluates an answer (Algorithm 1, line 15) using the capability metric discussed earlier. The answers with capability higher than this threshold for the given query vector qualify to be good answers. High selectivity means that only high quality answers qualify to be cached. The range of this threshold is 0.0 to 1.0 and we have used the values of 0.35 and 0.45.

**Fidelity:** This threshold is used to search the cache for a

matching query. If the similarity between the current query and a previous query exceeds this threshold, then it is considered a cache hit. If there are multiple hits, the best cached answer is returned. A fidelity of 1.0 means that an exact match between the queries is required, which would be rare, but a high fidelity improves the quality of the answers received. Hence the fidelity is set in the range 0.9 to 1.0.

### 3.3. Results

We evaluate this caching technique by comparing the performance of the service network with and without caching and by varying some of the control parameters discussed above. The evaluation depends on measures of the performance and structure of the network.

**3.3.1. Effect of Caching on Success Rate** This measures the ratio of the queries for which at least one good answer is found. Let  $T$  be the total number of queries. Let  $G$  be the number of queries for which at least one good answer is found. Then success rate is given by  $G/T$ . The success rate is calculated for each agent for one neighbor change of the simulation and then averaged over all service consumers. Since the service providers do not generate any queries, they are not factored into the calculation. An agent can find a good answer for its query, either from other agents or from its own cache. Initially, more answers are found through other agents. As agents fill their caches, more answers are retrieved from the caches, without the agents having to perform another search. This enables the agents to reach answers faster than before.

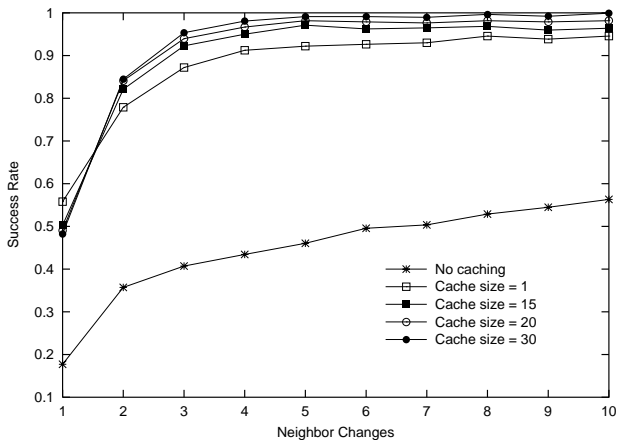


Figure 2. The increase in success rate

Since it is not practical to have an unlimited cache size, we limit the cache size. Figure 2 plots the success rate for

different levels of caching when the population has 20 experts and a moderate level of selectivity of 0.35. Even with a small cache size (size 1), the number of good answers received increases tremendously compared to no caching, as seen by the jump of success rate in Figure 2. The increase is even greater when there is a bound on how far the referrals are followed.

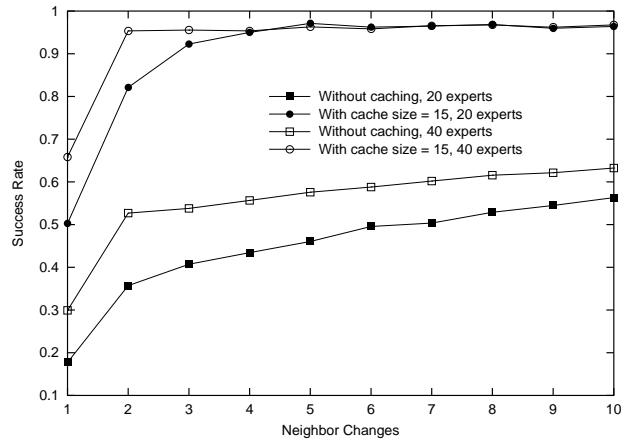


Figure 3. The effect of number of experts

There is a big improvement in the success rate when we change from no caching to a cache size of 1. The success rate saturates early with further increase in cache size. For example, it is slightly improved for a cache size of 15 and it soon peaks for a cache size of 30, with no further improvement. This saturation can be explained by the fact that after a certain size, the number of good answers retrieved from the cache tends to be the same.

**3.3.2. Effect of Experts on Success Rate** Figure 3 plots the success rates for the case without caching and with a caching size of 15, both with 20 and 40 experts, respectively. With caching, the lines for 20 and 40 experts overlap suggesting that the number of experts is less important when caching is used. On the other hand, without caching, the success rate is higher for 40 than for 20 experts. Intuitively, caching reduces the dependence on finding an expert since after a while many service consumers have good answers in their caches. That is, any agent who returns an answer from its cache takes on the role of a quasi-expert, even though it may not truly be an expert.

