Improving recommendations through an assumption-based multiagent approach: An application in the tourism domain

Fabiana Lorenzi [a,b,*], Ana L.C. Bazzan [a], Mara Abel [a], Francesco Ricci [c]

[a] PPGC–Instituto de Informática, UFRGS, Caixa Postal 15064, 91.501-970 Porto Alegre, RS, Brazil
[b] Universidade Luterana do Brasil, Av. Farroupilha, 8001 Canoas, RS, Brazil
[c] Free University of Bozen-Bolzano, Italy

ARTICLE INFO

ABSTRACT

Recommender systems are popular tools dealing with the information overload problem in e-commerce web sites. The more they know about the users, the better recommendations they can provide. However, sometimes, in real situations, it is necessary to make guesses about the value of missing but useful data in order to generate a recommendation immediately, rather than waiting the data becomes available. This paper presents an assumption-based multiagent recommender system capable of making these types of assumptions about the preferences of the users. The approach was validate in the tourism domain (recommendation of travel packages). Experiments were conducted to illustrate the impact of various assumption making strategies on the quality of the recommendations as well as the impact of trust assignment.

1. Introduction

Not long ago, the traditional way of choosing a travel destination was to look for catalogs and advertisements about specific destinations, or going to a travel agency to get the advice of some experienced travel agent. With the Internet and the huge increase of available options for traveling, advising a travel package with updated information has become a challenge for the travel agent her/himself. Internet has made available a huge amount of information about places to visit, transportation and accommodation, either in public repositories or in commercial sites of tourist attractions. Search engines reduce the users' burden by automating this process.

Although some persons still search in catalogs, this is rare. Usually, information about destinations comes from the Internet. Web search engines crawl the web to index pages and allow users to have prompt answers for a given query. A typical way of using a search engine to define travel options is to type the name of the place to visit, then go to another search service and look for options of flights, and then to another service to search for hotel options, etc.

Even though the travel agent would be able to find the required information on the web, s/he needs to deal with outdated, some-times conflicting information, before building an integrated solution that fits all the requests and preferences of the traveler.

E-commerce web sites (such as those used for buying train or flight tickets) allow users to make parametric searches. However, this suffers from the same problems of search engines. The user has to know the time and destination in advance and solve the conflicts produced by wrongly published information. This has led travel agents to deal with the information inconsistencies or overload.

Because the number of possible travel options and sources of available information is growing at a staggering rate, it has become more difficult to distinguish where the updated and correct information is. The required information for choosing the best option for a travel package is mostly distributed, i.e., one single web site or repository does not contain all the relevant knowledge about places, flights and accommodations. Moreover, sometimes a travel agent does not have all the information necessary to compose a recommendation and s/he has to communicate with other travel agents. We summarize the characteristics of the tourist domain as follows:

- The relevant information to support the decision is distributed.
- The information sources are specialized.
- The information is updated frequently and in an asynchronous way.
- The query is done in real time and the result needs to be efficient.

Recommender systems (RSs) are being used to recommend items among a huge stream of possibilities and according to users'

* Corresponding author at: PPGC–Instituto de Informática, UFRGS, Caixa Postal 15064, 91.501-970 Porto Alegre, RS, Brazil. Fax: +55 51 34765544.
E-mail addresses: fabilorenzi@gmail.com (F. Lorenzi), bazzan@inf.ufrgs.br (A.L.C. Bazzan), marabel@inf.ufrgs.br (M. Abel), fricci@unibz.it (F. Ricci).

interests. They have also been proposed to support the information selection and decision making processes on e-commerce web sites (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994). Usually, these systems can fully support the purchase (Goy, Ardissono, & Petrone, 2007). In general, RSs use a range of different approaches; the most popular are: collaborative filtering, content-based filtering and knowledge-based algorithms (Adomavicius & Tuzhilin, 2005).

Collaborative filtering algorithms work by recommending items to people based on what people with similar profiles have liked. Content-based filtering algorithms recommend items to a user if these items are similar to items the user has liked in the past. Knowledge-based algorithms apply preexisting knowledge about clients or products to build a more accurate recommendation (Wei, Moreau, & Jennings, 2003).

Agents can exploit deep knowledge about a customer's profile in order to determine solutions that suit his/her wishes and needs. Such agents are able to aggregate information and match the recommendations with the information that user is looking for. In addition, an agent-based recommender system can be built with a centralized architecture or a multiagent architecture. A multiagent system allows distributing the recommendation task among separate modules of software, which improve the efficiency and performance of the system.

In this paper our goal is to address the problem of having distributed information related to the tourism domain (transportation, accommodation, etc.) by using a multiagent recommender system on the task of finding, filtering, and integrating relevant information to build a travel package according to the user request.

Our previous research (Lorenzi, Bazzan, & Abel, 2009) showed how a community of agents deals with conflicting information by progressively specializing the data of each local agent knowledge base to produce a recommendation. Here we introduce a system, called MATRES (Multiagent Travel Recommender System), that integrates these approaches to recommend travel packages based on distributed, frequently updated information matched against the user requests. This research discusses a multiagent recommender system that integrates the local knowledge of several agents and deals with lack of information by mean of using assumptions. This help users dealing with information overload, guiding them to build a suitable recommendation. We believe that the proposed approach can be a significant improvement to currently available recommendation systems.

This paper is organized as follows: the next section reviews the related work for recommender systems. Section 3 presents the main components of our approach (MATRES). Section 4 shows the experiments performed in order to validate the effectiveness of the proposed approach in the recommendation process. Section 5 presents the conclusions and future work.

2. Related work

Multiagent Recommender Systems (MRSs) introduce a collection of interacting agents that execute the recommendation generation process and they have been applied to retrieve, filter and use information relevant to the requested recommendations (Macho, Torrens, & Faltings, 2000; Wei et al., 2003). Each agent may process a part of the recommendation and the interaction and cooperation among them (e.g., the exchange of information) helps in building the final recommendation.

Multiagent recommender systems have been applied in different domains such as legal (Drumond, Girardi, & Leite, 2007), marketplace (Wei, Jennings, Moreau, & Hall, 2008), e-commerce (Zhang, Simoff, Aciar, & Debenham, 2008), and tourism (Macho et al., 2000). In Drumond et al. (2007), for instance, authors present a multiagent recommender system for the legal domain. The system classifies legal normative instruments into legal branches. Each user specifies his/her interests for certain legal branches and receives recommendations of instruments they might be interested in.

In Wei et al. (2008) the authors investigated the feasibility of building a recommender system as a marketplace in which the various recommender agents compete to get their recommendations displayed to users. The market works as a coordinator of the multiple recommendation techniques in the system. The main idea of the system is to improve the recommendation quality choosing the best recommendation method to the current situation. The market correlates the agents' internal evaluation and the user's evaluation of the recommendations by invoking a bidding process and a rewarding regime. Auctions are used to decide the recommendation winners and the agents who provided good recommendations (chosen by the user) receive some reward in return. Agents that receive more reward become richer and are able to get their recommendations advertised more frequently. This behavior has the following disadvantage: it prevents the serendipity. Thus, potentially, the system will never be able to surprise the user with unexpected recommendations.

Another multiagent recommender system is presented in Zhang et al. (2008) and deals with reviews for digital cameras. This system automatically collects relevant comments from product reviews and generates recommendation. Three different types of agents were created: web mining, recommendation, and interface agents. Web mining agents are responsible for retrieving review comments from predefined web sites. Recommendation agents convert the comments from free text form to a structured ontology and calculate the rating of each product from the review comment ontology and formulate specific recommendations based on the rating calculations. User interface agents allow users to connect to the system via the Internet. This system calculates the product rating by the consumer skill level and product quality ranking rather than using a recommender algorithm such as content-based or knowledge base. There is no information about how new products can be included in the recommender systems and what happens if some of these products are not included in the ontology.

