Assumption-Based Reasoning for Multiagent Case-Based Recommender Systems

Fabiana Lorenzi
Instituto de Informatica, UFRGS
Caixa Postal 15064
Porto Alegre, RS, Brasil
and Universidade Luterana do Brasil
Canoas, RS, Brasil
lorenzi@ulbra.br

Francesco Ricci
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy
fricci@unibz.it

Mara Abel and Ana L. C. Bazzan
Instituto de Informatica, UFRGS
Caixa Postal 15064
Porto Alegre, RS, Brasil
marabel, bazzan@inf.ufrgs.br

Abstract

Recommender systems (RSs) are popular tools dealing with information overload problems in eCommerce Web sites. RSs match user preferences with item representations and recommend the items that better suit these preferences. However, sometimes, the required information may not be fully available, and it could be beneficial to make conjectures about these missing values in order to generate immediately a recommendation even if not optimal. This paper presents an assumption-based multiagent RS making this type of assumptions about the user's preferences. This approach was validated in a travel application analyzing the impact of various assumption making strategies on the quality and efficiency of the recommendation process. The agents are cooperative when solving their tasks, i.e., finding the appropriate travel services (e.g., hotel or flight) to be aggregated in the final recommendation (a complete travel).

Introduction

Several information search tasks are involved in a purchase process on an eCommerce Web site: the search for information about the products, including products' reviews, the search for different brands/suppliers, and the comparison of the identified alternatives. This process requires substantial user effort, hence several information retrieval and data mining techniques have been developed. In particular Recommender Systems (RSs) exploit intelligent techniques that can actively filter irrelevant information or products and can help the users in these processes (Adomavicius and Tuzhilin 2005). But in heterogeneous and dynamic information spaces, such as Internet, the relevant knowledge is typically distributed and one single Web site or repository cannot contain all the relevant data.

Due to this distributed nature of the information, agents have been used for helping users to make decisions (Zambonelli et al. 2001). They are able to jointly collect information from different sources, filter and use that regarded as relevant to the user. Although agents can cooperate and could be able to expand their knowledge communicating among them, during the decision-making process they still can face situations where there is a lack of some useful knowledge. When this happens, they should be able to make intelligent guesses, i.e., assumptions about what they do not know.

This paper proposes an assumption-based multiagent approach that allows agents to independently, collect, assume and exchange information useful to complete their tasks, i.e., helping users in their decision-making process. We show that assumptions can allow agents to improve their performance. Instead of waiting for information coming from other agents, they can immediately start solving their tasks without jeopardizing the system performance (quality and efficiency).

We show that assumptions can help agents to solve problems even when there are dependencies amongst their tasks. We have tested our experimental hypotheses in a tourism scenario: agents must cooperatively build a travel plan for a given user. In this paper we show how the assumption making strategies and the cooperation rules impact on the performance and quality of the recommendations.

In the following sections we present first some related work, and then the main components of our assumption-based multiagent approach and how they were modelled and interrelated. Then we illustrate a case study where we have applied the proposed approach to the tourism domain; we present the experiments and simulations done in order to validate the performance of the process (efficiency) and the quality of the recommendations. Finally we present our conclusions and some future work we have planned.

Related Work

Recommender Systems (RSs) have been proposed to deal with the information overload problems (Resnick et al. 1994; Adomavicius and Tuzhilin 2005), and more in general to support the information selection and decision making processes in eCommerce Web sites. Some RSs exploit multi-agent technologies to make a better use of preexisting knowledge; they build more accurate domain and user models, requiring less training data. For instance, in (Zhang et al. 2008) agents exploit deep knowledge about the customer profile to determine solutions that suit the wishes and needs of a customer. They are able to aggregate information stored in different repositories and better match the recommendations with the user's needs.

More in general, multiagent approaches have been applied to retrieve, filter and use information relevant to the recommendations (Wei et al. 2008). Multiagent RSs exploit a collection of interacting agents jointly executing the recommendation generation process. Each agent may process part of the recommendation and/or the interaction, and the established cooperation among them (e.g., exchanging information) helps in building the final recommendation.