**3.3.3. Effect of Selectivity on Success Rate** Selectivity decides the effect of caching on success rate. With higher selectivity the agents will be searching for higher quality answers. Since these answers are harder to find, there is more incentive in caching. In Figure 4, the lines with white boxes and white circles denote the success rate for cache size 1

and the black boxes and circles denote cache size 20 for the two values of selectivity. When the selectivity is set to 0.45, we observe that there is a big difference between the success rates for cache size 1 and cache size 20. The agents that cache more answers are at a clear advantage. However, the difference between cache size 1 and 20 is not this big when the selectivity is set to 0.35. Since the agents have lower standards for evaluation, they can find answers that match their queries more easily. Hence, agents do not benefit from larger caches as much.

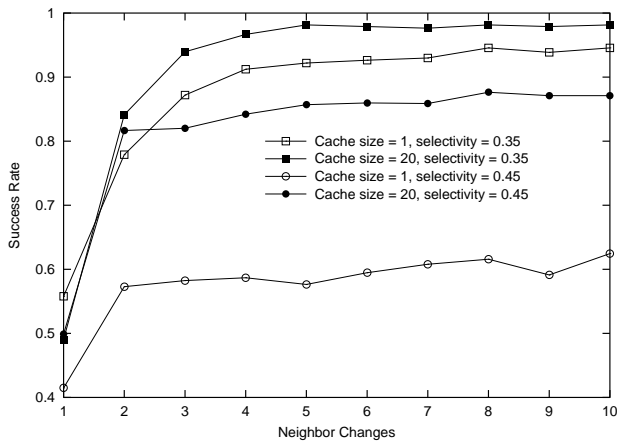


Figure 4. The effect of selectivity

**3.3.4. Effect of Caching Population Size on Success Rate** Figure 5 compares the success rate for three populations that vary in the percentage of agents that cache. The first result is that even when one fourth of the agents cache, a high success rate is achieved. This is compatible with our previous result on cache sizes. The second result is that initially when fewer agents cache, the success rate is higher. As the simulation proceeds, populations with higher number of cachers achieve a better success rate. In Figure 5 after neighbor change 1, the population where one fourth of the agents cache is the most successful. With the second neighbor change, the most successful population is the one with one half of the agents caching. Finally, after neighbor change 5, the population where all the agents are caching achieve the highest success rate. Intuitively, agents do not find the best possible answers at first. When there is more caching, these lower quality answers are circulated for a longer time in the system before being replaced by better answers. However, if there is less caching, these answers are sooner replaced with higher quality answers. As these better answers are found, then the population with more caches achieve a higher success rate.

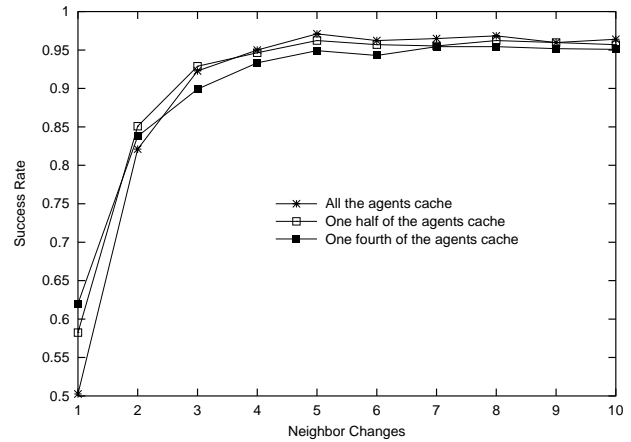


Figure 5. The effect of caching population size

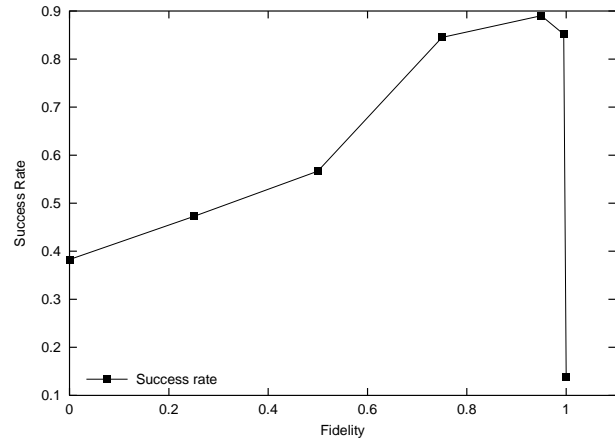


Figure 6. The effect of fidelity

**3.3.5. Effect of Fidelity on Success Rate** The fidelity threshold is important in determining the accuracy of the results returned from the cache. Figure 6 plots the success rate for different values of fidelity with 20 experts and a cache size of 30. We observe that the success rate keeps deteriorating as we lower the value of this threshold. On the other hand, we cannot have a perfect value of 1.0 for this threshold as the queries will have to be matched perfectly to be retrieved from cache and this happens rarely, if at all. There is a drastic drop in the success rate for 1.0 as it reduces to the case of no caching.

