A Multiagent Recommender System with Task-Based Agent Specialization

Fabiana Lorenzi[1,2] and Fabio A. C. Correa[1] and Ana L. C. Bazzan[1] and Mara Abel[1] and Francesco Ricci[3]

[1] Instituto de Informatica, UFRGS
Caixa Postal 15064, 91.501-970 Porto Alegre, RS, Brazil
faccorrea,bazzan,abel@inf.ufrgs.br
[2] Universidade Luterana do Brasil
Av. Farroupilha, 8001 Canoas, RS, Brazil
lorenzi@ulbra.br
[3] Free University of Bozen-Bolzano
Bolzano, Italy
fricci@unibz.it

Abstract. This paper describes a multiagent recommender system where agents maintain local knowledge bases and, when requested to support a travel planning task, they collaborate exchanging information stored in their local bases. A request for a travel recommendation is decomposed by the system into sub tasks, corresponding to travel services. Agents select tasks autonomously, and accomplish them with the help of the knowledge derived from previous solutions. In the proposed architecture, agents become experts in some task types, and this makes the recommendation generation more efficient. In this paper, we validate the model via simulations where agents collaborate to recommend a travel package to the user. The experiments show that specialization is useful hence providing a validation of the proposed model.

1 Introduction

Internet is a rich source of information where users search information about products and services related to their interests and preferences. However, this has generated new information problems. In fact, the overabundance of information may *overload* the users and ultimately can make very hard to locate the *right* information [9]. Moreover, the information required for a topic or a service (e.g., rent a car) is usually distributed in several repositories.

In order to cope with these issues, Recommender Systems have been proposed [11]. They are based on data mining algorithms, are capable to learn about user preferences over time, and can automatically identify relevant products or information that fit the user needs [1]. Different approaches are used in their core recommendation algorithms, such as collaborative filtering, content-based filtering or knowledge-based approaches [1]. Recommender systems have been applied in several domains [5] such as book recommendation (amazon.com) or

movie recommendation (netflix.com). It is worth noting that these web sites can fully support the purchase (or rent) process, they manage huge catalogues of products and they can rely on the user information acquired by direct contact with their customers (logs).

However in some application domains it may be possible that a single information source (web site) does not manage all the information needed to run the full recommendation process. The data available may be fragmented, overspecialized or overgeneralized, or even irrelevant to the recommendation at hand. There are several data sources and services distributed in Internet, which are not always available, and sometimes can they offer information that can be ambiguous or erroneous.

To cope with these problems, we propose the application and integration of two technologies: distributed recommender systems and multiagent technologies. We claim that a multiagent recommender system can be applied for retrieving, filtering and using information that is relevant to the recommendation task, and can better deal with the dynamic changes in the information source compared with more traditional non-distributed recommender systems [2, 10].

In the tourism domain, for instance, a travel package recommendation is typically supported by several service providers for flights, hotels and attractions. Moreover, specific knowledge is required to assemble all the components [12]. Usually this information cannot be found in a single repository. The tourism market is by its nature distributed, and several service providers and intermediaries manage and store in their databases (or in informal repositories) service information and users data [17]. In order to recommend a travel package, an intermediary (travel agent) must construct a model representing all the elements (information) required for generating this recommendation. This model can be implicitly defined in her mind, or explicitly documented in a formal plan in the travel agency. These elements would include resources (information, products or services), customers and their requirements, factors influencing the recommendation (such as the season), immediate strategies for finding the best options for the user, and so on. However, planning a travel and building a package is not performed by an intermediary alone. Collaboration among travel agents is required to integrate individual experiences into a coherent plan that satisfies user's preferences.

The main goal of a Multiagent Recommender System is therefore to implement this cooperation among the agents. Each agent should work as an expert and participate to the composition of the final recommendation. This work presents a distributed, and knowledge-based, recommender system implemented in a multiagent environment. The recommendation computation (travel package) is based on the collaboration of multiple agents exchanging information stored in their local knowledge bases. A recommendation request is decomposed into sub-tasks handled by different agents, each one maintaining its own knowledge base and working as an expert helping to compose the final recommendation. The proposed model supports agent specialization, i.e., the agents become experts in specific tasks. This agent specialization mimics what happens in the real

world, where it is common for travel agents to specialize in a particular kind of service (hotel, flights, interchanges, conferences, etc) in order to provide better and better advises to the customers. Specialization increases agent's confidence and improves quality of service. We want to replicate this in our multiagent scenario, and generate expert agents by letting them to specialize.

