Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-010-9147-0

Learning opponent's preferences for effective negotiation: an approach based on concept learning

Reyhan Aydoğan · Pınar Yolum

Abstract We consider automated negotiation as a process carried out by software agents to reach a consensus. To automate negotiation, we expect agents to understand their user's preferences, generate offers that will satisfy their user, and decide whether counter offers are satisfactory. For this purpose, a crucial aspect is the treatment of preferences. An agent not only needs to understand its own user's preferences, but also its opponent's preferences so that agreements can be reached. Accordingly, this paper proposes a learning algorithm that can be used by a producer during negotiation to understand consumer's needs and to offer services that respect consumer's preferences. Our proposed algorithm is based on inductive learning but also incorporates the idea of revision. Thus, as the negotiation proceeds, a producer can revise its idea of the consumer's preferences. The learning is enhanced with the use of ontologies so that similar service requests can be identified and treated similarly. Further, the algorithm is targeted to learning both conjunctive as well as disjunctive preferences. Hence, even if the consumer's preferences are specified in complex ways, our algorithm can learn and guide the producer to create well-targeted offers. Further, our algorithm can detect whether some preferences cannot be satisfied early and thus consensus cannot be reached. Our experimental results show that the producer using our learning algorithm negotiates faster and more successfully with customers compared to several other algorithms.

Keywords Negotiation · Preference Learning · Ontology Reasoning · Disjunctive Preferences

1 Introduction

In a typical e-commerce application, the producers advertise and provide services. The consumers request and possibly consume these services. The range of services is broad. A service may be selling a book, reserving a hotel room, and so on. The preferences or interests of the

R. Aydoğan (✉) · P. Yolum
Department of Computer Engineering, Bogazici University, 34342 Bebek, Istanbul, Turkey
e-mail: reyhan.aydogan@gmail.com

P. Yolum
e-mail: pinar.yolum@boun.edu.tr

Published online: 13 August 2010

participants may vary based on the service. As it happens in real life, some conflicts may occur. For example, the producer may prefer to charge a high price for a service whereas the consumer may prefer a lower price. When there is a conflict in the preferences of the participants, negotiation—the process of resolving conflicts and finding mutual acceptable agreements—takes place [16,26]. During this process, the participants try to reach a consensus by offering alternatives.

Traditional e-commerce applications are targeted for human users. However, as the number and extent of transactions increase, there is a tremendous demand for developing flexible, intelligent e-commerce applications that can help users fulfill their tasks. Agents have proven to be a successful paradigm for autonomous and intelligent software that can represent users and act on behalf of them. This paper studies agent-based service negotiation where service producers and consumers are represented by agents.

The simplest negotiation takes place between two agents on a single issue, such as price. Two agents interact to settle on a value for that single issue. The more complex negotiations take place over multiple issues [1,26]. In a multi-issue negotiation the importance of the issues may vary for the participants. One agent may consider a particular issue more important whereas the other agent may take another issue into account. However, in a multi-issue negotiation, it is possible to have trade-offs between issues, making consensus more plausible. For instance, the delivery time may be the main concern for a particular consumer whereas the price of the service may be more important for the producer. If the consumer pays more for fast delivery, both agents are at an advantage as far as their preferences are concerned. Meanwhile, when there is more than one issue to be considered, the search space for the acceptable agreements increases, which complicates the entire negotiation process.

When agents are given a large search space, an important challenge is to find ways to generate requests or counter offers. This is determined by the agent's negotiation strategy. A good negotiation strategy should not only consider the agent's own utility but the utility of the opponent as well. Otherwise, no matter how good the generated offer is for the agent, it will not be accepted by the other agent. To find an agreement, which is mutually beneficial for both participants, the agents need to have sufficient knowledge about the negotiation domain and to take the other agent's preferences into account [14]. However, preferences of participants are almost always private and hence cannot be accessed by others. If the agent shares its preferences with the other agent, this information can be exploited by the opponent agent [14]. For example, if the producer knows that the buyer can pay up to 100 USD for the service, it may not offer a price lower than 100 USD, although it can possibly afford to provide the service for 80 USD. As a result, this negotiation will end up with a lower gain for the buyer. Thus, the agent may not prefer to reveal its preferences completely. Alternatively, the preferences of the agents may be complicated. Because of the communication cost, these preferences may not be shared [19]. The best that can happen is that participants may learn each others' preferences through interactions over time.