MAPWEB (Camacho, Molina, Borrajo, & Aler, 2002) plans travels according to the preferences of the user. It includes the following agents: the UserAgent, which is responsible for the communication between the user and the system; the PlannerAgent that is responsible for planning the travel; the Webbot that is responsible for searching information in Internet; and the CoachAgent that acts like a coach for the group of agents, controlling them and assigning tasks to them. MAPWEB agents store plans as cases and use these cases to build new plans. During this process the agents communicate to help each other in completing their own tasks.

The CoachAgent agent does not match the distributed philosophy of MAPWEB because the control and management of the coach is centralized. A CoachAgent agent controls the communications between agents and indicates who must help whom. Moreover, there is no established process to validate the knowledge of the agents. Agents may work with outdated information during the planning process causing the generation of bad recommendations.

SmartClient (Torrens, Faltings, & Pu, 2002) is another multiagent system applied to the tourism domain that helps the user to plan a flight by representing the space of solutions (recommendations) as a constraint satisfaction problem. Users insert the departure city, the cities they want to visit and the travel dates. With these preferences in hand, the system builds a constraint satisfaction network able to exploit the possible routes. This approach has some drawbacks. First, in order to avoid several costly accesses, it collects route information from the server only once. That limits the search space because it is not possible for an user to modify her preferences and to refine the query. Secondly, the domain of the

variables are fixed by the system and it is not possible to explore solutions that are not originally in the search space.

MAPWEB and SmartClient differ from our approach in two aspects. In MATRES, an agent is fully autonomous and it can decide which task it will perform according to the specific knowledge it has. Secondly, in the absence of some information necessary to complete it task an agent makes assumptions to build the recommendation.

The present work describes a multiagent recommender approach that allows agents to use assumptions in order to recommend travel packages. It is inspired by the Assumption-based Truth Maintenance System (ATMS) presented by deKleer (1986). In our case, assumptions allow agents to achieve better performance when there are delays in receiving information from other agents. In fact, if an agent must wait for an information, then there is a negative impact on the performance of the recommender system, which will ultimately cause a negative perception of the system behavior on the user who can quit the system.

To guarantee the correct behavior of the recommender system, the agent dealing with the hotel recommendation, for instance, must be able to complete its work without waiting for the information from another agent to be available. For example, in the absence of the flight information, the agent may assume that the user will arrive at 1 p.m. and then search for the best hotel offer to recommend.

In deKleer (1986), an ATMS maintains independent chains of reasoning based on assumptions and adjusts current beliefs as new information is received. Given a new information, it analyzes the existing set of assumptions and returns *true* if any contradiction has arisen. Our approach of using assumptions differs from deKleer (1986) and Mason and Johnson (1989) in the sense that the set of assumptions is computed on demand by the agent. As tourism is a dynamic domain it is important to have an approach that deals with this feature. We propose to make assumptions when necessary rather than keeping a set with all assumptions and their relationship, which are reviewed when some new information arrives. Agents are able to create assumptions over the cases stored in their knowledge bases during the recommendation process according to the their need.

3. MATRES: a Multiagent Travel Recommender System

In order to deal with the requirements of the travel recommendation system, we have proposed a system architecture based on multiagent components in which the agents are able to specialize itself in different kinds of tasks and incorporate features to deal both with lacking and inconsistent information. The MATRES application was developed to run in the work environment of a travel agency, being the user both, the staff of the agency, or a client who wants to simulate a travel package to acquire.

As a multiagent system, the MATRES aggregates important skills that define the quality of recommendation:

- the system is able to receive a request for travel recommendation from the user and decompose it in several tasks;
- agents are able to perform tasks in order to cooperate for presenting the final recommendation to the user;
- agents have a mechanism that exploits trust management to decide which task it is better to perform;
- agents are able to search for information needed in their own knowledge bases;
- agents are able to use assumptions when necessary information for the recommendation process is missing;
- agents are able to exchange information with other agents when necessary.

These characteristics of the system lead it to a better performance in terms of recommendation quality as we will show in the validation section. Here we present how MATRES was developed as well as we describe the way in which the recommendation cycle was structured, its main architecture components, the
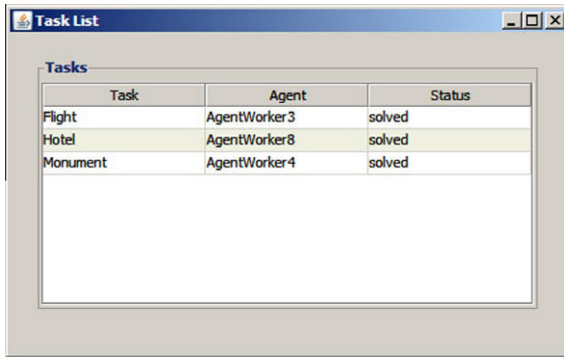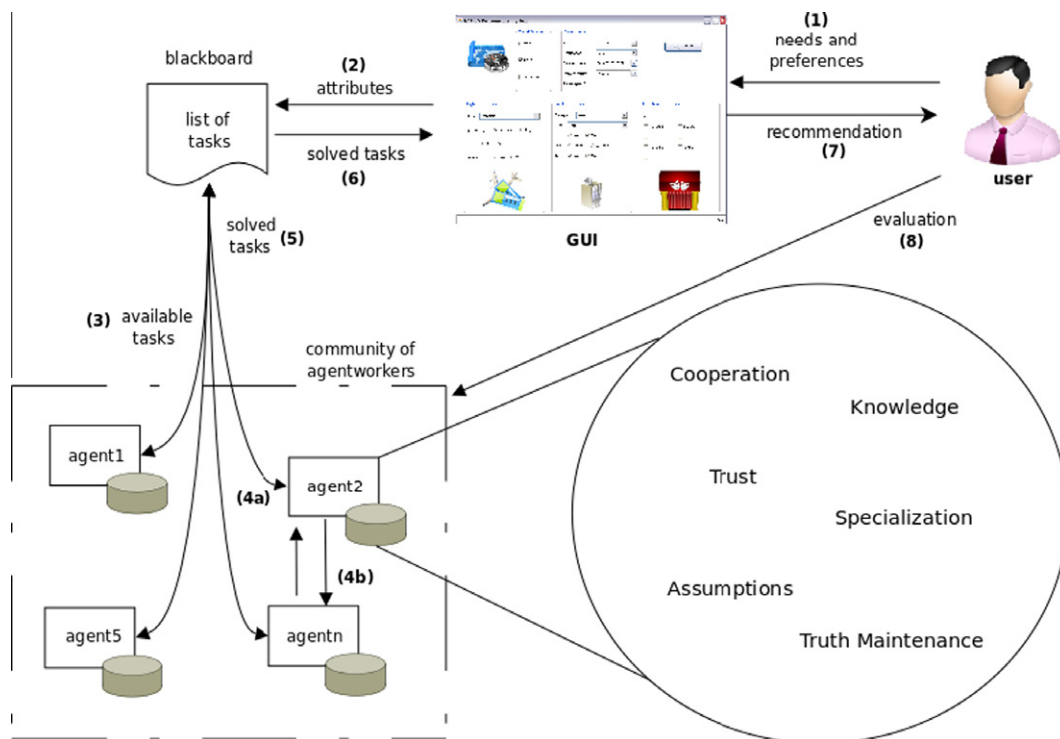

Fig. 1. MATRES architecture.