This paper is structured as follow: the next section discusses some related work on Multiagent Recommender Systems and Recommender Systems. Section 3 describes the multiagent recommender model and Section 4 presents the experiments we conducted. Finally section 5 summarizes the paper's contributions.

2 Related work

Our approach explores the ability of multiagent systems to decompose a complex recommendation problem into smaller ones. This allows specific knowledge to be applied in solving each subproblem and updated when changes occur in the domain. The baseline technologies applied in our research are introduced in this section.

2.1 Recommender Systems

As mentioned above the three major recommendation techniques are: collaborative filtering, content-based filtering and knowledge-based systems. The most popular technique is collaborative filtering that aggregates customer's preferences, expressed as product ratings, to recommend new products [11]. In collaborative filtering the system predicts the target user ratings and select the products with highest rating. However, a large number of ratings from similar users are required to build reliable recommendations for a target user. This is hard to achieve and "data sparsity" is the primary source of erroneous recommendations.

Amazon.com is a very popular e-Commerce site that exploits collaborative-filtering. In its book section for instance, the system encourages direct feedback from customers about books that they have already read [13]. After this, the customer may request recommendation for books that he/she might like.

In content-based filtering the preferences of a specific customer are exploited to build new recommendations to her. NewsDude [4], for instance, observes what online news stories the user has read and not read and learns which articles the user may be interested in reading. In content-based systems only data related to the current user are exploited in building a recommendation. It requires a description of user interests that either matches the items' catalog or provides an input for the user model that was learned in order to output a recommendation.

Collaborative and content-based filtering can deliver poor recommendations if not trained with an adequate number of examples (product ratings or pattern of user preferences). This limitation mostly motivates the knowledge-based approach. A knowledge-based system learns about user preferences over time and

automatically suggests products that fit the user model. This technique tries to better use preexisting knowledge, which is specific to the application domain, in order to build a more accurate prediction model still based on a limited number of training instances.

Usually knowledge-based approaches, such as case-based reasoning [14, 6], are combined with collaborative or content-based filtering to provide better recommendations. Knowledge about customers and the application domain are used to reason about the products that fit the customer's preferences. The most important advantage of the combined approach is that it does not depend (exclusively) on customer's rates, hence avoiding the mentioned difficulty to bootstrapping the system. The knowledge that improves the recommendation can be expressed as a detailed user model; a model of the selection process or a description of the items that will be suggested.

Travel recommendation is a complex task and there are still many open issues. In this paper we are mostly focussed on the integration of different information sources distributed over the Internet, and to exploit experts' specific knowledge in the recommendation. In order to deal with these issues, we propose a multiagent recommender system.

2.2 Multiagent Recommender Systems

Multiagent systems (MAS) can be applied to retrieve, filter and use information relevant to recommendations [16] [8]. MAS can deal with distributed information sources and there are several advantages in developing these systems [15], such as: 1) the information sources may be already distributed, and it would be wasteful to replicate agent information gathering and problem solving capabilities for each user and each application; 2) agents can interact flexibly in new configurations on demand; and 3) the system performance can degrade gracefully when some agents are out of service temporarily.

Classical recommendation technologies can be described as *single agent*, as one single intelligent system provides the recommendation function [10]. In a multiagent recommender system a collection of interacting agents manage the recommendation generation process, trying to improve the recommendation quality obtained by a single agent. The agents cooperate and negotiate in order to satisfy the users, interacting among themselves to complement their partial solutions or even to solve conflicts that may arise.

CASIS [7] is an example of multiagent case-based recommender system [14, 6], where the authors proposed a metaphor from swarm intelligence to help the negotiation process among agents. The honey bees' dancing metaphor is applied with case-based reasoning approach to recommend the best travel to the user. The recommendation process works as follows: the user elicits her preferences; the bees visit all cases in the case base and when they find the best case (according to the user's preferences) they dance to that case, recruiting other bees to that case; and the case with the most number of bees dancing for it is the one recommended to the user.