Reasoning with assumptions was introduced by DeKleer with his Assumption-based Truth Maintenance System (ATMS) (de Kleer 1986) and later by (Mason and Johnson 1989), with Distributed ATMS, where agents can communicate and see other agents' assumptions. In DATMS, each agent considers rules that are fired by facts and producing one assumptions set for each fact. DATMS allow agents to make assumptions and work with their consequents until some contradictory information is found.

In (Bogaerts and Leake 2004), the authors investigated strategies for retrieving appropriate cases while handling missing information. In fact, the assumption making techniques proposed in our work are similar to those introduced in (Bogaerts and Leake 2004) for dealing with missing data; but we applied them in a different context, i.e., a distributed recommendation domain.

Finally we want to quote a related line of research, i.e., Distributed Case-Based Reasoning. In these systems, as in our approach, the cases are distributed among the agents and efficiency gains are achieved by distributing the workload (Plaza and McGinty 2005; Ontanon and Plaza 2004).

This paper presents a multiagent RS where cooperative and specialized agents perform their tasks using distributed knowledge (cases), and make assumptions during the recommendation process when some information necessary to complete their tasks is missing. We observe that differently from DATMS, in our approach agents can assume information, independently from each other, during the recommendation process. They can also exchange these assumptions when necessary.

The Assumption-Based Multiagent Approach

In this section we present the assumption-based multiagent approach suited for helping users to find services (items) satisfying their needs. Agents work in a cooperative way to identify recommend items to their users, and are able to make assumptions in the absence of some information.

Agent's model

The agent community $C = \{a_1, a_2, ..., a_n\}$, has the following characteristics:

- **Cooperation**: agents cooperate when performing their tasks in order to generate high quality recommendations;
- **Knowledge**: agents have their own knowledge bases where they store the generated recommendations;
- **Assumptions**: agents can deliberately assume the validity of some missing information when performing their tasks. The assumptions pertain to data, which are needed to complete a task, and in principle should be acquired as the outcome of another agent's task;
- **Truth maintenance**: agents have a truth maintenance reasoner that helps them to manage the potential inconsistencies in their knowledge bases;
- **Specialization**: agents can become expert in specific types of tasks during their activity.

Preferences Model and Tasks

The user preferences are stored in the user model. This is represented by a set of attributes $\mathcal{A} = \{i_1, \ldots, i_m\}$ where each i_m takes values in a collection of possible options $\mathcal{O}_j = \{o_1, o_2, ..., o_{k_j}\}$.

Agents perform several types of tasks $\mathcal{T} = \{t_1, ..., t_n\}$ and each attribute of the preference model i_m is associated with a type of task. The tasks' types may be interdependent, i.e., a task type t may have a set of predecessor tasks, $Pred(t)$ (possibly empty), i.e., the set of task types that should be performed before t.

Agent Specialization

An important feature of our approach is agent specialization, i.e., an agent can become expert in some type(s) of tasks, when its knowledge base of solved problems contains more computed recommendations for these tasks. Agents will prefer to solve the tasks that they are specialized in. The agent specialization is modelled as a confidence index for each type of task as below:

$$confind_a(t) = \frac{F_a(t)}{\sum_{k \in \mathcal{T}} F_a(k)} \times \zeta(t) \tag{1}$$

where $F_a(t)$ is the number of tasks of type t performed by agent a; $\mathcal{T}$ is the set of all task types, and $\zeta(t)$ is the user cumulative evaluation of the recommendations for the tasks of type t. Hence, the confidence index for a task type is calculated as the multiplication of two factors: the proportion of the tasks of that particular type performed by the agent, over the total number of tasks executed, and the user evaluation of the agent performance on that type of tasks. The cumulative evaluation $\zeta(t)$ is simply the number of relevant, i.e., correct recommendations provided to the user. An agent will choose the next task to perform as one belonging to the type with the largest confidence index.