Fig. 2. Example of the list of tasks *L*.

knowledge model of the agents and the algorithm that matches the user desires and travel conditions.

A cycle of recommendation starts when the user inputs a new request for a travel. As shown in Fig. 1, one cycle of recommendation has eight steps in MATRES. It starts with collecting the user's preferences (step 1) and extracting attributes from these preferences (**step 2**) in order to create the tasks, i.e., divide the request in small pieces (**step 3**) that will be performed by the agents. Each agent chooses one task to perform and it searches for the information necessary to perform the chosen task. This search process may be executed in two different ways: local (**step 4a**), when agents search in their knowledge base, or in the community (**step 4b**),
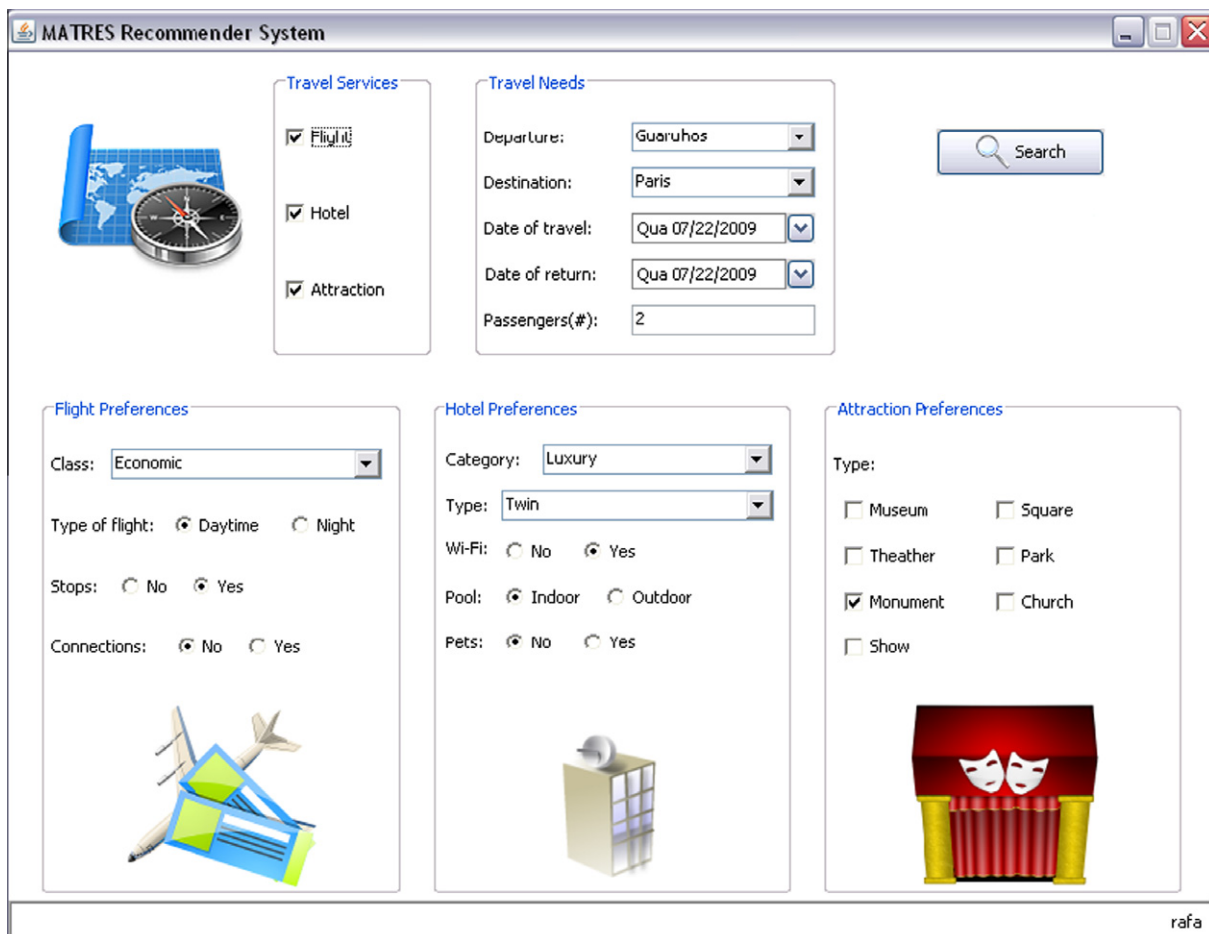
when agents cooperate, communicating to each other to get the information. After solving the task, agents change the status of the performed task (**step 5**), the result of the task is returned to the interface (**step 6**) and the system presents the whole recommendation to the user (**step 7**). To complete the cycle of recommendation, the user evaluates the recommendation received (**step 8**) and this evaluation is stored in the agent knowledge bases.

3.1. Tasks

A special characteristic of multiagent recommender systems is that the user request may be divided in small pieces to be performed by different agents. These pieces represent different tasks needed to build the final recommendation. In travel package recommendations, the user's request may be decomposed in transportation and accommodation travel services. One agent would perform the task of flight finding and other agent would perform the task of hotel finding. This allows them to perform their tasks in an asynchronous way.

In MATRES, each type of travel service is considered a task and the set of types of tasks is given by $\mathcal{K} = \{t_1, t_2, t_3\}$ where t_1 is flight service, t_2 is accommodation service, and t_3 is attraction service.

The user requests a recommendation through the main graphic user interface (shown in Fig. 3), choosing which travel service s/he wants in the recommendation. Also this user sets preferred values for the respective attributes. Once this is done, a list *L* of task is created from these attributes. Thus, *L* represents the user preferences in the current cycle of recommendation.
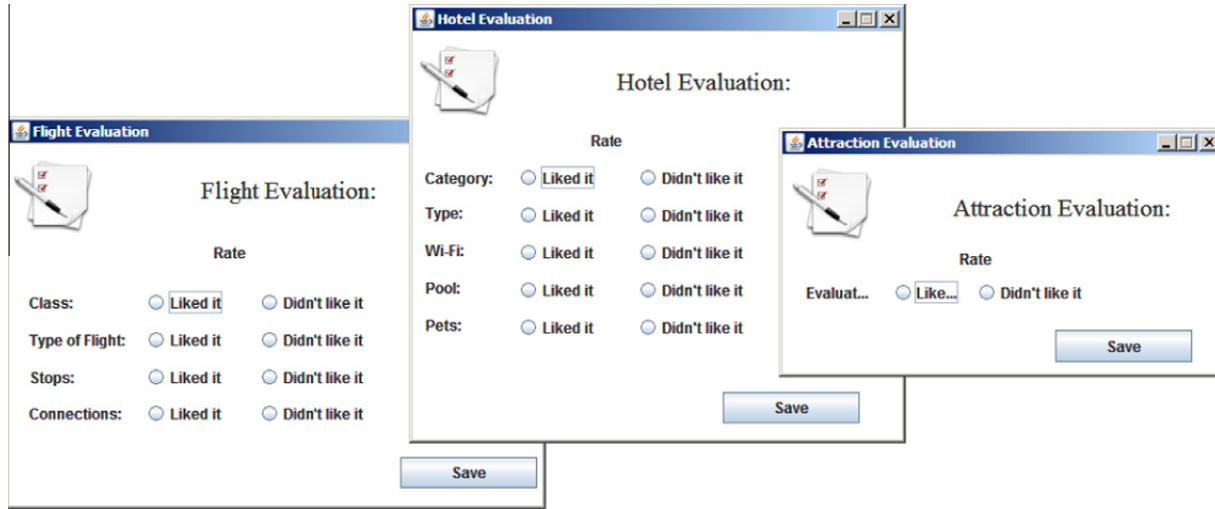

Fig. 3. MATRES main interface.