The advantage of this application is that the bees always return some recommendation to the user. Normally, case-based recommender systems use similarity-based retrieval to identify the best cases. The recommendation results depend on the retrieval algorithm (similarity function, similarity threshold, feature weights, etc.) and sometimes the system does not find cases that sufficiently match the probe (problem definition). In a real travel package recommendation, the travel agent always recommend something to the user, even if there is not a travel package that matches the customer's preferences. The travel agent always provides to the customer an answer, as the customer could always switch to another travel agency. Because of this, CASIS always returns some recommendation. However, the main disadvantage of this system is that the case-base is unique and centralized. It is not possible to search information located in other sources.

Another example of multiagent recommender system is presented in [8]. This system arranges meetings for several participants taking into account constraints for personal agendas. In this system, three different agents were proposed: the *personal assistant agent* is the interface between the user and the MAS; the *flight travel agent* is connected to a database of flights; and the *accommodation hotel agent* is responsible to find an accommodation on the cities involved in the meeting. A disadvantage found in this approach is that the problem is not solved dynamically because the recommender system has to collect information from different information gathering agents to model the problem-solving.

3 MAS Recommendation Model

This section describes the proposed Multiagent System (MAS) approach and uses the tourism domain as motivating scenario. Planning a travel is a difficult task even for an expert travel agent. She needs to know many details about the destination chosen by the passenger and many features of the whole trip such as the attractions' details, hotels or flights times and costs. In several cases, this knowledge is distributed among different travel agents, and they must communicate and exchange information to compose the final recommendation.

3.1 The Agents

Using the travel recommendation example, we have created a multiagent system where a community of agents share a common goal (the travel package recommendation) as well as individual goals (the travel components that each one must identify). A community C of agents consists of n Searcher (*Src*) agents $a_1, a_2, ..., a_n$.

When the user asks for a recommendation, a list of tasks, here restricted to flights, hotels or attractions, is created and communicated to the *Src* agents. The agents choose a task from the received list and perform it. When a task is performed by an agent, it is marked as *not available*, which means that another agent can perform that task at the same time.

The *Src* agent is represented by $a = (P, LocalKB)$ where P is the agent's profile and *LocalKB* is the agent knowledge base.

The agent's profile is defined as $P = (id, status, tcurrent, confind)$; *id* is the identification id of the agent, *status* indicates if the agent is *online*, *offline* or *not available* to perform a task. The agent can be offline if, for example, the computer is switched off or the network is down. In this case, it will not be able to perform a task. When the agent is performing a task its status becomes *not available*. *tcurrent* is the current chosen task and *confind* are the confidence indexes of the agent to each type of task.

Each confidence index is calculated through the number of tasks performed of the respective type and the evaluation of each task done by the user. Each task solved by an agent receives an evaluation from the user, with a rating ranging from 0 to 10, where 0 is the worst and 10 is the best rating. The task evaluation is then used in the agent confidence index computation so that the agent increases more the confidence when it solved better the task.

Two features are desired when addressing the set of evaluations: quality and uniformity. The first one is quite obvious, the goal is to give the best recommendation to obtain a high evaluation. The second one relates to the evaluations homogeneity. Here, the lower the variability of evaluations the more reliable will be the results. Thus, we search for a set of evaluations with a high average evaluation and low standard deviation.

$$confind^{(n)}(t) = \frac{confind^{(n-1)}(t) + F^{(n-1)}(t)}{\sum_{i \in T} F^{(n-1)}(i)} \times evaluation \tag{1}$$

Equation 1 shows the *confind* (confidence indexes) update function of an agent, where t is the task type, T is the set of task types, $confind^{(n)}(t)$ is the new confidence index of task type t, $F^{(n)}(t)$ is the number of tasks of type t performed by the agent in the instant n. Finally, the *evaluation* is defined as follow:

$$evaluation = \begin{cases} \alpha \left(\frac{\mu(t)}{\sum_{j \in T} \mu(j)} \right) & \text{if } \sum_{k \in T} \sigma(k) = 0 \\ \alpha \left(\frac{\mu(t)}{\sum_{j \in T} \mu(j)} \right) + (1 - \alpha) \left(\frac{\frac{1}{\sigma(t)}}{\sum_{k \in T} \frac{1}{\sigma(k)}} \right) & \text{otherwise} \end{cases} \tag{2}$$

where: $\mu(t)$ is the evaluation average of type task t performed by the agent; $\sigma(t)$ is the evaluation standard deviation of type task t performed by the agent.