As agents learn each others' preferences, they can provide better-targeted offers and thus enable faster negotiation. Learning and understanding the preferences of the opponent agent reduce the search space for the alternatives, which leads to more efficient negotiation. This may come up with an earlier consensus, which also reduces the communication cost. Hence, learning opponent agent's preferences and reasoning on these learned preferences constitute an irreplaceable part of negotiation.

This paper studies service negotiation that takes place between a consumer and a producer agent to reach a consensus on a service description. The service description consists of various attributes of a service. The consumer and the producer interact by turn taking: The consumer starts the negotiation by requesting a service. If the producer cannot fulfill

this need, it proposes a counter offer and so on. We call these requests and offers that are exchanged bids. The preferences of the consumer are represented in the form of conjunctives and disjunctives. During this process, the consumer generates its requests according to its private preferences while the producer agent generates its counter offers from its available services that are ranked according to their profitability for the producer. The producer agent prefers an offer whose gain is higher for itself. However, it does not only consider its own preferences but also considers the consumer's need. Thus, the producer generates an offer, that is both likely to be preferred by the consumer and profitable for the producer. To do this, we develop an algorithm that is used by the producer to learn the consumer's preferences from bid exchanges during the negotiation. The algorithm has the following properties:

- **Inductive:** We expect the producer to build a model of the consumer's preferences by looking at the bids and predict whether a potential offer will be accepted by the consumer. If the model predicts rejection, there is no need to offer that to the consumer.
- **Incremental:** The learning algorithm uses the bids exchanged as training instances. Since these bids become available during the negotiation, the appropriate learning algorithm should be incremental.
- **Supports Disjunctive Preferences:** When preferences are represented as constraints on the values of issues, two basic variations are possible: conjunctive constraints or disjunctive constraints. An example to the conjunctive constraints is the following: The customer prefers red and dry wine. Constraints on both color and body of the wine need to be satisfied to please this customer. However in many realistic scenarios, participants' preferences are disjunctive. For instance, the customer prefers red or dry wine, meaning that satisfying either of the constraint is enough to please the customer. Preference representation allowing both conjunctions and disjunctions of the preference constraints are much more powerful and realistic then the representation supporting only conjunctions. Accordingly, our algorithm should learn both conjunctive and disjunctive preferences.
- **Ontology-Based:** In addition to learning consumer's preferences using exchanged bids, the producer can reason on the domain knowledge using an ontology. With an ontology, we can capture information about the relations between issues and use these relations when generating offers. Following the previous example, if a red and dry wine cannot be supplied to a customer, it would be better to find a similar wine (e.g., semi-dry, red wine) than a randomly-picked wine.
- **Retractable:** The algorithm should be able to retract its findings as more bids are exchanged. This is integral to the incremental nature of negotiation. As more bids become available, previous models of the consumer may no longer be accurate. Hence, the algorithm should adapt to this.

Two inductive learning approaches that can be adopted for this purpose are: Candidate Elimination Algorithm (CEA) [23], which is based on building and maintaining version spaces and ID3 [24], which is based on building and classifying decision trees. CEA, by design, is incremental, whereas ID3 needs to be modified to make it incremental. Hence, we take CEA as our starting point. ID3 supports learning disjunctives, but CEA does not. In either case, the domain knowledge cannot be utilized by the algorithm. To improve the negotiation process, we need a learning algorithm that both supports disjunctives and uses the domain knowledge in a way that the agent is able to reason on the attribute values. We combine these ideas in an extension of CEA, called Revisable Candidate Elimination Algorithm (RCEA) [4], which is based on CEA. RCEA is incremental and inductive as CEA, but it also supports learning disjunctive concepts, can utilize an ontology, and can retract its hypothesis about what it has learned as more interactions take place.