Agent's Knowledge Base

In our approach, each agent has its knowledge base that stores all the recommendations performed by the agent. Each case is composed of a problem description (the set of preferences of the user) and the solution description (the information recommended for that user's query). The agent's knowledge base is stored in XML format. Each task performed by the agent generates a case in its knowledge base with:

- **The user's query**: provides the set of needs and preferences of the user (the problem), and includes those that she cannot compromise, such as the destination city;
- **The recommendation**: the items (services) recommended to the user that represent the solution of the user's query. The recommendation is decomposed according to

the tasks defined in the preference model. In the tourism application the tasks are the flight, the hotel and the attractions;

- *source*: the source of each recommended item. The source can be the same agent that started initially the task or another agent with whom the starting agent communicated to obtain the solution;
- *evaluation*: the user's evaluation for each attribute of the received recommendation.

In additional to the knowledge base, each agent is described by some attributes that are necessary for the recommendation process: the agent ID, the confidence indexes (one for each type of task) and the current task that the agent it is performing (ctask).

In order to perform a task, the agent may search for the information needed in two different ways: LocalSearch or/and CommunitySearch. In LocalSearch, the agent searches for the required information in its knowledge base. If the agent does not find the required information in its knowledge base, then it starts the CommunitySearch, i.e., it communicates with other agents, asking for the information (see the next section).

CommunitySearch

When the agent is searching for some information necessary to perform a task, for example, a cheap hotel in Paris, and it does not find any in its knowledge base, it may ask another agent for this information. When an agent asks for some information (cooperation) it stores the information received and the originating agent in its knowledge base. This behavior is accomplished by creating a *help task*, broadcasting it to the agents in the community, and waiting for the first response. The process of choosing a *help task* to solve is the same used in the regular tasks. Agents, among the broadcasted tasks, choose those that they are more confident with.

Assumptions

In dynamic domains, i.e., where the service features and service availability can change with time, agents could not solve their tasks in an independent way. For, instance if task A depends on task B, then the solution of A should be computed only after the agent knows how B has been solved. However, this behavior introduces potential delays and can jeopardize the agents performance. In many cases, e.g., in online RSs, the time performance of the agents is very important since the user will not accept long response times. Thus, in our model instead of imposing a strict task execution order the agents can make assumptions about the value of some data when they are not immediately available.

Thus, when task t has a predecessor set, the agent a working on t may generate a set of k assumptions given by $\mathcal{S}_a = \{s_{j_1}, \ldots, s_{j_k}\}$, where each s_{j_l} represents a different assumption that the agent can formulate during the recommendation process for the attribute i_{j_l}.

Assumption Making Algorithm The assumption algorithm depends on the agent's knowledge of the user. For instance, if the active user is new to the system and he has no profile yet, it is difficult for the agent to assume some missing information for this user. We have investigated different methods for generating "good" assumptions' sets for new users.

- *Method 1: The most popular option in the community of users*: for new users, the agent assumes as the current value (option) for an attribute the most popular one in other users' cases, contained in the agent's knowledge base;
- *Method 2: Similar cases*: the agent searches for the most similar case in its knowledge base, and for any missing attribute value it uses the options found in this similar case. This method may be applied to new or old users. We have here used also a similarity threshold, that in our experiments was set to 0.5. In order to get this ideal threshold, during the validation stage we performed the same query several times and the expert analyzed the results;
- *Method 3: The most popular option to the user*: for old users, the agent assumes as the current value (option) for an attribute the most popular one in this user's cases, contained in the agent's knowledge base.

We suppose that these assumption making methods can help the agent to complete its task in the recommendation process: in an acceptable time for the user, and without jeopardizing the performance of the system. This proposition will be validated in the next section.

User Evaluation

In a real application of the proposed methods the user, after a recommendation is produced, must be invited to evaluate it. The opinion of the user about the received recommendation is important because that helps validating the effectiveness of each agent in the recommendation process. In this study we have used the evaluation done by an expert (travel agent) as an indicator of the quality of the recommendation.

The expert evaluated the solution by assigning to each attribute in the original query recommended a 0/1 rate, $e(i_j, r_j)$ judging if the output r_j, for attribute j determined by the recommender is acceptable, given the input i_j provided by the user for the same attribute j in the query. The expert rated the attributes individually, i.e., the rating is done attribute by attribute of each travel service.

