Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 68 (2015) 217 – 226

Procedia Computer Science

HOLACONF - Cloud Forward: From Distributed to Complete Computing

Applying short-term memory to social search agents

Albert Trias i Mansilla[a]*, Sam Sethserey[b], Josep Lluís de la Rosa i Esteva[a]

[a]*Agents Research Lab, TECNIO Centre EASY, University of Girona, Campus de Montilivi, E17071 Girona, Catalonia (EU)*
[b]*Computer Science Department, Institute of Technology of Cambodia, BP 86, Bvd. Pochentong, Phnom Penh, Cambodia*

Abstract

This paper presents about our research in social search. Generally, the research in social search falls into two principal challenges. The first challenge is how to find more relevant answers to the question. The second one is how to increase speed in finding relevant answers.
Recently, we had provided two algorithms called Asknext and Question Waves to find more relevant answers compared to the baseline algorithm BFS. But, the search speed of the two proposed algorithms still the subject to improve.
In this paper, we introduce the agents' ability of learning the answers from the interactions with other agents so that they can quickly answer the question of other agents. We model this learning process by implementing the concept of data caching as the short-term memory of each social search agent. The result improvement of the speediness and the reduction of the number of messages used to communicate between agents, after apply agent's short-term memory concept, demonstrates the usefulness of the proposed approach.

Keywords: Social search; query routing, BFS; Asknext; Question Waves; LRU Cache;

1. Introduction

The advent of online social networks and the advances on artificial intelligence encourage the automation of social search within the village paradigm[1] in the information search process. The village paradigm consists on asking someone that may satisfy information needs; while the alternative paradigm, the library[1], consists in accessing a document that may contain the answer through a catalogue.

* Corresponding author. Tel.: +34-972418478; fax: +0-000-000-0000 .
E-mail address: albert.trias@udg.edu

doi:10.1016/j.procs.2015.09.237

The library paradigm has been largely automated, in the case of the web, there are several search engines that automate their features in large scale. The village paradigm is also present in the web in the form of Q&A portals and forums, but this paradigm, even being the oldest, has been less automated than the library paradigm. One key aspect of the village paradigm is query routing, which consists in finding a candidate that may satisfy the information need.

Although search engines provide fast results, they usually provide a large list of documents in which the desired information can be found, but there is not much help with searching through the list of documents provided. In that vein, Smyth et al. [2] note that 70% of the time users are searching for information previously found by their friends or colleagues. For these reasons, there are recently developed tools, such as HeyStacks[2] or the +1 button of Google, that contribute to the sharing of interesting results.

Social search[3], a related term to village paradigm, is a type of search that uses social interactions, implicitly or explicitly, to get results or answers. One of the key aspects of social search is finding someone that can provide a relevant answer, it is the query routing. Considering social search, query-routing has been applied to several domains, such as Internet browsers[4], question answering[1,5,6] and recommender systems [7].

We see the automation of the village paradigm as a clear application of social machines, where each user is represented by an agent that works on behalf of him. These agents increase the users bandwidth allowing them to take more actions, and at the same time increase the availability of the users.

We consider working with unstructured P2P social networks for social search, as social networks are inherently bottom-up. Query routing can be automated and we consider this automation workable through intelligent agents. Each user has an agent that represents her, each of these agents has a contact lists that contains the agents owned by the people in the contact list of its owner. When a user u_i has an information need, she requests it to her agent a_i. The agent a_i first will check its knowledge base, if it can satisfy the information of its owner it will do it and finish its task. Otherwise, the agent will send the question to a subset of its contact list, and each of the agents that receive the message can either ignore it, answer it with their knowledge base, show the question to its owner with the aim that she answers, or forward the question to a subset of its contacts (in such case the process continues, as the new receivers can perform the same tasks).

For each question, the agents can play three roles:

- **Questioner**: The questioner is the agent who started the question.
- **Answerer**: An answerer is an agent who answers a question with the agent's own knowledge or its user's knowledge.
- **Mediator**: A mediator is an agent who receives a question and forwards it to others. They also may forward an answer that they receive from another mediator or an answerer, action that it is called "asking-next"

The classification of the query-routing algorithms used in those domains is based on whether the algorithms are unicast or multicast. In unicast query-routing algorithms, a query is only sent to one acquaintance or user [5,8,9] while multicast algorithms are the ones in which a query is sent to many of them[4,6,7].