Several approaches have been developed in the literature to enable an agent to generate bids. In many negotiation frameworks, preferences are represented as utility functions [11,14]. Given a service, a utility function can calculate how beneficial that service is for the user. One trend is to use opponent's last request to generate a new offer. The intuition is that among several alternatives, the service that is most similar to the opponent's last request, is most likely to be accepted by the opponent [11]. Another trend is to analyze the opponent's all previous requests, to model them as a utility function and to learn this function. Methods that have been applied to learn these utility functions include Bayesian learning [14,32] and Genetic algorithms [9]. We also propose to use all bid information to learn the opponent's preferences, however we do not represent the preference model as a utility function, but as constraints on possible services. The main motivation for this is that users can represent their preferences more easily with qualitative preferences, such as constraints. To use utility functions, users need to put in a lot of effort to come up with a utility function in the first place. When service issues are interdependent, coming up with a utility function becomes even more difficult. However, with qualitative representations, the preferences can be stated as constraints that need to be satisfied or as ordering of available services. For this reason, it is intuitive to learn the qualitative representation of the opponent rather than casting this as a utility function. To the best of our knowledge, RCEA is the first CEA-based algorithm that benefits from above properties and is used for learning opponent preferences for negotiation.

The producer uses RCEA to learn the customer's preferences and to generate offers that respect them. In other words, instead of blindly searching for agreements in the search space, the producer will do an informed search. The aim of using RCEA is three fold. First, if consensus is possible, we want the agents to find the mutually agreeable service. That is, we want to enable successful negotiations. Second, we want the agents to reach this consensus in as few steps as possible. If a producer offers a possible service after too many interactions, the customer would walk away. Hence, the number of interactions to reach a consensus is important. Third, if consensus is not possible, we want to detect this and terminate the negotiation as early as possible. If the producer does not realize that it would not be able to satisfy the customer's needs in a reasonable time, it would waste the customer's time.

Compared to the existing disjunctive learning approaches, such as DCEA (Disjunctive CEA) [3], naive Bayes' classifiers [2] and ID3, RCEA not only learns the opponent's preferences well but also facilitates faster negotiation of services. If no consensus can be found, RCEA signals this much earlier than DCEA, Bayes' classifier, and ID3.

The rest of this paper is organized as follows. Section 2 provides the necessary technical background on various learning algorithms and negotiation in general. Section 3 explains our negotiation architecture. Our learning algorithm is explained in Sect. 4. Section 5 provides our experimental setup and comparison results with existing algorithms. Section 6 discusses our work with references to the literature.

2 Technical background

In this section we give a brief introduction to several learning algorithms and classifiers such as CEA, DEA, ID3 and naive Bayes' classifier and describe our use of ontologies.

2.1 Preliminaries

In our setting, each service description is represented as a vector of attribute values. For simplicity, let us assume that a service is described with three issues: region, color and sugar

level. For example, *(French, Red, Dry)* represents a red and dry French wine. Both consumer and producer agents express their bids with this representation.

The consumer's preferences are represented as a set of acceptable service descriptions. The only major difference is that in describing preferences, attributes may have a value of "?" ,which means that any value is acceptable for that attribute. Our representation allows preferences to be specified as conjunctive and disjunctive constraints on attribute values. Here a conjunctive constraint means that each individual constraint is connected with the "and" operator ($\wedge$) and if and only if all of the individual constraints are satisfied, we say that the whole constraint is satisfied. Otherwise, it is not satisfied. In our setting, each individual acceptable service description is formed as a conjunctive constraint. To illustrate this, consider *(?, Red, Dry)*. We can interpret this service description as (Region = ? $\wedge$ Color = Red $\wedge$ Sugar = Dry). Similarly, a disjunctive constraint means that each individual constraint is connected with the "or" operator ($\vee$) and if at least one of them is satisfied, the whole constraint is satisfied. If none of the individual constraints are satisfied, we say that it is not satisfied. For example, *{(?, Red, Dry), (French, ?, ?) }* means that any red and dry wine or any French wine is acceptable for the consumer.