Fig. 4. Evaluations of the recommendations received.

L is represented as a blackboard model, where each agent is able to see available tasks and choose one to perform. This normally involves searching for the information required in order to make the recommendation. After solving the task, an agent changes the status of the task, partially contributing to the recommendation.

Fig. 2 shows an example of *L* in a cycle of recommendation. It contains the task, the agent who picked the task and its current status. The types of tasks may be interdependent, i.e., a type of task t_n may have a set of predecessor tasks. This set of predecessor is given by $Pred(t)$ and it indicates which type of task should be performed first.

In MATRES, there are some types of tasks that are interdependent. For example, t_1 (flight service) should be finished before t_2 (accommodation service) and t_3 (attraction service) can be accomplished because both of them need information from t_1. It means that $Pred(t_1) = \emptyset$, $Pred(t_2) = Pred(t_3) = \{t_1\}$.

3.2. User model

The preferences of the user are the most important source of information for a recommender system. These preferences are used to create the user's model and ultimately for searching the most suitable recommendations. In MATRES, each t_n has a set of preferences represented by $(U_{t_n}) = \{i_1, \ldots, i_m\}$, where each i_m represents the option chosen by the user in the attribute m. i_m may be a value from a collection of possible options $\mathcal{O}_{m_j} = \{o_1, o_2, \ldots, o_{m_j}\}$. The set of all U_{t_n} represents the user model. For example, assuming that the user has chosen the preferred values for the attributes related to *class of flight*, *type of flight*, and *hotel category*, then the sets of preferences would be $U_1 = \{class\_of\_flight, type\_of\_flight\}$, for i_1 and i_2, respectively, where $\mathcal{O}_{1_1} = \{economic, business, first\}$, $\mathcal{O}_{1_2} = \{daytime, night\}$, and $U_2 = \{hotel\_category\}$ for i_1, where $\mathcal{O}_{2_1} = \{touristic, luxe, first\}$.

3.3. Evaluation of the recommendation

At the end of each cycle of recommendation, the user is invited to evaluate the recommendations obtained from the system, as shown in Fig. 4. The opinion of the user about the received recommendation is important because it helps to validate the effectiveness of each agent in the recommendation process. MATRES uses the evaluation done by the user as an indicator of the quality of the recommendation.

The evaluation model is represented by $\overrightarrow{E_{t_n}} = \langle e(i_1, r_1), \ldots, e(i_m, r_m) \rangle$, where $e(i_j, r_j) = 1$ if the user liked the recommendation or $e(i_j, r_j) = 0$ otherwise.

As shown in Fig. 4, the user rates the attributes individually in each travel service, i.e., the rating is done attribute by attribute for each type of task. For each type of task t_n, the final evaluation rate may be a value in the range $[0\ldots1]$. Eq. (1) shows how the final evaluation rate is calculated for each travel service t_n, where m is the number of attributes for the travel service t_n, and $e(i_j, r_j)$ is the evaluation rate for the attribute i_j.

In MATRES, a "good recommendation" means $v_{(t_n)} \geqslant 0.5$, i.e., that the user has rated as "I liked it" at least for half of the attributes

$$v_{(t_n)} = \frac{\sum_{j=1}^{m} e(i_j, r_j)}{m} \tag{1}$$

After the user has evaluated all items, a XML file is created with the evaluations of each travel service, as shown in Fig. 5.

3.4. AgentWorkers

MATRES consists of a set of *AgentWorkers* distributed in a community of agents defined by $C = \{a_1, a_2, \ldots, a_i\}$. An *AgentWorker* is able to choose a task from the list L in each cycle of recommendation, perform it and produce part of the recommendation in response to the user query. When an *AgentWorker* does not have information necessary to complete the chosen task, they may

```
<object.FlightEvaluationXML>
  <flightE>
    <fClass>true</fClass>
    <typeFlight>false</typeFlight>
    <stops>false</stops>
    <connections>false</connections>
  </flightE>
</object.FlightEvaluationXML>
```

```
<object.HotelEvaluationXML>
  <hotelE>
    <category>true</category>
    <type>true</type>
    <wifi>true</wifi>
    <pool>false</pool>
    <pets>false</pets>
  </hotelE>
</object.HotelEvaluationXML>
```

```
<object.AttractionEvaluationXML>
  <attractionE>
    <like>true</like>
  </attractionE>
</object.AttractionEvaluationXML>
```
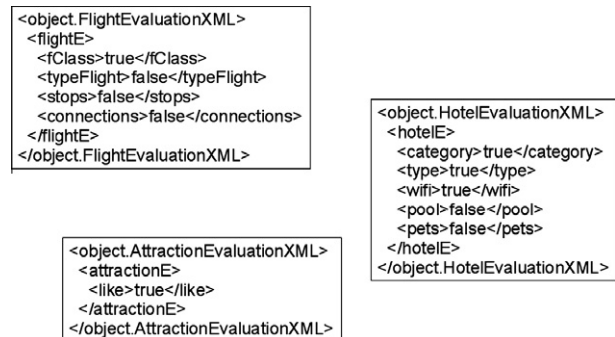
Fig. 5. XML files of evaluations for each travel service.

cooperate. This cooperation occurs by means of communication, i.e., exchange of information that is necessary to complete a task.

Each *AgentWorker* a_i has the following characteristics:

- **Knowledge**: an agent has its knowledge base where it stores all recommendations performed. The knowledge is stored as cases.
- **Assumptions**: an agent is capable of making assumptions about some information in order to perform a task. These assumptions are related to the data that the agent needs during the recommendation process but is not available when the agent needs it.
- **Truth maintenance**: each agent has a truth maintenance component that is responsible for checking the knowledge base looking for inconsistencies. An inconsistency happens when two agents have the same information stored in their knowledge base with different status.
- **Specialization**: when an agent performs more tasks of a specific type t_n, it will have more information of this type in its knowledge base. This means that the more tasks of type t_n that an agent solves, probably the better are the evaluations coming from users; thus this agent will become an expert regarding the type of task t_n.
- **Trust**: agents have a mechanism of trust where they keep a trust record about other agents.

3.5. Agent specialization and trust

An important feature of MATRES is the agent specialization that allows it to become expert in some type of task. This specialization helps agents to decide which task they should perform during the recommendation process, as they will prefer to solve the tasks in which they are specialized in order to get a better recommendation.

The agent specialization is modeled in MATRES by means of a trust mechanism. An agent can be aware of both, its capacity of doing a given type of task (its *confidence degree* for a type of task t_n), and the *trust* it has in the other agents that have cooperated with it to perform a task. The confident degree helps the agent when choosing a task to accomplish, while trust degree helps the agent when choosing another agent to exchange information necessary for the accomplishment of a task. The goal is to make the recommendation process more efficient, for example by avoiding unnecessary communication, exchanging information just with agents it trusts.

In MATRES, the trust of agents is based on context. The trust of each agent on the other agents (as well as the confidence in itself) depends on the type of task. Agent a_1 may trust agent a_2 regarding task t_1 but may not trust a_2 regarding task t_2.

Due to the dynamic and distributed nature of MATRES scenarios, we defined that each agent has a XML file with the trust degree in other agents and the confident degree in itself. Fig. 6 shows agent a_2's XML file, which shows level of trust in other agents regarding a flight service. We can see that the trust value of a_2 on a_1 is 0.9375 and the trust value of a_2 on a_{10} is 0.5. The agent stores also information about the number of positive, negative and neutral evaluations (*intpos*, *intneg* and *intind*, respectively).

When agent a_i receives an evaluation from the user concerning a type of task t_n there are two possible situations:

- a_i did not need help to perform the task;
- a_i has to communicate to other agent (a_j) to get information to perform the task.