Recently, we had proposed two multicast query routing algorithms in social search using search agents called "Asknext"[6] that used stop messages (SM) and "Question Waves" (QW)[10]. The second does not only improve significantly the number of relevance answers compared to the first one and much more compared to the generic Breadth-First Search (BFS)[4] but also reduces notably the number of exchange messages, which caused perturbing of searching process, between agents.

However, according to our recent experiments, QW and Asknext did not give a promising speed to social search as BFS. The reason of this decrease of the speed is because QW and Asknext introduce messages delay where the algorithms increase the scalability, that is good, but the time needed to satisfy a question is also increased, that is a drawback to be partly solved in this paper.

There are several possibilities to speed up the answers in this social search protocols and algorithms, notably reshaping the QA social network so that the relevant users or their agents are closely connected to questioners and thus relevant answers can get the questioners in lower time, by fine tuning the trust algorithms so that the real relevant users are at the top of the contact lists, natural language resources for matching more accurately the expertise, or introduce some learning capabilities to the agents regarding the answers they are asking-next.

This paper focuses on the last possibility, proposing simple learning mechanisms as a first step of the improvement of the speed of question answering in social networks. Neither in BFS, Asknext, and QW the questioners do not remember the answers sent by their friends (answerers) so the questioners need to request to the answerers again in the case they need again the same answers of their previous questions. In case that the agents were able to remember the answers that they received previously, it would be possible to improve the system scalability and reduce the time needed to satisfy a question.

To overcome this bottle-neck, we propose to grant agents with the ability of learning from the received answers. As a first, simple solution we propose to use a short-term memory model implemented as an embedded cache memory for each agent, so that each agent can get the reusable answer from its cache or from its closest friend cache quickly. Being it a workable solution will inspire us for future further expansions of the learning ability for the question answering agents.

Section 2 contains a brief literature review of social search algorithms used in our experiments including BFS, Asknext and QW. In Section 3, we detail the problem we contribute to. In Section 4, principles of agent's short-term memory are presented. In Section 5, simulations data are described. In Section 5, we show our experiment results. Finally, in Section 6, the conclusions and future work are presented.

2. On Multicast Query-routing Algorithms Applied in Social Search

Milgram's experiments[9], which were carried out in 1960s, consisted in a set of participants where each one was requested to deliver a document to a different target person. In case that the participant knew the target, she should send directly the document to; otherwise she must send the document to someone else more likely to deliver the document to the target person. The number of participants was 296, the 73% of which followed the instructions. Each time the document was forwarded, there was a probability that the receiver drops it. As a result, 64 folders reached their target person.

Under our point of view, Milgram's experiments can be considered as the basis of query-routing in unstructured P2P social networks. P2P query-routing algorithms can be classified as unicast or multicast. In the case of unicast query-routing, each requester sends the query to one and only one candidate. In the case of multi-cast the query is sent to more candidates at a time than the unicast, and many multi-cast algorithms use all the acquaintances as candidates.

The Social Query Model (SQM)[5], allows a better understanding of unicast query-routing. The SQM is a probabilistic model that indicates the probability of obtaining a relevant answer. SQM is represented in equation 1 and considers the following parameters:

- The expertise $e_i \in [0,1]$ indicates the probability that the node a_i answers a query, with a probability 1- e_i that the node a_i forwards the query.
- The correctness $w_i \in [0,1]$ indicates the probability that the answer provided by the node a_i is correct.
- The response rate $r_{i,j} \in [0,1]$ indicates the probability that a node a_j accepts a query from node a_i. The probability that a node a_j drops the query is then 1- $r_{i,j}$.
- The policy π_j of a node a_i is a probability distribution that indicates the probability of selecting neighbours of a_i (N_i) when a_i decides to forward a query; concretely, each π_j^i indicates the probability that a_i forwards the query to a_j.

$$P_i = w_i e_i + (1 - e_i) \sum_{j \in N_i}^{n} \pi_j^i \, r_{i,j} P_j \tag{1}$$

What we can extract from SQM and Milgram's paradigm is that, yet it is possible to find a target person, that in the case of social search would be someone able to provide a relevant answer, the probability of finding her is really affected by $r_{i,j}$ and w_i. Furthermore, if someone drops the query then the answer will not be found, and the requester will not know that the search process has been stopped. In the case of multi-cast query routing, where the requester and each candidate can send the request to many candidates, the process continues when someone drops the query.