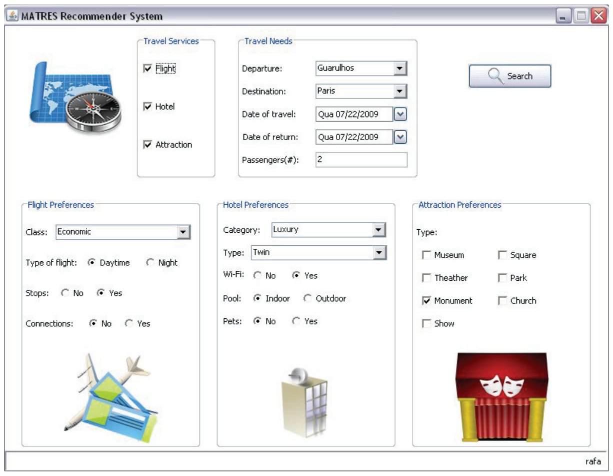
Case Study: Recommending Travel Packages

A case study for a tourism RS was chosen to validate the proposed assumption-based multiagent approach. In this scenario we have developed MATRES (Multiagent Travel Recommender System) and the next sections present its main characteristics.

User Preferences and Tasks

Figure 1 shows the main screen of MATRES. Here the user can enter the travel needs and preferences for each type of task. A task represents a travel service request. We have three types of services: $\mathcal{T} = \{t_1, t_2, t_3\}$, where t_1 represents the flight service, t_2 the accommodation service and t_3 the attraction service.

Considering the example shown in figure 1, assuming that the user has chosen his preferred values for the *class of flight*, *type of flight* and *hotel category* attributes, then the preference model would be $A = \{class\_of\_flight, type\_of\_flight, hotel\_category\}$, for i_1, i_2 and i_3 respectively, and the possible values for these attributes are $\mathcal{O}_1 = \{economic, business, first\}$, $\mathcal{O}_2 = \{daytime, night\}$ and $\mathcal{O}_3 = \{touristic, luxe, first\}$.


MATRES Recommender System

Travel Services
[✓ Flight]
[✓ Hotel]
[✓ Attraction]

Travel Needs
Departure: Guarulhos
Destination: Paris
Date of travel: Qua 07/22/2009
Date of return: Qua 07/22/2009
Passengers(#): 2

Flight Preferences
Class: Economic
Type of flight: [✓ Daytime] Night
Stops: No [✓ Yes]
Connections: [✓ No] Yes

Hotel Preferences
Category: Luxury
Type: Twin
Wi-Fi: No [✓ Yes]
Pool: [✓ Indoor] Outdoor
Pets: [✓ No] Yes

Attraction Preferences
Type:
Museum Square
Theather Park
[✓ Monument] Church
Show

rafa

Figure 1: MATRES system - user's preferences

Assumptions in MATRES

In MATRES the tasks are interdependent; t_1 (flight service) should be finished before t_2 (accommodation service) and t_3 (attraction service) can be completed because both of them are constrained by information contained in the result of t_1. In fact, when an agent searches the best hotel offer for a passenger it should know at what time the passenger will check-in in the hotel, which depends on the arrival time of the flight. Hence, the agent dealing with the hotel recommendation task could hypothesize a "reasonable" flight schedule and complete its task, assuming the *arrival time* and *arrival date*. This flight related assumptions also help the agents to perform the attraction tasks, where it is necessary to know when the customer will arrive in the city in order to schedule the possible tour visits.

Knowledge Base and the Retrieval Algorithm

Table 1 shows an example of a case stored in one agent's knowledge base. Each task performed by an agent generates a new case (a XML file). The set of XML files composes the agent's knowledge base. The user query contains the set of needs and preferences of the user. The **travel needs** are strict requirements, which the system must satisfy, and are used as filters during the search process. They are the destination, the travel date, the return date and the number of passengers. Conversely, the **preferences of the user** represent additional wishes of the user, such as the hotel category or the flight class. The preferences, differently from the needs, are negotiable. Hence, if the user prefers night flight than day flight it is still possible to violate this requirement.