In the first case, the new evaluation may increase or decrease a_i's confidence degree regarding t_n. If the user's evaluation was good, then the confidence degree will be increased, which means that the agent is becoming more expert in type of task t_n. On the

```
<trustFlight>
 <trustAg1>
  <trust>0.9375</trust>
  <lastIntPos>2009-11-30 14:46:42.871 BRST</lastIntPos>
  <intPos>4</intPos>
  <intNeg>0</intNeg>
  <intInd>0</intInd>
 </trustAg1>
 ...
 <trustAg10>
  <trust>0.5</trust>
  <lastIntPos>2009-11-29 21:11:54.899 BRST</lastIntPos>
  <lastIntNeg>2009-11-29 19:48:38.239 BRST</lastIntNeg>
  <intPos>1</intPos>
  <intNeg>1</intNeg>
  <intInd>0</intInd>
 </trustAg10>
</trustFlight>
```

Fig. 6. XML file of trust in flight service of agent a_2.

other hand, if the user's evaluation was not good, the confidence degree will be decreased. The task evaluation is then used in the agent confidence degree computation so that the agent increases the confidence regarding a type of task when it solves the task in a better way. As the confidence degree represents how much an agent is turning expert in a specific type of task, it is used by the agent to choose the task it will perform. For example, if the agent always has solved tasks about flights, probably it would have better results performing another flight task rather than a hotel task. Thus, agents consider their confidence degrees when they choose the next task to perform. When the list of tasks L is available, the agent will choose one task that belongs to the type t_n with the greater confidence degree. For example, if its confidence degree is 0.4 in flight, 0.6 in hotel and 0.2 in attraction, the agent will choose a hotel task to perform because it is more expert in hotel tasks than others.

In second case mentioned above, both the a_i's confidence degree and the trust degree a_i has on a_j may change. When a_i exchanges information with a_j, its confidence degree may increase or decrease. Moreover, the trust degree a_i has on a_j also may increase or decrease, according to the user's evaluation

$$T_{a_i,a_j}^{t_n} = \frac{\sum_{\theta=1}^{\eta} v_\theta^{(t)} \cdot e^{-\tau \cdot \theta}}{\sum_{\theta=1}^{\eta} e^{-\tau \cdot \theta}} \qquad (2)$$

We use Eq. (2) to update trust degrees where:

- $v_\theta^{(t)} \in [0,1]$ is the user evaluation of the type of task t_n at time θ;
- τ is a constant;
- η is the time step in which the new evaluation arises.

$T_{a_i,a_j}^{t_n}$ returns a value between 0 and 1, where 0 represents the minimum trust degree of a_i in a_j, and 1 represents the maximum trust degree of a_i in a_j.

τ has to be set according to the number of days (θ) the evaluation lasts. Eq. (2) must return 0 when the evaluation period is over. For example, if $\theta = 50$ (meaning that the evaluation should last 50 days), τ should be such that it returns zero when $\theta = 50$ ($\tau = 0.088$ in this case). When updating trust degrees we consider that most recent evaluations have more influence than older evaluations.

3.6. Agent's knowledge base

In MATRES, each agent has its knowledge base that stores the recommendations performed by itself. All the recommendations are stored as cases. Each case is composed by the description of the problem (the set of preferences of the user) and the description

of the solution (the information recommended to that user's query).

Each task performed by the agent generates a case in its knowledge base with:

- **The user's query**: the query is the set of needs and preferences chosen by the user. It represents the description of the problem and contains:
  - **The travel needs**: agents use the main travel needs of the user as filters during the search process. The user does not compromise the needs. If the user wants to go to Rome, for instance, the system cannot recommend travel services to Madrid.
  - **Preferences of the user**: this set of preferences represents additional wishes of the user, such as the hotel category or the flight class. The preferences may differ from needs because user may compromise some preferences. For instance, if the user prefers a hotel in the top category rather than a tourist one, it is still possible to violate this requirement.
- **The recommendation**: the travels recommended to the user. These travels represent the solution of the problem, i.e., the solution for the user's query. The recommendation is partitioned by tasks: **flightR**, **accommodationR**, and **attractionR**. In addition to the recommendation items, each agent stores information about the recommendation:
  - *source*: the source of the information recommended. The source can be the agent that managed the task itself, or another agent with whom it has communicated to obtain the solution.
  - *status*: the status of the information exchanged with other agent, that can be *true* or *false*. The status indicates if the information is valid. This status is set to *false* when the agent finds out that the information received from other agent is no longer true.
  - *evaluation*: the user's evaluation for each attribute of the received recommendation.

Each case is stored in XML format. Table 1 shows an example of a case stored in the knowledge base of *AgentWorker2*, with the user's query and the recommendation presented to the user. Table 2 shows the evaluation of the user for each travel service recommendation received.

3.7. Performing the tasks

In order to perform a task, each agent may search for the information necessary to complete the recommendation. For example, if a_1 has chosen flight task and the user set his preferences (New York, daytime, no stops and no connections), a_1 has to search for the information about flights to New York in order to recommend to the user. In MATRES, there are three types of searching process: LocalSearch, CommunitySearch and SearchWeb.

3.7.1. LocalSearch

In LocalSearch, an agent searches for the information in its knowledge base. As mentioned in Section 3.6, the user may set needs and preferences in the query. As *needs* cannot be compromised, first the agent retrieves a set of cases in its knowledge base that matches the user needs (destination, travel date, return date and number of passengers). Then, the agent searches in the set of cases and retrieves the most similar case giving the user's preferences. Then, the most similar case is presented to the user (as shown in Fig. 7).

The agent then compares the user's preferences with the attributes of all cases. This comparison is done using Euclidean

Table 1
Example of a case – AgentWorker2.

```
<case>
<user>
  <username> JohnDoe </username>
  <password> 1234 </password>
</user >
<query>
    <travelneeds>
      <departure> Guarulhos </departure>
      <destination> New York </destination>
      <departuredate> 01/01/2010 </departuredate>
      <returndate> 01/10/2010 </returndate>
      <passengers> 2 </passengers>
      </travelneeds>
    <Preferences>
      <flight>
        <class> Economic </class>
        <typeflight> Daytime </typeflight>
        <stops> No </stops>
        <connections> No </connections>
      </flight>
      <hotel>
        <category> Touristic </category>
        <room> Double </room>
        <wifi> Yes </wifi>
        <pool> Indoor </pool>
        <pets> No </pets>
        <facilities> None </facilities>
      </hotel>
      <attraction>
        <type>Museum</type>
        </attraction>
</query>
<recommendation>
    <flightR>
        <source>AgentWorker2</source>
        <status>yes</status>
        <destination>New York</destination>
        <cia>Tam</cia>
        <numFlight>8082</numFlight>
        <typeFlight>daytime</typeFlight>
        <departureTime>2009-07-22 08:45:45.515 BRT</departureTime>
        <arrivalTime>2009-07-22 17:45:45.515 BRT</arrivalTime>
        <stops>false</stops>
        <connections>false</connections>
        <fClass>Economic</fClass>
    </flightR>
    <accommodationR>
        <source>AgentWorker4</source>
        <status>yes</status>
        <name>Best Western President</name>
        <destination>New York</destination>
        <category>Touristic</category>
        <type>Double</type>
        <wifi>true</wifi>
        <pool>Indoor</pool>
        <pets>false</pets>
    </accommodationR>
    <attractionR>
        <source>AgentWorker2</source>
        <status>yes</status>
        <type>Museum</type>
        <name>MOMA</name>
    </attractionR>
</recommendation>
</case>
```

Distance, as shown in Eq. (3) where q is the user's query, c is the case that is being compared, f is an attribute, k is the number of attributes and $sim(q_f, c_f)$ is the local similarity. All attributes are symbolic, and the local similarity is 0 when $q_f = c_f$ or 1 otherwise

$$Dist(q,c) = \frac{\sum_{f=1}^{k} sim(q_f, c_f)}{k} \tag{3}$$

Table 2
Example of a case – AgentWorker2 – evaluation of the user.

```
<caseEvaluation>
   <flightE>
      <fClass>true</fClass>
      <typeFlight>false</typeFlight>
      <stops>false</stops>
      <connections>false</connections>
   </flightE>
   <hotelE>
      <category>true</category>
      <type>true</type>
      <wifi>true</wifi>
      <pool>false</pool>
      <pets>false</pets>
   </hotelE>
   <attractionE>
      <like>true</like>
   </attractionE>
</caseEvaluation>
```


Fig. 8. Example of communication among agents.