In order to get the normalized evaluations average value of the type task t, the value is divided by the sum over all task types evaluations. The standard deviation value is calculated in the same way. Since the formula is a recurrence equation, it was necessary to set initial conditions, such as:

- Each agent has performed one task of each type;
- Each performed task got an evaluation equal to 5 (a neutral rate);
- There is no standard deviation in instant 0, which means that the standard deviation was not considered in instant 0.

The α coefficient determines the relevance of the average and the standard deviation over the confidence indexes. In the experiments, described later, α was set to 0.5. That means that the average and the standard deviation have the same importance in the calculation of the confidence index.

Thus, the confidence index must be proportional to the average and inversely proportional to the standard deviation, which means that we will get best results with a high average evaluation and a low standard deviation. Equation 2 expresses the influence that the tasks evaluations have over the confidence index.

The agent updates the confidence indexes every time it must choose a task to execute. Based on those indexes, it can check which task type, among those available, is better considering. However, sometimes the agent can be forced to choose a task that is not its specialty (for example, a flight recommendation is required but the flight expert is missing and only the (agent) expert in hotels is available).

This behavior helps the agent to become an expert in a task type. The agent specialization improves the system performance. If the agent has enough information about a task type in its knowledge base, it can provide a faster recommendation for that kind of service. But more importantly, it gives, in the long run, high quality recommendations because the agent becomes an expert in that type of recommendations.

LocalKB stores the knowledge the agent has already used to solve previous recommendations. Figure 1 shows the agent's model. The user's preferences and the list of tasks (appearing inside the box) do not belong to the agent, but they are needed to understand the agent's LocalKB. *Id* is the task's identification. *type* is the task's type (flight, hotel or attractions) and *timespent* represents the time spent by the agent to perform the task. *requirements* represent the user's preferences, i.e., the user's query. *Solution* represents the task already performed by the agent. The combination of the *requirements* and the *solution* attributes represents a case, i.e., the description of the problem and the description of the solution, respectively.

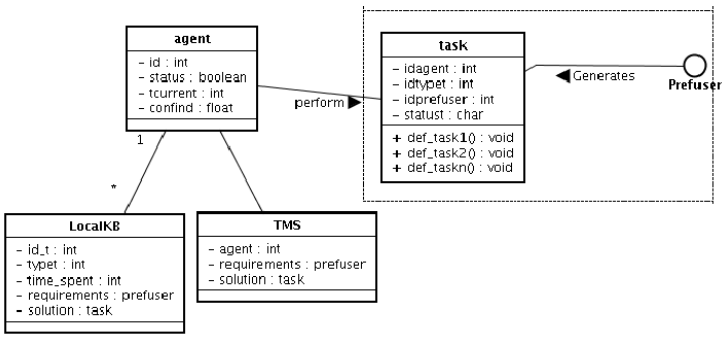

Fig. 1. Agent's model

The agent's knowledge base is increased with the number of tasks it decides to save. The bigger the knowledge base becomes, the worse becomes the search performance. On the other hand, a small knowledge base forces the agent to search in the community or in the Web, which results in a waste of time and slower performance. For this reason, it is important for the agent to control the size of its knowledge base. It must have enough knowledge to give good recommendations. The confidence indexes help to control the knowledge base size according to the tasks the agent has performed. After performing the task, the agent checks if that task has increased or not its confidence index and it decides to keep or not the task in its knowledge base.

3.2 The Recommendation Algorithm

As shown in the previous subsections, each agent performs a task (of type flight, hotel or attraction) to contribute to the final complete recommendation, i.e., a travel package.