Table 1 shows an example of user query and how it is stored as a case. The recommendation has not been determined yet. The items **flightR**, **accommodationR** and **attractionR** elements will be filled as soon as the agents complete the corresponding tasks. The evaluations of each task attribute will be also stored after the user evaluates them.

Table 1: Example of a case

< travelneeds >	**<attraction>**
<departure> Rome </departure>	<type>Museum</type>
<destination> Paris </destination>	**</attraction>**
<departuredate> 01/01/2010 </departuredate>	**< /Preferences >**
<returndate> 01/10/2010 </returndate>	**<recommendation>**
<passangers> 2 </passangers>	**<flightR>**
< /travelneeds >	**</flightR>**
< Preferences >	**<accommodationR>**
< flight >	**</accommodationR>**
<class> Economic </class>	**<attractionR>**
<typeflight> Daytime </typeflight>	**</attractionR>**
<stops> No </stops>	**</recommendation>**
<connections> No </connections>	**<evaluation>**
</flight>	**<flight>**
<hotel>	**</flight>**
<category> Touristic </category>	**<hotel>**
<room> Double </room>	**</hotel>**
<wifi> Yes </wifi>	**<attraction>**
<pool> Indoor </pool>	**</attraction>**
<pets> No </pets>	**</evaluation>**
</hotel>	

When an agent starts a task, it initially searches in its knowledge base and selects the cases that match the user needs (destination, travel date, return date and number of passengers). Then, it tries to match the user's preferences in this set of cases. Since the knowledge is stored as a set of cases, the goal is to search there for the most similar case and to apply the solution (recommendation) of this old case to the new user's query. The agent should compare the user's query with all the cases stored in its knowledge base, i.e, compare the user's query with all the recommendations that the agent has performed in the past. This comparison is done using the distance function shown in equation 2.

$$Dist(q, c) = \frac{\sum_{j=1}^{k} dis(q_j, c_j)}{k} \tag{2}$$

where q is the user's query, c is the case that is being compared, j is an attribute, k is the number of attributes and $dis(q_j, c_j)$ is the local distance. We observe that all the attributes are symbolic, and the local distance is 0 when $q_f = c_f$ and 1 otherwise.

Experimental Results

A community of 10 agents was created to run the experiments. Then, 30 cases were acquired from a travel agency and distributed randomly in the agents' knowledge bases. This distribution was not uniform, i.e., some agents received more cases than others. This was done to mimic what could happen fo real travel agents, where some of them may have solved more cases than others. However, their confidence indexes were set to a constant value (0.5), i.e., we assume that initially they have performed the same number of tasks and they were evaluated similarly. Thus, no agent is considered an expert in the initial stage. These 30 cases were generated by four human travel agents. They were complete travels, with flights, hotel and attraction travel services, and the travel agents also provided the passengers evaluations for each case.

Besides the cases that populated the agents knowledge bases, the travel agents provided 27 additional recently solved cases (we call them "test cases") to be used as queries in the simulations. These cases also were completed with the evaluations provided by the travel agents. This is important because for validating our multiagent RS we could compare the system results (recommendations) with the experts' results. The advantage of using real cases as new queries is that we can measure the output of our approach on realistic problems and we can use the solutions as correct evaluations of the recommendations.

Simulations

To evaluate our approach we ran some simulations where the agents solved some queries that were also solved by the travel agents, so that we could compare the two solutions. The simulations were carried out in two steps: 1) we have defined three different recommendation scenarios to evaluate how the assumption making feature impacts on the recommendation performance; 2) we have run new queries (selected from the 27 test cases) with two different assumption making methods and then compared their results with the solution in the test cases.

The three scenarios defined are:

- **Scenario 1**: the agents perform their tasks independently from the other tasks, ignoring the dependencies among tasks and without making assumptions;
- **Scenario 2**: when an agent must perform a task that depends on another task it waits for the information produced by the preceding task solution;
- **Scenario 3**: when an agent must perform a task that depends on another task it will make the most likely assumptions on the solutions of the preceding tasks instead of waiting for the true information.