Dist(q, case1) = 0.45 and comparing the user's query with *case 2* we would have:
Dist(q, case2) = (sim(economic, business) + sim(night, daytime) + sim(Yes, No) + sim(No, No) + sim(touristic, touristic) + sim(double, double) + sim(No, No) + sim(Indoor, Indoor) + sim(No, No) + sim(None, None) + sim(Show, Show)).
Dist(q, case2) = 0.27.

The case with the lowest value is considered the most similar and it is shown to the user as a recommendation. In this example, *case 2* would be presented to the user.

If the agent does not find the required information in its knowledge base, then it starts the communication with other agents.

3.7.2. CommunitySearch

In MATRES, agents are able to communicate with each other, cooperating during the recommendation process. When an agent does not find information in its knowledge base, it may ask for the information necessary to perform its task. As agents were created using JADE, the communication between agents follows the FIPA standard.

Fig. 8 shows an example of communication where a_3 does not have information about attractions in New York City and it has to communicate to other agents. There are three available agents: a_1, a_4 and a_7. a_3 could broadcast a message to all available agents. However, in order to avoid unnecessary communication among agents, they use the trust degree to choose which agent it will communicate to. Then, a_3 evaluates the trust degree it has in each available agent (0.25 in a_1, 0.5 in a_4 and 0.75 in a_7) and it starts the communication with a_7, asking for information of attractions in New York City.

3.7.3. SearchWeb

When *AgentWorker* does not find information either in its knowledge base and in the community of agents, it starts the SearchWeb process. A search tool was developed, which enables the *AgentWorker* to search for the information needed to complete its task. Based on the user preferences and according to the type of task, the agent searches for the information in the web sites available on the Internet.

In flight tasks, for instance, the agent searches in flight companies. The drawback is that the web sites must be previously registered and the strength of the tool is that the agent searches according to the need, which means that there are no unnecessary searches.

3.8. Assumptions

As mentioned in Section 3.1, tasks may be interdependent, which means that some tasks should be performed before than others. However, as the agents work asynchronously, sometimes

Fig. 7. Example of the recommendation presented to the user.

Considering a query example (*q*) where the user has chosen the following preferences: *flight* (class: economic, type: night, stops: Yes, connections: No), *hotel* (category: tourist, room: double, WiFi: Yes, pool: Indoor, pets: No, facilities: None), and *attraction* (type: show), the agent would compare this query with all the cases stored in its knowledge base.
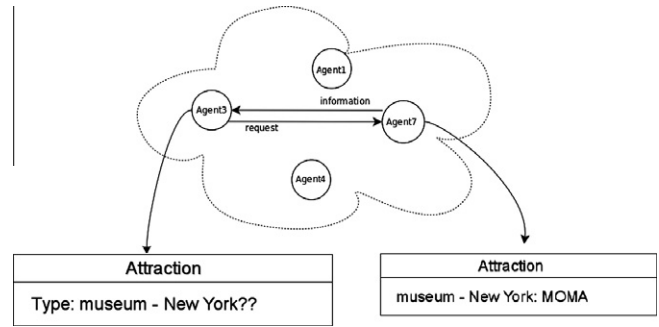
Let us assume that the agent's knowledge base has two different cases stored:

Case 1: *flight* (class: economic, type: daytime, stops: No, connections: No), *hotel* (category: tourist, room: single, WiFi: No, pool: Indoor, pets: No, facilities: None), and *attraction* (type: museum) and
Case 2: *flight* (business, daytime, No, No), *hotel* (tourist, double, No, Indoor, No, None), and *attraction* (show).

Comparing the user's query with *case 1* we would have:

Dist(q, case1) = (sim(economic, economic) + sim(night, daytime) + sim(Yes, No) + sim(No, No) + sim(tourist, tourist) + sim(double, single) + sim(Yes, No) + sim(Indoor, Indoor) + sim(No, No) + sim(None, None) + sim(Show, Museum)) .

the ordering of tasks is not respected. When this happens, the agent who is performing a given task that needs information from other task(s) is not able to complete its task.

For instance, in order to decide the best hotel offer, an agent needs to know at which day and time the passenger arrives at its destination. This example shows that in such dynamic domains tasks may be interdependent. This interdependency among tasks may cause some problems: an agent may have to wait too long for the information from another task, or an agent may have no information if some other agent never finishes its task. Also, agents may be off-line, thus unable to share results and information. To solve this problem, as mentioned, we propose that agents use assumptions. Assumptions allow agent to reason with incomplete information by making guesses.

Using assumption, when $Pred(t) \neq \emptyset$, the agent a_n working on t_n may generate a set of k assumptions given by $\mathcal{S}_{a_n} = \{s_{j_1}, \ldots, s_{j_k}\}$, where each s_{j_l} represents a different assumption that a_n can make during the recommendation process for the attribute i_{j_l}. If, for instance, the agent is performing a hotel task it may use the following set of assumption: $s_2 = \{daytime\}$ that represents the assumption about the user's preference on type of flight.

3.8.1. Formulating assumptions

In order to use assumptions during the recommendation process, it is necessary to formulate the set of assumptions that agents may access. We propose three different methods to formulate the set of assumptions:

1. *The most popular option in the community of users*: if is the first time of the user in the system and he has no profile yet, then the agent sums the number of occurrence of each option for each attribute in the past travels of all users stored and the most popular option will be used by the agent as a set of assumptions.
2. *Similar cases*: the agent searches for the most similar user in its knowledge base and uses the options to the new user. In order to find the most similar user, the agent uses the Euclidean Distance, comparing attributes of the new user with all users in the knowledge base. Here, it is necessary to define a similarity threshold.
3. *The most popular option for the user*: the agent sums the number of occurrence of each option for each attribute in the past travels of the user stored in the case base. The most popular options will be used by the agent as a set of assumption. This method can be applied only for known users because it is necessary to have past cases.

As mention before, in MATRES the hotel and attraction tasks depend on the information from the flight task. In hotel task, for instance, the agent needs information about the arrival time in order to accomplish its recommendation.

We claim that the use of assumptions in the recommendation process helps the agents to complete their tasks in an acceptable time for the user, without jeopardizing the performance of the system. This proposition will be validated next.

4. Experiments and simulations

A community of ten *Agentworkers* was created and three different simulations were run in order to measure the effectiveness of the agents.

4.1. Knowledge acquisition

In the knowledge acquisition step, the knowledge bases of the agents were created with 30 cases obtained from a travel agency.

These cases were composed by the request of the users, the recommendation made by the travel agents and the users' evaluations. Each case was composed by three travel services (flight, accommodation and attraction) which represents 90 tasks. These solved tasks were inserted into the knowledge base of the ten *Agentworkers* at random.