When the user explicit her preferences, the recommendation process starts and the list of tasks is created. Let us now consider a hypothetical scenario where the user has chosen Paris as destination to her vacation; she would like to travel on March 10th, to stay in a three-stars hotel and would like to know the possible attractions in Paris. Three new tasks are created with these preferences.

After that, each agent picks a task (considering its confidence indexes) and turns its status to *not available*. This choice is made in a decentralized manner, where the agent chooses the task it has the highest confidence index. A task can be performed only by one agent which means that two different agents cannot pick the same task. When an agent chooses a task, it chance its status and this task is not available anymore.

Algorithm 1 Multiagent Recommendation

```
{C is the agent community}
{T is the set of tasks to be solved}
Procedure Recommendation (agent_x, t, C)
repeat
   taskToBeSolved = PickTask(agent_x, t)
   T = T \ {taskToBeSolved}
   Solution = SearchLocalKB(taskToBeSolved)
   if Solution = ∅ then
      for each agent_x ∈ C
      Solution = CommunitySearch(agent_y, taskToBeSolved)
      if Solution = ∅ then
         T = T ∪ { taskToBeSolved }
      end if
   end if
until (T ≠ ∅)
UpdateConfind()
```

The agent might find the information necessary to solve the task in two ways: searching in its own knowledge base or exchanging information with agents in the community. As shown in algorithm 1, two levels of information search are proposed.

Through the *SearchLocalKB* procedure, agents search for the information necessary to complete their selected tasks in their own knowledge bases. Each agent has a knowledge base that stores the task episodes already performed by the agent. These episodes are stored as cases with attributes describing the user's preferences (the description of the problem) and attributes describing the task itself (the solution of that recommendation episode).

If the agent does not find the information required to complete the selected task in its knowledge base, then it proceeds with the second type of search, here called *CommunitySearch*. In this stage, the agent communicates with the available agents to exchange information. This communication is important because the agent can find another agent that is expert in the selected task and exchanging information with this expert agent, it will better solve the task. It is important to note that the *CommunitySearch* is not exploited in this paper and it will be detailed in a future work.

4 Experimental Results

In order to validate the multiagent recommender approach in the tourism domain, a preliminary experiment was done, where we have simulated different users asking for different recommendations and 35 users queries were generated. From these queries, 100 new tasks to be performed were created (i.e, the list of tasks was created from these new queries).

The agents were developed in the JADE (Java Agent Development Framework) framework [3]. An acquisition knowledge step was done in a travel agency and the *Src* agents' knowledge bases were populated randomly with real cases of clients of this agency.

Here we assume that, at instant 0, each agent has performed one task for each type and these tasks got an evaluation equal to 5 (a neutral rate). That means that the agent has all confidence indexes with the same value and therefore at the beginning it will choose the task randomly.

Another goal of the experiments was to validate the scalability of the system. For this reason, the experiments were done with different number of available agents. We have calculated the average evaluation of agents in each solved task and we shall show how this value changes during the whole tasks performed by the agents.

We have run the same experiment under two conditions, to show that specializing the agents can be useful. In the first one, the agents confidence indexes were calculated and the agents used these values to select the task to execute. In the second one, the agents did not considered the indexes and they chose the tasks randomly.

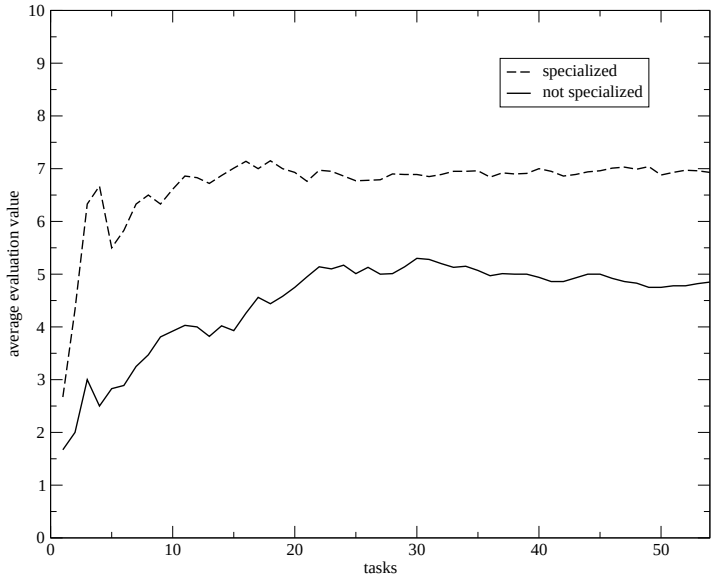

Fig. 2. Average evaluation value - 3 agents (54 tasks)