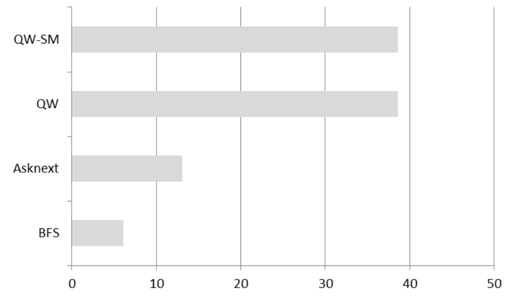
The most usual multicast query-routing algorithm is the Breadth First Search (BFS) or flooding; it has been used for example in [2] and [6]. In BFS, the requester sends the question to all her acquaintances. The receivers of the message answer or forward the question to all their acquaintances. BFS shows higher probability of answering the question, and will answer it faster too, but requires a high number of messages that threatens the system scalability. With the aim to limit the query propagation, BFS often uses the Time-To-Live (TTL). TTL is initialized to the value that indicates the maximum number of hops: each time that a question message is forwarded the TTL value is decreased by one hop, and when its value is 0 it cannot be further forwarded.

Recently, with the Asknext protocol [6], we proposed the usage of stop messages (SM) that aim to stop the question propagation when the information need related to the query message is satisfied. In our previous work, Asknext reduced 20 times the number of messages compared to BFS in a social network of almost 400 users. The idea is that when an agent receives an answer that satisfies the query it sends a SM to the other nodes that also forwarded the query. When an agent receives a SM, it stops searching answers and forwards the SM to the nodes that have been requested by it. SM needs that answer messages and stop messages travel faster than question messages, Asknext solves it introducing a delay in the question messages in base of the distance to the questioner; considering that the period of forwarding an answer or a SM is T_R and the current distance to the questioner is r , the period of forwarding the question T_F can be found using equation 2.

$$r < \left\lfloor \frac{T_F}{2T_R} \right\rfloor = \left\lfloor \frac{v_R}{2v_F} \right\rfloor \tag{2}$$

In average SM reduce the number of messages, but questions need more time to be satisfied.

More recently, we proposed the algorithm Question Waves (QW)[10,11] to take advantage of the properties of Asknext protocol. The idea of QW is delaying question propagation inversely proportional to the probability that the acquaintance will find the answer: the agents with high probability of satisfying the information need will be requested as fast as possible, while the acquaintances with a low probability will be requested after a high delay. In QW, each question message corresponds to an attempt of satisfying the information need. QW also introduces other mechanisms: delays in the answers with the aim that further and better answers arrive first. QW not only reduces the number of messages but its special characteristic is that the answers arrive sorted by relevance or probability of satisfying the information need. QW is naturally compatible with SM (QW-SM). In our previous work[10], we obtained that QW-SM needs 3.79 times less messages than BFS and 2.44 times less messages than SM to obtain a same recall, and QW-SM needs 16 times less effort to obtain the same recall, both cases when the relevant items taken into account have very high answer relevance. One of the drawbacks of QW is that the addition of delays increment the time needed to satisfy the information need, as can be observed in Figure 1. QW and QW-SM need 6.3 times more simulation steps to close a question than BFS. The objective of using a short term memory for each agent is reducing this increase of time needed to close a question without losing relevance nor increasing the number of messages.



Time needed to close a question (in simulation steps)

Fig 1. Time needed to close a question for each algorithm.