4.2. Simulations

In order to validate the proposed approach, we took into account the **quality of the recommendation**. In recommender systems, users want to receive reasonably good recommendations and they did not like suggestions that depart considerably from their requirements. Therefore, we have run simulations in order to analyze the results with respect the quality of the recommendations (effectiveness) brought by the agents. Thus, we run three different kind of validations of the quality of the recommendations presented for the users:

- **Validation 1: With cooperation among agents**: Agents were allowed to use information from their knowledge bases and/or to exchange information among themselves in order to complete their tasks. In this case they did not make assumptions. The goal of this kind of validation was to verify how good is the result when agents cooperate to solve a task, exchanging information. In a real travel agency the exchanging of information to compose a travel package is a common practice.
- **Validation 2: Making assumptions – Method 1 (the most popular option in the community of users)**: In this validation, agents did not cooperate. Rather, they made assumptions when information was lacking. In order to solve new queries agents made assumptions through *Method 1* searching for the most popular option in the community of users.
- **Validation 3: Making assumptions – Method 2 (similar cases)**: In this validation, again, no cooperation was allowed. Assumptions were made using *Method 2*, i.e., searching for the most similar users.

In order to run the simulations, the travel agency has provided 35 recent test cases (travel packages sold in October and November/2009 in the travel agency) to be used as queries. The travel agency chose these months because the cases were quite heterogeneous. These new 35 queries represent 105 tasks, since each query has three travel services: flight, accommodation and attraction. Then, these 105 queries were run using each of the three validation types just mentioned. The results were compared to verify the effectiveness of the agents. The quality of the recommendations generated by the agents in all validations were compared to the results obtained from human travel agents.

In these validations we did not run *method 3* (the most popular option for the user) because in this experiment we did not consider queries of existing users. We were interested in validating the effectiveness of the agents exploring assumptions for new users. We remark that this is considered by human experts to be more difficult than making assumptions about known users. Further studies will explore the effectively of recommendations to known users.

Before running the validations, it was necessary to set the parameters involved in the recommendation process:

- The confidence degree of the agent for each $t_n - \left(T^{t_n}_{a_i, a_i} \right)$ was defined as 0.5. This means that agents do not behave like experts in the beginning of the execution.
- The confidence degree of the agent in other agents – $\left(T^{t_n}_{a_i, a_j} \right)$ was defined as 0.5. This means that agents do not have opinion about other agents in the initial stage of the system.

- τ was set as 0.058 that represents 90 days for all types of tasks t_n ($e^{-0.058\cdot90} = 0$). According to the human travel agent, 90 days is the ideal lasting time of an evaluation.
- Each t_n may have a maximum number of evaluations. This number corresponds to the maximum number of preferences (i_m) defined for each t_n. We have defined four attributes for flight service (*flight* = 4), five attributes for hotel service (*hotel* = 5), and one attribute for attraction service (*attraction* = 1). Thus, the higher the number of the evaluation, the better the result.
- Agents were able to make three different types of assumptions: *arrival date*, *arrival time*, and *departure date*, i.e., $\mathcal{S}_{a_n} = \{s_1, s_2, s_3\}$. These assumptions were necessary to compose the accommodation of the customer and the available attractions in the destination city.
- The similarity threshold used in *method 2* (similar cases) was set as 0.5, which means that the acceptable distance between the cases and the user must be in the range of 0–0.5.
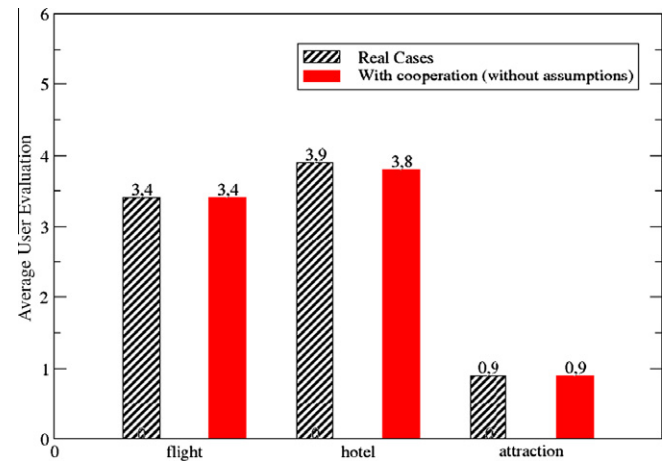
The advantage of using real cases as new queries is that we can measure the output of our approach on realistic problems and we can use the case solutions as correct evaluations of the recommendations. Table 3 presents the queries used in the simulations.

Fig. 9 shows the results obtained from the first validation where new queries were performed by the agents without making assumptions. Agents were allowed just to exchange information among themselves. In flight and attraction tasks, agents obtained roughly the same evaluation as the human expert. However, in hotel task, agents received a worse evaluation than the human expert.

Analyzing the results of validation 1 we observed two weakness in our trust mechanism:

Fig. 9. Users evaluation – real cases × agents (with cooperation among agents, without assumptions).

1. Agents were very severe about the trust degree on other agents. If a_i did not trust a_j, even if a_j was an expert in the type of task t_n, a_i did not communicate with a_j to get information;
2. When the trust degree of a_i in a_j decreased, a_i started to not trust in a_j and it did not consider a_j in future communications.

These weakness contributed to the poor result obtained by the agents in this validation. Thus, we decided to apply a probabilistic method in the trust approach to avoid this too selective behavior. There are three kinds of labels associated to a communication:

Table 3
Cases used as new queries in the simulations.

Cases	Destination	PAXs	Class	Type	Category	Room	Attraction
1	Lisbon	2	Economic	Day	Tourist	Double	Monument
2	Paris	2	Economic	Night	Tourist	Double	Museum
3	Punta Cana	2	Economic	Night	Luxury	Double	Show
4	Milan	1	Economic	Night	Tourist	Single	Monument
5	Lisbon	2	Economic	Night	Tourist	Double	Monument
6	London	2	Economic	Night	Tourist	Double	museum
7	Cancun	2	Economic	Night	Luxury	Double	Show
8	Miami	2	Economic	Night	Tourist	Double	Show
9	Amsterdam	1	Economic	Night	Tourist	Single	Museum
10	Madrid	1	Economic	Night	Tourist	Single	Monument
11	New York	3	Economic	Day	Tourist	Triple	Shows
12	Rome	2	Economic	Night	luxury	Double	Monument
13	Porto	1	Economic	Day	Tourist	Single	Museum
14	Valencia	2	Economic	Night	Tourist	Double	Museum
15	Dublin	3	Economic	Night	Tourist	Triple	Shows
16	Edinburgh	2	Economic	Day	Tourist	Double	Monument
17	Barcelona	3	Economic	Night	Tourist	Triple	Museum
18	Orlando	3	Economic	Night	Tourist	Triple	Parks
19	Rio de Janeiro	2	Economic	Night	Tourist	Double	Monument
20	Madrid	2	Economic	Night	Tourist	Double	Museum
21	Aruba	2	Economic	Night	Tourist	Double	Beach
22	Fortaleza	1	Economic	Night	Tourist	Single	Show
23	New York	2	First	Day	Luxury	Double	Shows
24	Rio de Janeiro	1	Economic	Day	Tourist	Single	Shows
25	Lisbon	2	First	Day	luxury	Double	Monument
26	Madrid	1	Economic	Night	Tourist	Single	Monument
27	Porto Alegre	3	Economic	Day	Tourist	Triple	Monument
28	Recife	2	Economic	Night	Tourist	Double	Beach
29	San Andres	2	Economic	Night	Luxury	Double	Beach
30	Mexico City	3	Economic	Night	Tourist	Triple	Museum
31	Curacao	2	Economic	Night	Luxury	Double	Beach
32	Dubai	2	Economic	Day	Luxury	Double	Beach
33	Orlando	3	Economic	Night	Tourist	Triple	Parks
34	Rio de Janeiro	1	Economic	Day	Tourist	Single	Shows
35	Porto	2	Economic	Day	Tourist	Double	Monument

towards *trusted* agents, towards *new* agents, towards and *old* agents. *Trusted* represent those agents that are trusted in that moment; *new* agents are those agents that are not yet trusted; and *old* represent agents already trusted in the past but whose trust degree is low in the present moment. This is associated with probability values that guide the communication.