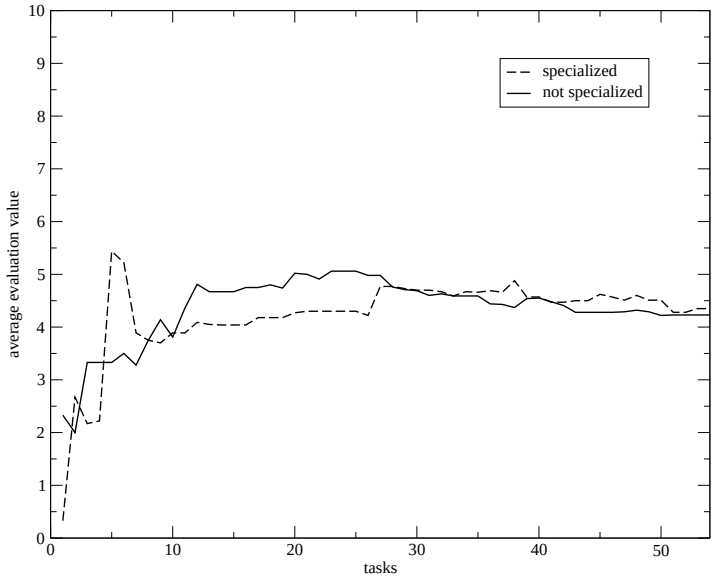

Fig. 3. Average evaluation value - 10 agents (54 tasks)

Figure 2 shows the average evaluation values for the 54 tasks performed, where 3 agents were available to perform tasks. The average evaluation value of

expert agents was high (almost 7) which means a better result comparing to the non-expert agents that reached only 5.

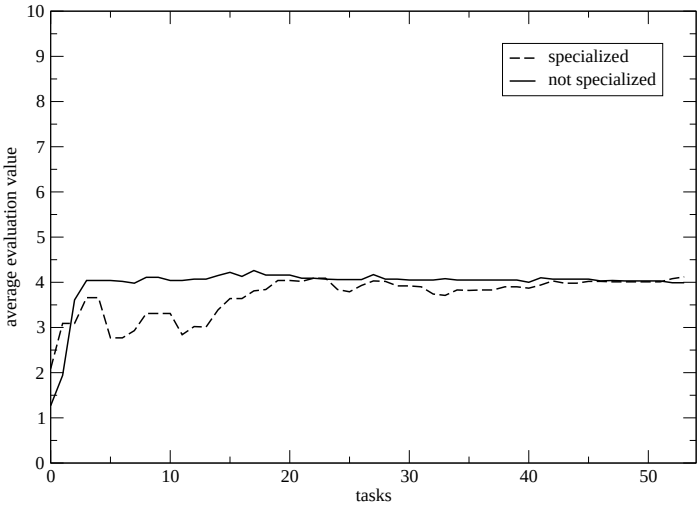

Fig. 4. Average evaluation value - 15 agents (54 tasks)

In the second run, 10 agents were available to perform tasks. Since we have more agents for the same number of tasks, and the same number of task types, the individual performance decreases. The objective was understand how the agents behave if we increase the number of available agents to perform the tasks. Figure 3 shows that, as expected, that increasing the number of agents and keeping the same number of tasks, the average evaluation value is lower for the specialized agents, than the previous scenario (with 3 agents).

Figure 4 confirms the behavior shown in the previous figure. Here we have 15 agents and 54 tasks and the average evaluation value was around 4 for both agents: expert and non-expert. Similarly the results from the previous experiment, as larger the number of agents performing tasks, lower is the average evaluation value.