3. Problem to Solve: query routing for social search in unstructured P2P social networks.

Query routing for social search in unstructured P2P social networks aims to satisfy the information need of the users through requesting other users, or their agents. Considering the relevance of the answers received is as important as considering the time needed to obtain these answers, and the cost of obtaining these answers.

The relevance of the answers can be measured considering the relevance of the answers received until the question is closed. We consider that a question is closed when an agent considers the answer relevance of the answers received in a given moment is enough to satisfy the information need of the requester. For example, an agent might consider that the information need is satisfied with few high relevant answers or with a more answers of lower relevance. The measures we use for measuring the relevance of answers are the recall and the precision. The recall is theoretical measure yet not useful in practice because it requires full knowledge about the number of relevant items in the system, that normally is not possible to know. The recall is the proportion of the relevant items retrieved from the relevant items available in the whole social network. On the other hand, precision considers how many items are relevant out of the retrieved items; In our experiments we take the first three answers received to measure the precision.

The time needed to satisfy an information need can be measured as the time-elapse between the questioner agent sends the question until the last answer needed to satisfy the information need is received.

The cost of obtaining the answers can be measured with the effort that the user/agent put on solving the information need, which is estimated as the number of messages that the system uses. The best behaviour, far of being realistic, would consist in requesting only to one agent the question which was able to answer immediately with the best answer. It is not possible to be sure of obtaining the best answer without requesting all users and evaluating all answers. Moreover requesting directly the best candidate for question answering would require that the agent was able to communicate with her and she will attend the request: we consider that agents only attend requests sent by their acquaintances. In real scenarios, usually the answer quality would be directly proportional to the number of messages and the waiting time, while the objective is to maximize the answer quality and reduce waiting time and the number of messages.

We consider that there are mainly two approaches to improve the answers relevance and reduce the answering time and the number of messages at a time. The first one is improving the social network, while the second one consists in improving the query routing algorithms, or using the query routing algorithm that bests suits the application context. In this paper, we consider the query routing algorithms only, and more specifically the case that agents can handle all the messages and requests that they receive at each simulation step.

From the algorithms seen in section 2 we consider that the faster algorithm is BFS, as the time needed to close a question is twice the distance from the requester and the further required answerer. The drawbacks of this algorithm are that it requires many messages and the relevant answers that are far from the questioner agents may not be obtained or considered. Asknext reduces the number of messages at the cost of increasing the time needed to satisfy the information need, the time is increased as it uses delays in the question propagation, the number of messages is reduced as it uses stop messages, and the answer quality is equivalent to BFS. Question Waves uses delays in the question and answers propagation; this approach has many benefits, as reduces the number of messages and increase the answer relevance, but also it has an important drawback: it is needed more time to close a question. Question Waves provide the best answer relevance and the minimum number of messages from the algorithms considered, but it is far the slowest algorithm. In this paper we would try to improve Question Waves speed, although that the solution provided may reduce slightly the answer relevance and increase the number of messages.

4. Using Cache as Agent's Short-term Memory

4.1. The Principal of the Proposal Short-term memory

Smyth et al.[2] note that 70% of the time users are searching for information previously found by their friends or colleagues.

Recently, we discussed that one of the motivations to sending back the message to the inverse path that the question followed, was the possibility that all the agents in the chain may have the opportunity to learn the answers[6].

With these regards, the idea of this paper, is to temporarily store the answers passed through each agent. We believe that the fact of temporarily store the answers by using agent's short-term memory can reduce the number of messages at the same time that reduces the time needed to satisfy the information needs.

4.2. Using LRU Cache as Short-term Memory:

In the previous studies (BFS[4], Asknext[6] and Question Waves[10]); the answer stored only in the answerers so when several agents request the same question, they need to get the answer only from those answerers. This principle makes the social search slower due to query-routing duration to get the answer.

To solve this problem, we embedded a short-term memory to each agent. So when an agent receives an answer which is not in its knowledge base, it will add this answer to its short-term memory; and the agent can reuse the answer when a related question is received. The process, that each agent access to its short-term memory will be detailed in section 4.3.

The principle of the search agent's short-term memory is based on the basic and useful cache technic called least recently used cache (LRU Cache)[12]. The LRU cache evicts the element that was accessed least recently when the cache is full.