We observe that since these recommendation techniques must be included in an eCommerce Web site the waiting time and the quality of the recommendations must be taken into account. In fact, users do not like to wait too long for an answer, so the system should return an acceptable solution in a short time. Moreover, users want to receive reasonably good recommendations and they will not like suggestions that differentiate considerably from their requirements. Therefore, in the simulations we analyzed the results with respect to the performance of the system (efficiency) and the quality of the recommendations, as it is defined later.

Performance In the first simulation, we considered the time required by the agents to complete the tasks in the three defined scenarios. Table 2 shows the average time required for processing the 27 queries. The simulation was run in a "Core2Duo" - 1.83GHz computer, with 2GB RAM.

Table 2: Average time of processing 27 queries (in minutes)

Queries	**Scenario 1**	**Scenario 2**	**Scenario 3** (method 2)
1-27	1.37	2.32	1.54

As table 2 shows, when the agents solve the tasks independently (*scenario 1*), or when they do not wait for the outcomes from other agents, i.e., they make assumptions (*scenario 3*), then they can complete their own task quicker. This is not surprising because an agent searches only locally for the information in its knowledge base and this does not require a major effort.

As expected, in *scenario 2* the time required to generated a full travel plan increases when agents wait for information coming from other agents. In *scenario 3* the agents had on average better performance than in *scenario 2* because they made some assumptions instead of waiting for information from other agents. In *scenario 3* we choose *method 2* to make assumptions (see Section).

Quality of the Recommendations We also verified the effectiveness of the agents, i.e., the quality of the recommendations presented to the users, using *method 1* and *method 2* for making assumptions. In this simulation, from the original 27 test cases, we selected the 20 queries from new users. The quality of the recommendations obtained in the scenarios were then compared with those generated by a human expert (travel agent). Table 3 shows some examples of the used queries.

Table 3: Examples of cases used as new queries in the simulations

Cases	Destination	Paxs	Class	Type	Categ.	Room	Attraction
1	Lisbon	2	econ.	day	tourist	double	monument
2	Paris	2	econ.	night	tourist	double	museum
3	Madrid	2	econ.	night	luxe	double	show

Table 4 shows the average quality of the recommendations for each travel service and for each scenario defined. The higher the value, the better the result. In the evaluation each input attribute in the query was compared with the final value of the attribute in the agents' and expert's solutions. We note that the expert solution correspond to what the traveller bought.

Moreover, note that the maximum evaluation corresponds to the maximum number of possible input preferences (attributes) selected by the user: (*flight*=4, *hotel*=5 and *attraction*=1, as we considered just one attraction option chosen by the user in the query.

As we expected, when the agents do not consider the dependencies among the travel services (*scenario 1*), they had the worst results. In *scenario 2* the results are a little better than the expert solution. However, we expected better results, considering that agents shared information.

Table 4: Effectiveness of recommendations for each travel service in Scenario 3

	Average Rating		
	Flight	Hotel	Attraction
Expert Solution	3.2	3.9	0.8
Agents (Scenario 1)	3.1	3.2	0.7
Agents (Scenario 2)	3.6	4.0	0.8
Method 1 (Scenario 3)	3.4	4.2	0.8
Method 2 (Scenario 3)	3.5	4.7	0.9

The average users' evaluation for the hotel services produced by *method 2*, which is an assumption-based method, is 4.7. This is a good result considering that the best possible result is 5, i.e., when all the 5 input preferences in all the queries are satisfied in the solutions. In fact, the produced recommendations were presented to the human travel agent who evaluated them as excellent and liked the idea of using similar cases.

We can see that *method 2*, i.e., using similar cases, produces better results for the Hotel tasks, compared with *method 1*, i.e., using the most popular value for a missing information. *method 2* is also better than the expert solution. *method 2* is significantly better than the expert and *method 1* ; t-test probabilities are p=0.002, and p=0.0003 respectively. It means that using similar cases is a good strategy to make assumptions when necessary.