We also have run a second experiment, where we increased the number of tasks to perform (and kept the same number of types). In this experiment we had 100 tasks to be performed. The same number of agents were used in the experiments: 3, 10, and 15.

Figure 5 shows the average evaluation value of specialized and non-specialized agents, through the 100 tasks performed, with only 3 agents available. Both

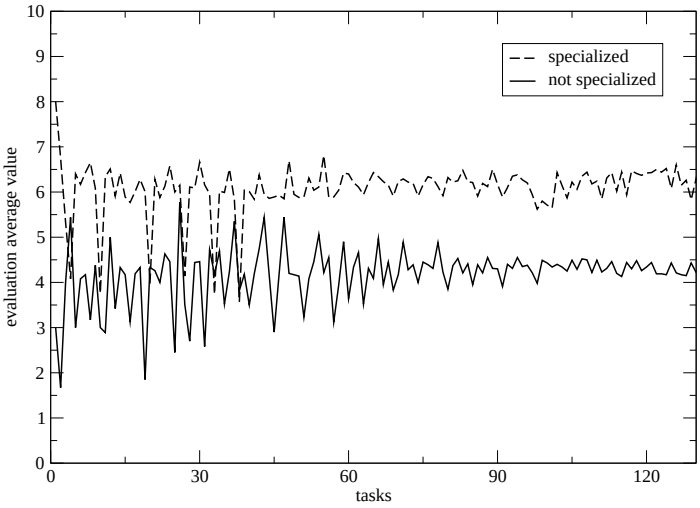

Fig. 5. Average evaluation value - 3 agents (100 tasks)

non-expert and expert agents had unstable average evaluation values in the beginning. However, the expert agents had better results in the end.

In figure 6 we can see the average evaluation value of expert and non-expert agents with 10 agents working. Expert agents got better results than non-expert agents. They reached 4.5 of evaluation average value and the result was stable starting from the thirties task. Considering that we have 10 agents and 100 tasks to be performed, the results of expert agents were good. It was almost the same results got in the experiment with less tasks.

Figure 7 shows the average evaluation value when we used 15 agents working to solve the tasks. Despite the expert agents started with unstable values, the evaluation average became stable soon and kept the same performance until all the tasks were completed.

5 Conclusions

This paper presented a distributed knowledge-based multiagent model applied to the tourism domain. The agents are cooperative and recommend travel packages to the user.

A recommendation is divided in small pieces called tasks and each agent is responsible to perform some tasks not necessarily for the same travel-package request. As long as the agents will perform different tasks, they become experts

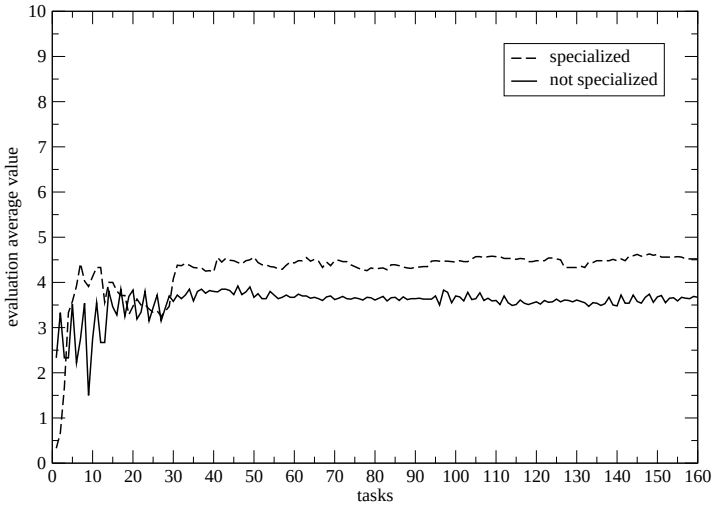

Fig. 6. Average evaluation value - 10 agents (100 tasks)

in a specific task type. The agents become travel agents, where each one have specific knowledge and the cooperation among them can bring good recommendations.

It is important to mention that the tourism domain was chosen to validate the architecture but we believe that the approach can answer the requests of other applications that deal with problems that require the application of a dynamic and distributed knowledge to solve that application tasks. Testing the reusability of the proposal is going to be done in a next phase of this work.