The rationale for choosing the least recently used element is, if an element has not been accessed in a while then it may not be accessed again, based in the principle of temporal locality. Each time an element is accessed we check if it is in the queue and remove it and place it at the back of the queue. Therefore, the least recently used element is always at the front of the queue. We consider that has sense to implement the short-term memory as a cache LRU, as the items accessed are refreshed by the agent, and the ones that are not in use at the end are forgotten. In the social search case, a cache element could be composed of: a question, an answer, an answerer (an agent who answered the question), question effort, date of answer.

4.3. Agent's Answer Retrieving Process

When a question arrived, the ordinary search agent will look in its knowledge base (KB). If it did not find the answer, then it forwards the question to other agents. But if it find the answer, agent needs to decide whether to send its answer only (high confidence case); or to send its answer and also forward the question to other agents (low confidence case). Figure 2 illustrates the answer retrieving process of the agent without short-term memory.

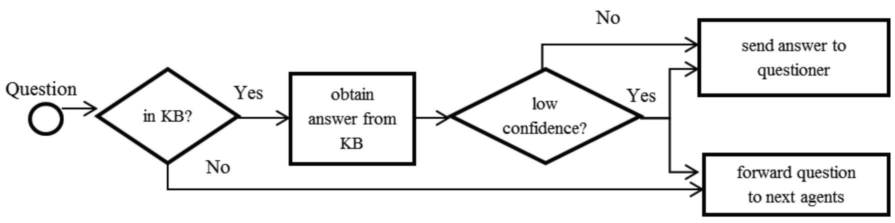


Fig 2. The answer retrieving process of search agent without short-term memory.

In the case that we embedded a cache to each agent, the process to retrieve the answer will be illustrated as Figure 3. The process is slightly different from the agent without cache. The answer will retrieve from the cache only when there is no answer in the agent's knowledge base or the agent did not confident in its answer.

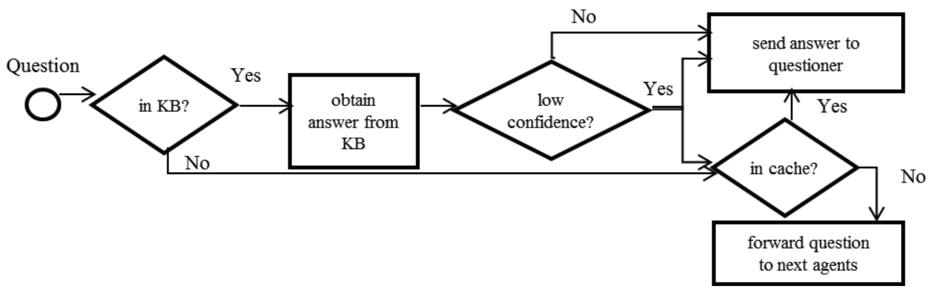


Fig 3. The answer retrieving process of search agent with short-term memory (LRU cache).

It is important to mention that a question composed of two parameters: question value and questioner id. An answer composed of four parameters: question value, answer value, answerer id and confidential level of the answerer.

On the other hand, when an agent using cache receives an answer, it will put the answer to the cache before transferring the answer to other agents.

4.4. Using Effort for Optimization

Cache was believed for long time to reduce the traffic and increase speed to retrieve the information especially in computer networking. But the problem of the cache is how to know the information in the cache is still relevant.

To increase the relevance level of the information, two solutions should be considered. First solution is the agent need to inform others agents if its answer was updated. Second solution, the questioner can put more effort to the question that need strong relevant answer. It means that the question with more effort will not get the answer from the cache but from the original answerers. The first solution will give the best relevance answer in the cache but it will perturb the traffic of the network if million agents occurred and it is not the goal of our studies. In this paper, we use the second solution to optimize the answer retrieving process of the agent with cache. The process is illustrated in the Figure 4.

The different between the agent with cache (Figure 5) and the agent with cache and with effort optimization is that when retrieving the answers from the cache, the agent need also to compare the effort of that question stored in cache (effortCache) with the effort of the input question (effortQuestion). If the effortQuestion is bigger than effortCache, the answer in the cache will not consider and the question will be forwarded to another agents. But if the effortCache is bigger, the answers in the cache will be sent to the questioner.