**3.3.6. Coverage and Correctness** The following two measures guide the behavior of success rate when fidelity is varied. Let  $G$  and  $T$  be as in the definition of success rate. Let  $A$  be the number of queries for which an answer is found. *Coverage* is the ratio of the num-

ber of queries answered to the number of queries posed, i.e.,  $A/T$ . *Correctness* is the ratio of the number of queries with a good answer (i.e., answered correctly) to the number of queries answered, i.e.,  $G/A$ . The product of these two measures is the success rate.

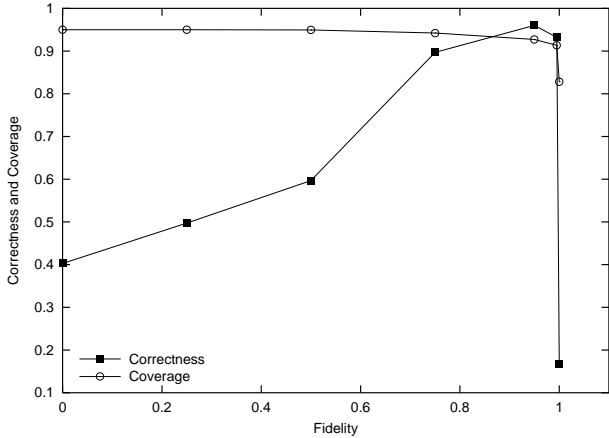


Figure 7. Coverage and correctness

Figure 7 plots coverage and correctness against fidelity. We observe that coverage does not vary much as we increase fidelity, but there is a sudden decrease when fidelity approaches 1.0. Correctness increases steadily with fidelity, but this also drops suddenly for the fidelity value of 1.0. This can be explained as follows. When the fidelity is 1.0, the system is essentially reduced to the case of no caching, hence few queries are answered. Comparing Figures 6 and 7, we observe that the success rate increases gradually with the increase in correctness. It reaches a maximum when the correctness reaches a maximum, while the coverage does not vary much. When fidelity approaches 1.0, as both correctness and coverage drop, the success rate too drops.

**3.3.7. Interest Similarity** Interest similarity measures how similar the interests of two agents are (based on their interest vectors).

$$I_A \oplus I_B = \frac{e^{-\|I_A - I_B\|^2} - e^{-n}}{1 - e^{-n}} \quad (2)$$

In Equation 2,  $I_A$  and  $I_B$  denote the interest vectors of two agents  $A$  and  $B$ , respectively. Here  $n$  is the length of the interest vectors. The metric captures the Euclidean distance between the two vectors and normalizes it.

For each agent, its similarity to its neighbors is calculated. Then, the average over all the agents is taken. Figure 8 plots the variation of interest similarity for different levels of caching with 20 experts. (Our results for 40 experts

are similar.) We observe that interest similarity increases with every neighbor change. This means that the agents tend to form neighbors with those agents who have similar interests. The intuitive explanation of this is that when two agents  $A$  and  $B$  have similar interests, they tend to generate similar queries. If  $A$  has cached an answer to a query that is later also generated by  $B$ ,  $B$  can locate  $A$  as a good provider. Hence,  $A$ 's cache provides an incentive for  $B$  to choose  $A$  as a neighbor.

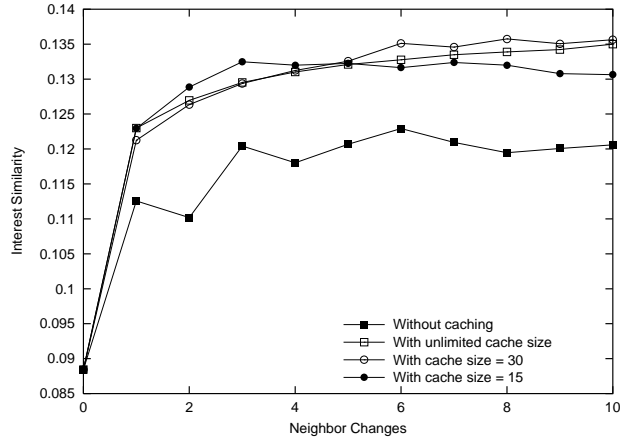


Figure 8. The change in interest similarity

We see that the values of interest similarity in the cases with caching are higher than the case without caching. We conclude that caching results in the agents with similar interests to group together. This situation could lead to some interesting observations such as the formation of communities in the network of agents.

## 4. Discussion

We review the literature with respect to our approach.