The performance of the two assumption-based methods on the Attraction and Flight tasks were not very different from those obtained by the expert. In the Attractions this also depends on the fact that in this prototype just one attraction is recommended. Anyway, it is not surprising that on these two types of tasks the performances are very similar, since in these cases no assumptions are made as their solutions are not dependent from other tasks (as it was for the hotel task).

An important issue identified in these simulations is that the assumption-based multiagent approach is able to balance *performance* and *quality*. In fact, it is not important for a system to produce quickly its recommendations if their quality is poor. Users want high performance but acceptable quality as well. Thus we cannot compromise too much the quality by optimizing the performance. The current solution seems to achieve a good balance, in this particular simple combination of tasks.

Conclusions and Future Work

This paper presented an assumption-based multiagent RS that allows agents to cooperate in solving their tasks, and let them make assumptions regarding missing information during the problem-solving process. We have shown that these assumptions can help to improve the quality of the recommendations compared to the case where agents do not wait for the missing data to be available and do not make assumptions. In this last case the solution is found quicker but it is often incorrect. Moreover we have shown that the agents that make assumptions can still obtain good quality results, when they have to deal with interdependent tasks. A case study in the tourism domain has been presented and we validated the proposed approach in a real application. We have introduced two methods for generating assumptions, one based on the most often observed value for an attribute and another based on the examination of similar recommendation cases. In our case study, both approaches worked well.

As future work we intend to develop a mechanism that helps in identifying when agents make wrong assumptions. This mechanism will be helpful to correct the preference model. We also want to extend the assumption mechanism and validate results when the agents share their assumptions. We imagine that this would improve the agents performance because they would be able to use other agents assumptions rather than compute them by themselves.

Moreover, a trust mechanism is being developed in order to let the agents to select their tasks and also to improve the communication among agents. Agents will use a trust degree to select a task and instead of using *help tasks* to exchange information, agents will communicate only with agents they trust. This will avoid unnecessary communication and thus improve the recommendation process.

This work was challenged by the features of the tourism domain. The prototype was simplified and a restricted number of assumptions were explored. However, we intend to consider other applications where a larger set of assumptions and dependencies must be managed.

References

Adomavicius, G., and Tuzhilin, A. 2005. 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.

Bogaerts, S., and Leake, D. 2004. Facilitating cbr for incompletely-described cases: Distance metrics for partial problem descriptions. In Funk, P., and Calero, P. A. G., eds., *Proceedings of the 7th European Conference on Case Based Reasoning, Advances in Case-Based Reasoning*, Lecture Notes in Artificial Intelligence, 62–76.

de Kleer, J. 1986. An assumption-based tms, extending the atms, and problem solving with the atms. *Artificial Intelligence* 28(2):127–224.

Mason, C. L., and Johnson, R. R. 1989. Datms: A framework for distributed assumption based reasoning. *Distributed Artificial Intelligence* 2:293–317.

Ontanon, S., and Plaza, E. 2004. Recycling data for multiagent learning. In de Raed, L., and Wrobel, S., eds., *Proceedings 22nd International Conference on Machine Learning ICML-2005*, ACM Press, 633–640.

Plaza, E., and McGinty, L. 2005. Distributed case-based reasoning. *The Knowledge Engineering Review* 20(3):261–265.

Resnick, P.; Iacovou, N.; Suchak, M.; Bergstrom, P.; and Riedl, J. 1994. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings ACM Conference on Computer-Supported Cooperative Work*, 175–186.

Wei, Y. Z.; Jennings, N. R.; Moreau, L.; and Hall, W. 2008. User evaluation of a market-based recommender system. *Autonomous Agents and Multi-Agent Systems* 17(2):251–269.

Zambonelli, F.; Jennings, N. R.; Omicini, A.; and Wooldridge, M. 2001. Agent-oriented software engineering for internet applications. In Omicini, A.; Zambonelli, F.; Klusch, M.; and Tolksdorf, R., eds., *Coordination of Internet Agents*, 326–346. Springer-Verlag.

Zhang, D.; Simoff, S.; Aciar, S.; and Debenham, J. 2008. A multi agent recommender system that utilises consumer reviews in its recommendations. *International Journal of Intelligent Information and Database Systems* 2(1):69–81.