In this case, the input question composed of three parameters: the questioner id, the question value and the effort value. Moreover, each answer stored in the cache must contain five parameters: the question value, the answer value, the answerer id, the confidential level of the answerer and also the effort value of the question stored in the cache.

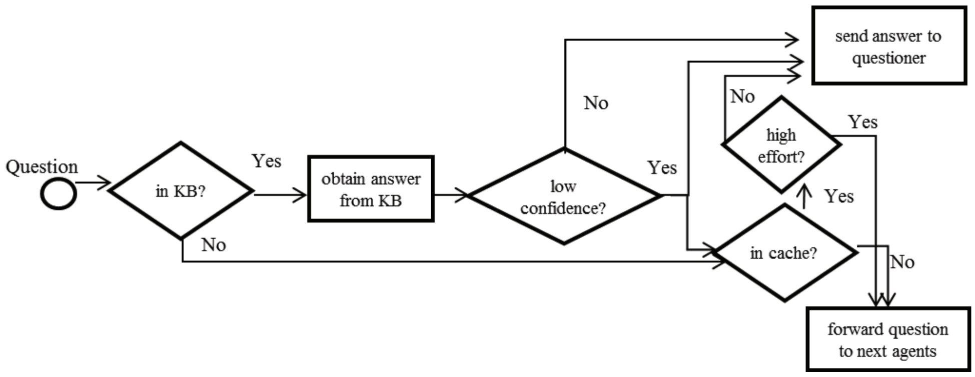


Fig 4. The answer retrieving process of search agent with cache and with effort optimization.

5. Simulation Data and Implementation Details

For our simulations we used the same data that in our previous works[6,10], it contains 19,463 questions distributed in 387 users. Questions are distributed randomly between the steps 1 and 1,000 of simulation. All of the simulations end at step 2,000. We run 20 instances of each configuration.

As a result, the number of questions is 19,463, and there are 387 users. We used SN1[10] as social network. SN1 has 3089 links, a clustering coefficient of .53, an assortativity of .0401, an average path length of 3.02 and a diameter of 5.

How many relevant answers are per question is represented in Figure 5, while Figure 6 represents the frequency of the questions.

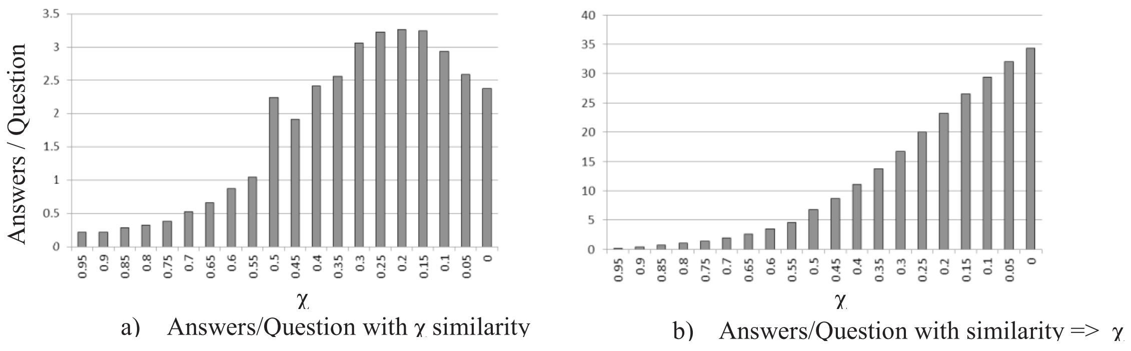


Fig 5. Number answers with a relevance of χ (a). (b) shows the accumulated representation, or answers with a relevance => χ.

Collaborative Filtering Recommender Systems data, is adequate for social search simulation, as this recommender systems can be seen as Social Feedback Systems [3].