Using this probabilistic approach, results are as shown in Table 4. In *hotel* task agents improved the average evaluation from 3.8 to 4.1.

Fig. 10 shows the results obtained from the human travel agents compared to the results obtained from *method 1*. In *flight* and *attractions* tasks, the results were very similar in both simulations. In *hotel* task, agents got better evaluations than the human travel agents (4.0 compared to 3.9). However, it was a modest improvement and this result is not significant (t-test). Since in *method 1*
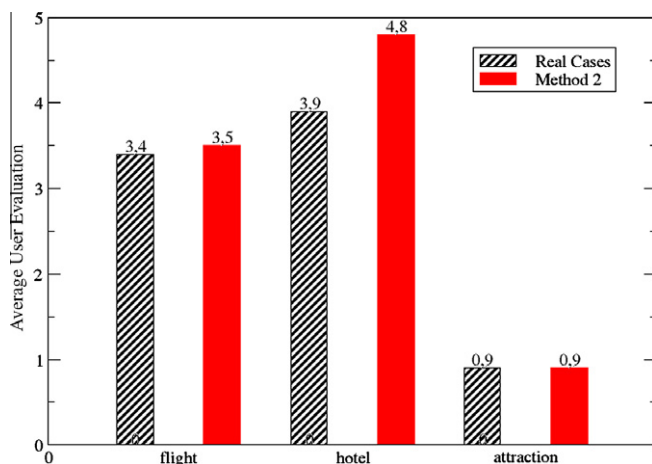
Table 4
Averages of user evaluations for each travel service.

Simulations	Averages		
	Flight	Hotel	Attraction
Real cases	3.4	3.9	0.9
Communication	3.4	4.1	0.9
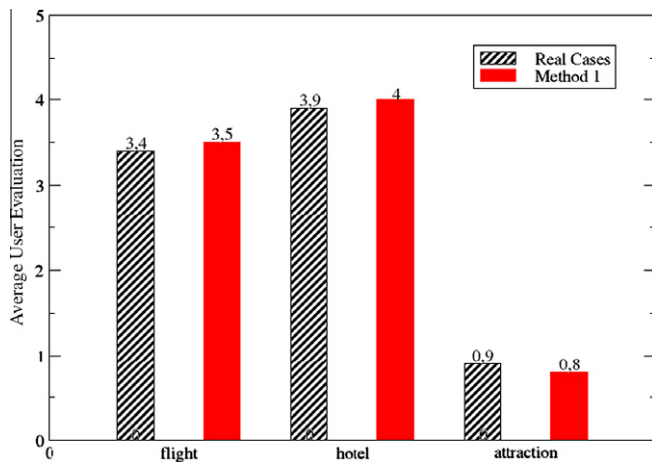
Fig. 10. Users evaluation – real cases × method 1 (with assumptions – the most popular option of users).

Fig. 11. Users evaluation – real cases × method 2 (with assumptions – similar cases).

agents use the most popular option of all users, the small improvement shows that users have different tastes. Thus, assuming the most popular option is not efficient if users are *grey sheep*, for example.

In Fig. 11 we can see the comparison of results obtained from the travel agents and the results obtained from *method 2*. Different from what happened with *method 1*, *method 2* produced better evaluations. The average users' evaluation for the *Hotel* services produced by *method 2* is 4.8. This is a good result considering that the best result is 5 (if all recommendations are evaluated good for all the items). 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* produces better results for the *Hotel* tasks, compared with *method 1* and with the expert solution. We have run a t-test and have observed that both differences (between this value and the corresponding results for the expert solution and *method 1*), are statistically significant. It means that using similar cases is a good strategy to make assumptions for new users when necessary.

5. Conclusion and future work

This paper presented MATRES, an assumption-based multiagent recommender approach where agents cooperate in order to recommend travel packages to the user. MATRES is a novel application that recommends different travel services for the user, considering the users preferences in each travel service.

MATRES is able to generate personalized experiences to customers that are searching for tourism options. Also, MATRES presents important features such as:

1. It is able to compose a travel package considering different travel services (flight, hotel and attractions) respecting the dependencies among these travel services.
2. It considers the user preferences and past travels, avoiding the overload of recommendations to the user.

In MATRES agents are able to make assumptions during the recommendation process. Assumptions are useful in a multiagent system, especially when agents perform interdependent tasks (e.g. the result of a task is necessary to perform another one). In these cases, agents cannot wait for the information. We have shown that the assumption-based multiagent recommender approach helps agents to improve their recommendations when they have to deal with these problems.

Another contribution is the ability of trusting other agents. The trust degree helps to improve the exchange information process among agents. By using degrees of trust, agents are able to decide which task they should perform.

Some limitations of this approach should be mentioned. First, the prototype was developed with the most important attributes for the travel package recommendation indicated by the travel agent expert. However, the number of attributes involved in the recommendation process has an implication on the number of possible assumptions. In our experiments we had only three possible assumptions that agents could make.

Second, the trust approach originally proposed did not yield good results in every case. Thus, some adjusts were needed, as already explained in Section 4.2.

As future work we want to extend the assumption mechanism by inserting the possibility of agents sharing assumptions. This feature will improve the system performance since agents would be able to use other agents assumptions rather than compute them by themselves.

References

Adomavicius, G., & 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.

Camacho, D., Molina, J. M., Borrajo, D., & Aler, R. (2002). Mapweb: Cooperation between planning agents and web agents [Special issue on agent-based technologies]. *Information and Security, 8*(2), 209–238.

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

Drumond, L., Girardi, R., & Leite, A. (2007). Architectural design of a multi-agent recommender system for the legal domain. In *ICAIL '07: Proceedings of the 11th international conference on Artificial intelligence and law* (pp. 183–187). New York, NY, USA: ACM.

Goy, A., Ardissono, L., & Petrone, G. (2007). Personalization in e-commerce applications. In *The adaptive web* (pp. 485–520). Berlin/Heidelberg: Springer.

Lorenzi, F., Bazzan, A. L. C., & Abel, M. (2009). Multiagent truth maintenance applied to a tourism recommender system. In N. Sharda (Ed.), *Tourism informatics: Visual travel recommender systems, social communities and user interface design. information science reference* (pp. 54–72).

Macho, S., Torrens, M., & Faltings, B. (2000). A multi-agent recommender system for planning meetings. In *Workshop on agent-based recommender systems*.

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

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings ACM conference on computer-supported cooperative work* (pp. 175–186).

Torrens, M., Faltings, B., & Pu, P. (2002). Smartclient: Constraint satisfaction as a paradigm for scaleable intelligent information systems. *CONSTRAINTS: An International Journal, 7*(1), 49–69.

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

Wei, Y., Moreau, L., & Jennings, N. (2003). Recommender systems: A market-based design. In *Proceedings second international joint conference on autonomous agents and multi agent systems (AAMAS03), Melbourne, Australia* (pp. 600–607).

Zhang, D., Simoff, S., Aciar, S., & 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.