### 4.1. Literature

Referrals provide a powerful way in locating trustworthy services. Multiple Intelligent Node Document Servers (MINDS), based on the documents used by each user, was an early agent-based referral system [2]. Each node is allocated a set of documents and the nodes help each other find documents in the network. Kautz *et al.* model social networks statically as graphs [6] and study their properties. Semantic overlay networks (SONs) are systems where agents with semantically related content are clustered together [3]. A query generated by an agent is sent to another agent within its SON if the query corresponds to the area of that SON, else the query is forwarded on to a node of a

suitable SON. This is similar to the querying approach of our service network, where the query is forwarded to those whose interests match the query. The above approaches do not incorporate caching.

Recently, several peer-to-peer file storage architectures have been proposed, e.g., [4]. Rather than allowing peers to autonomously decide what to cache or index, these systems model the network as a distributed hash table that maps keys to peers. However, peers do not choose which items they want to keep. In our approach, agents cache items that are of interest to them. This allows the agent to reach the items of interest faster.

PeerOLAP is an adaptive P2P system for caching online analytical processing queries [5]. Each peer autonomously decides based on its policies on which items to cache. Some policies take into account only the peer who is caching while other policies consider what the neighbors may be interested in and caches those items as well. Peers select their neighbors based on how well they have answered previous questions. There is an associated cost to search each node. The search tries to minimize the total cost of accessing an item. Our neighbor selection policies are similar to this and we can easily accommodate a policy based on previous answers.

Aberer and Despotovic develop a reputation-based trust model for peer-to-peer systems [1]. The complaints from agents who are cheated are stored in a distributed structure maintained by multiple parties. An agent who is looking for a trustworthy agent pulls the complaints from this structure and aggregates them. In our approach, the agents that provide evidence are rated. Hence, agents adapt by choosing neighbors that are most useful to them. Contrary to aggregating evidence, the agents that have not been useful in previous interactions are not considered in future interactions.

In Richardson et al.'s approach, each user maintains trusts in a small number of other users [7]. These trusts are then composed into trust values for all other users. They do not form an agglomerate "trustworthiness" of each user. Each receives her own personalized set of trusts, which may be vastly different from person to person. Our referral approach can easily incorporate this interpretation of trust.

## 4.2. Conclusions: Design Rules

The above motivated the case for caching information services in service networks, showed how caching could be integrated with a referrals-based search process for trustworthy providers, and studied various important properties of the effectiveness and evolution of service networks. Now based on our studies, we synthesize some tentative *design rules* for the design and configuration of service networks. This set of rules is far from complete but it indicates the kinds of empirical, heuristic guidelines that our research can

produce.

**Cache Judiciously** Be reluctant about introducing a poor answer into the cache and keeping it there. If the agents are not selective, then they should be eager to find better answers than what they already have. That is, they can only have a small cache so that they will replace often. This means that the agent must continually seek improved answers.

**Speak Cautiously** Only circulate high quality answers—be more selective about responding than caching. That is, keep the fidelity high, especially when not receiving high quality answers.

**Be Selective** Be picky in the beginning; relax later. If the agents are initially selective, they can cache good answers early and be neighbors with good experts.

The trade-offs among the rules might depend on criteria such as the urgency of the information need and the staleness of the information with respect to the rate at which the agent's interests change, which we defer to future work. Another future direction is to allow the fidelity to vary based on the quality of the answers in the cache. In other future work, we plan to study the problem of revoking answers or letting cached answers expire. If the essential updates or revocations can propagate through the network, it would produce better responses without compromising the overall quality.

## References

- [1] K. Aberer and Z. Despotovic. Managing trust in a peer-to-peer information system. *Proc. 10th Intl Conf Info & Know Mgt (CIKM)*, pp. 310–317, 2001.
- [2] R. Bonnell, M. Huhns, L. Stephens, and U. Mukhopadhyay. MINDS: Multiple intelligent node document servers. *Proc. 1st IEEE Intl Conf Office Automation*, pp. 125–136, 1984.
- [3] A. Crespo and H. Garcia-Molina. Semantic overlay networks for P2P systems. Computer Science TR, Stanford U, 2002.
- [4] F. Dabrek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *Proc. ACM Symp Operating System Principles*, pp. 202–215, 2001.
- [5] P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K. L. Tan. An adaptive peer-to-peer network for distributed caching of OLAP results. *Proc. ACM SIGMOD*, pp. 25–36, 2002.
- [6] H. Kautz, B. Selman, and M. Shah. ReferralWeb: Combining social networks and collaborative filtering. *Comm ACM*, 40(3):63–65, Mar. 1997.
- [7] M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic Web. *Proc. 2nd Intl Semantic Web Conf (ISWC)*, pp. 351–368, 2003.
- [8] G. Salton and M. J. McGill. *An Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.
- [9] M. P. Singh, B. Yu, and M. Venkatraman. Community-based service location. *Comm. ACM*, 44(4):49–54, Apr. 2001.
- [10] P. Yolum and M. P. Singh. Emergent properties of referral systems. *Proc. AAMAS*, pp. 592–597. 2003.