Collaborative filtering (CF) is used to recommend items that similar users have liked. CF is based on the social practice of exchange opinions. In the 1990s, GroupLens worked with this family of algorithms[13,14], leading a research line on recommender systems that expanded globally because of growing interest in the Internet. Usually, these systems rate items between 1 and 5. Pearson's correlation[14,15] can be used to measure the similarity between users. In this paper we will use QW as in[10].

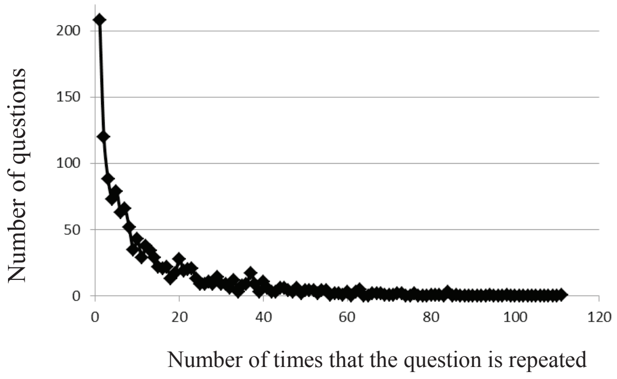


Fig 6. Representation of how many times questions are repeated.

6. Results and Discussion

The results that we consider that should be analyzed are the number of messages, the time needed to close an answer, the precision and the recall. In Table 1 we show the relative differences from not using cache and using a cache of 2000 answers.

Table 1. Relative difference of applying a cache of 2000 answers.

	Effort				No Effort			
	SM	BFS	QW	QW-SM	SM	BFS	QW	QW-SM
Time	-16.84%	-11.17%	-13.45%	-13.23%	-29.09%	-19.95%	-18.77%	-18.78%
Messages	-27.6%	-2.9%	-6.95%	-15.01%	-38.9%	-3.07%	-9.09%	-20.92%
Precision	1.02%	0.18%	2.69%	2.61%	-2.41%	-0.66%	-0.12%	-0.13%
Recall	-14.59%	-14.24%	1.68%	1.64%	-25.35%	-25.05%	-0.91%	-0.96%

We can observe that applying a cache without Effort, reduces more the time needed to close an answer and the number of messages, but there are higher loses of precision and recall. The results also show that applying effort there is no loss of precision and the recall is only decreased for SM and BFS. The algorithm that has higher decrease of time and messages is the SM. We consider that BFS and SM have a high drop of recall using short term memory, and it will not be available at all. In the other hand, QW and QW-SM have a small loss of precision in recall when cache without effort is applied and when they use an effort cache, the precision and recall increases slightly. The decision of using a cache with or without effort in the case of QW and QW-SM will depend on the context; if the small reduction of relevance (precision and recall) is assumable then is better to use the cache without effort.

We also observed that the number of messages is more reduced in the algorithms that use stop messages (SM and QW-SM)

7. Conclusions and Future Work

From our experiments we can obtain that using short-term memory in agents reduces the time needed to satisfy an information need, and at the same time reduces the number of messages needed. But in the case of the algorithms SM and BFS there is a considerable reduction of the recall, although that the usage of a cache with effort reduces the decrease of recall, it also provide less improvement in the reduction of time and number of messages.

In the case of QW and QW-SM the decrease of relevance is small, and it is increased slightly when an effort cache is used. We consider that if the small loss of relevance is acceptable it should be better to use the cache without effort for QW and QW-SM.

The time needed to close a question using the short-term memory is 18.77% lower in the case of cache without effort, and of 13.23% lower in the case of cache with effort; QW with short-term memory is still slower than BFS and Asknext. Considering QW-SM and QW, in the case of cache with effort the recall increased 1.9% and the precision a 2.6%; in the case of cache without effort the recall decreased a 0.96% and the precision a 0.13%. In the case of QW and QW-SM our experiments show that the reduction of relevance with this data is smaller than a 1%, while the speedup can be improved a 19.78%.

As future work, it should be studied the possible loss of relevance due the un-updated answers contained in the cache, that will not contain the most fresh information available, this feature will depend on the context, and can be taken into account deleting the answers in the short term memory after a time period. Other approaches based into improving the social network to speed-up the results also will be considered.