Decomposing the problem and distribute it to several different agents that are becoming more and more specialized can yield good recommendations, even when applied to tourism that is a complex domain that needs specific knowledge distributed over different sources. The agents can be considered experts, i.e., thanks to the confidence indexes that were added in the agent's model in order to help the agent to select the tasks that it is more prepared to attend.

Another interesting point is that the ideas presented here are being validated in a real scenario. A knowledge acquisition phase was conducted and the agents' knowledge bases were created with knowledge from a real travel agency. The queries used in the experiments were obtained from the travel agency as well, which provide a real scene for collecting and understand the requirements of the application.

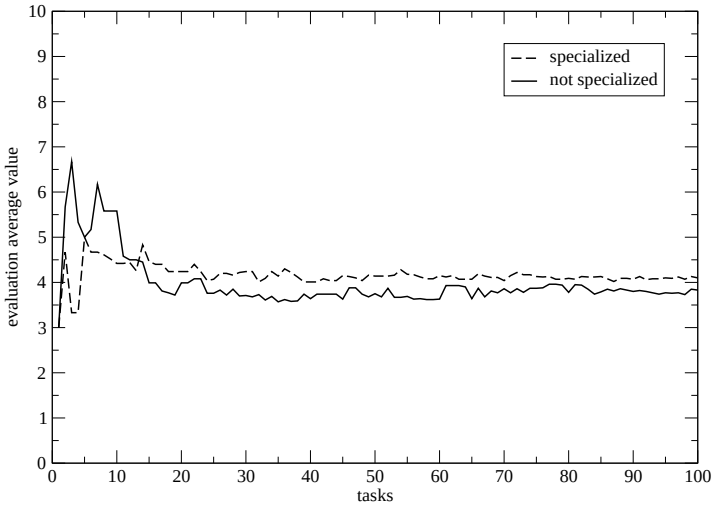

Fig. 7. Average evaluation value - 15 agents (100 tasks)

6 Acknowledgments

This work is supported by the Brazilian Council of Research - CNPq through the founding Universal Program, Process 475597/2006-0.

References

1. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
2. M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *Communications of the Association for Computing Machinery*, 40(3):66–72, July 1997.
3. F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
4. D. Billsus and M. Pazzani. A hybrid user model for news story classification. In *Proceedings of the Seventh International Conference on User Modeling, UM '99*, Banff, Canada, 1999.
5. A. Goy, L. Ardissono, and G. Petrone. Personalization in e-commerce applications. In *The Adaptive Web*, pages 485–520. Springer Berlin / Heidelberg, 2007.
6. F. Lorenzi and F. Ricci. Case-based recommender systems: a unifying view. In B. Mobasher and S. Anand, editors, *Intelligent Techniques for Web Personalization*, pages 89–113. Springer Verlag, 2005.

7. F. Lorenzi, D. S. Santos, D. de Oliveira, and A. L. C. Bazzan. Task allocation in case-based recommender systems: a swarm intelligence approach. In H. Lin, editor, *Architectural Design of Multi-Agent Systems*, pages 268–279. Information Science Reference, 2007.
8. S. Macho, M. Torrens, and B. Faltings. A multi-agent recommender system for planning meetings. In *Workshop on Agent-based recommender systems*, 2000.
9. P. Maes. Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40, 1994.
10. M. Montaner, B. López, and J. L. de la Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19(4):285–330, 2003.
11. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings ACM Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.
12. F. Ricci. Travel recommender systems. *IEEE Intelligent Systems*, 17(6):55–57, 2002.
13. J. Schafer, J. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.
14. B. Smyth. Case-based recommendation. In *The Adaptive Web*, pages 342–376. Springer Berlin / Heidelberg, 2007.
15. K. Sycara. Multiagents systems. *AI Magazine*, 19(2):79–82, 1998.
16. Y. Wei, L. Moreau, and N. Jennings. Recommender systems: A market-based design. In *Proceedings Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS03)*, pages 600–607, Melbourne Australia, July 2003.
17. H. Werthner and F. Ricci. E-commerce and tourism. *Commun. ACM*, 47(12):101–105, 2004.