8. Acknowledgements

This work was supported in part by by the EU's 7FP under grant agreement no 316097, by the TIN2013-48040-R (QWAVES) Nuevos métodos de automatización de la búsqueda social basados en waves de preguntas, the IPT20120482430000 (MIDPOINT) Nuevos enfoques de preservación digital con mejor gestión de costes que garantizan su sostenibilidad, VirCoin2SME – num. H2020-MSCA-RISE SEP 210165853. Social, complementary or community virtual currencies transfer of knowledge to SME: a new era for competitiveness and entrepreneurship, and VISUAL AD, RTC-2014-2566-7 and GEPID, RTC-2014-2576-7, as well as the grup de recerca consolidat CSI-ref. 2014 SGR 1469.

References

1. Horowitz, D. & Kamvar, S. D. The anatomy of a large-scale social search engine. *Proc. 19th Int. Conf. World wide web WWW 10* **22,** 431 (2010).
2. Smyth, B., Briggs, P., Coyle, M. & O'Mahony, M. in *User Model. Adapt. Pers. SE - 27* (Houben, G.-J., McCalla, G., Pianesi, F. & Zancanaro, M.) **5535,** 283–294 (Springer Berlin Heidelberg, 2009).
3. Chi, E. H. Information Seeking Can Be Social. *Computer (Long. Beach. Calif).* **42,** 42–46 (2009).
4. Wu, L.-S., Akavipat, R., Maguitman, A. G. & Menczer, F. in *Soc. Inf. Retr. Syst. Emergent Technol. Appl. Search. Web Eff.* (Goh, D. & Foo, S.) 155–178 (IGI Global, 2007). doi:10.4018/978-1-59904-543-6
5. Banerjee, A. & Basu, S. A social query model for decentralized search. in *... 2nd Work. Soc. ...* (2008). at <http://www-users.cs.umn.edu/~banerjee/papers/08/sqm-snakdd.pdf>
6. Trias I Mansilla, A. & De La Rosa I Esteva, J. L. Asknext: An agent protocol for social search. *Inf. Sci. (Ny).* **190,** 144–161 (2011).
7. Walter, F. E., Battiston, S. & Schweitzer, F. A model of a trust-based recommendation system on a social network. *Auton. Agent. Multi. Agent. Syst.* **16,** 57–74 (2007).
8. Michlmayr, E., Pany, A. & Kappel, G. Using taxonomies for content-based routing with ants. *Comput. Networks* **51,** 4514–4528 (2007).
9. Travers, J. & Milgram, S. An Experimental Study of the Small World Problem. *Sociometry* **32,** 425–443 (1969).
10. Trias i Mansilla, A. & de la Rosa i Esteva, J. L. Question Waves: A multicast query routing algorithm for social search. *Inf. Sci. (Ny).* **253,** 1–25 (2013).
11. Trias Mansilla, A. & de la Rosa Esteva, J. L. Propagation of Question Waves by Means of Trust in a Social Network. in *Flex. Query Answering Syst. SE - 17* (Christiansen, H. et al.) **7022,** 186–197 (Springer Berlin Heidelberg, 2011).
12. King, W. . Analysis of Paging Algorithms. in *IFIP Congr.* 485–490 (1971).
13. Konstan, J. A., Kapoor, N., Mcnee, S. M. & Butler, J. T. TechLens : Exploring the Use of Recommenders to Support Users of Digital Libraries. in (2005). at <http://www.grouplens.org/papers/pdf/CNI-TechLens-Final.pdf>
14. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. GroupLens : An Open Architecture for Collaborative Filtering of Netnews. in *Proc. 1994 ACM Conf. Comput. Support. Coop. Work* (Furuta, R. & Neuwirth, C. M.) **pp,** 175–186 (ACM, 1994).
15. Shardanand, U. & Maes, P. Social information filtering: algorithms for automating "word of mouth. in *Proc. ACM Conf. Hum. Factors Comput. Syst.* (Katz, I. R., Mack, R., Marks, L., Rosson, M. B. & Nielsen, J.) **1,** 210–217 (ACM Press/Addison-Wesley Publishing Co., 1995).