When the consumer and producer interact, they observe each other's bids. From these bids, producer tries to learn the consumer's preferences as stated above. Since we are interested in applying supervised learning methods, we need some training data to train our algorithm and then test on some data. The bids exchanged between the consumer and producer constitute the training set of the producer's learning algorithm. When a consumer makes a request, the producer interprets this request as a positive training instance, since if it were not consistent with the consumer's preferences, the consumer would not have requested it in the first place. When the producer makes an offer that is not accepted by the consumer, the producer interprets that service description as a negative training instance, since if it were acceptable, then the consumer would have taken that offer. After each request and offer, the algorithm is trained. This training enables the producer to have a set of hypotheses of what the consumer's preferences may be at any time point. When it is time for the producer to make an offer to the consumer, it first creates a service description and then checks to see if this service description is covered by its hypotheses. That is, the producer would not want to make an offer that is inconsistent with the consumer's preferences.

When the producer agent tries to learn the consumer's preferences, it takes consumer's each service request as a positive sample and producer's each service offer rejected by the consumer is accepted as a negative example. The learned service descriptions (target concept) consist of hypotheses that are represented as a set of attribute values. For example, a possible hypothesis can be *(?, ?, Dry)* that covers the services whose sugar level is dry. In detail, the wine service *(Italian, Rose, Dry)* is covered by this hypothesis. The learned hypotheses are used to decide whether the available service is possibly be rejected by the consumer or it may be an acceptable service with respect to consumer's preferences. During the negotiation, the producer agent filters out its available services, which seems to be rejected by the consumer. Among the remaining services, it offers the most convenient service for both agents.

2.2 Candidate elimination algorithm (CEA)

CEA is an inductive learning algorithm that is based on version spaces in which a target concept is learned from the observed examples [23]. In a version space [22], there are two significant hypothesis sets: the most general (G) and the most specific (S). G includes the general hypotheses whose boundary is as large as possible whereas the hypotheses in S are as specific as possible so that they minimally cover the positive samples.

At the beginning, G contains the most general hypothesis, *(?, ?, ?)* and S contains the description of the first positive sample. When a negative training sample comes, the hypotheses in G are specialized not to cover this sample any more. Example 1 explains this process in detail. If any hypothesis in S covers this negative sample, that hypothesis is removed from S. In such a case, if S contains only one hypothesis, the learning algorithm becomes inconsistent and fails.

Example 1 At the beginning, G includes the most general hypothesis covering all possible services *(?, ?, ?)*. Let assume that the consumer asks for *(Chianti, Rose, OffDry)* as a first request. Since S should cover the positive examples minimally, we directly add the first request to S. When the producer offers *(Chianti, Rose, Sweet)*, the consumer rejects this service because it is not an acceptable service with respect to consumer's preferences. Note that the rejected service offers are taken as negative examples. Since the current G covers this hypothesis, it should be specialized in a minimal way not to cover that sample. To do it, we use the hypothesis in S. We compare the values of the attributes in the negative example with those in the specific hypothesis. When the values are different, we use these values to specialize the current general hypothesis. In this example, only the value of the sugar level is different for them. Thus, G becomes *{(?, ?, OffDry)}*.

When a positive example arrives, the hypotheses in S are generalized to cover this sample. For instance, if S contains *(French, Red, Sweet)* and the current positive example (consumer's request) is equal to *(French, Rose, OffDry)*, S becomes *{(French, ?, ?)}* in order to cover the current positive example. And, the hypotheses in G, which do not cover this positive sample are removed from G. This rule does not allow CEA to learn disjunctive preferences because all general hypotheses cannot be enforced to cover each positive sample when there are disjunctive preferences. Table 1 illustrates how CEA fails in the case of this generalization process.

Consequently, the hypotheses in G become more specific where those in S become more general over time. Eventually, G and S intersect when the target concept is learned. Since the hypothesis sets (G and S) are revised according to current training sample, the order of the training examples affects the learning process. In other words, the same training instances in different order may give different results.

2.3 Disjunctive candidate elimination algorithm (DCEA)

CEA does not support learning disjunctives. DCEA [3] improves CEA to handle disjunctives by extending the hypothesis language to include disjunctive hypothesis in addition to the conjunctives. Each attribute of the hypothesis has two parts: inclusive list, which holds the list of valid values for that attribute and exclusive list, which is the list of values which cannot be used for that attribute.

Table 1 When candidate elimination algorithm fails

Type	Sample	The most general set	The most specific set
+	(French, Red, Dry)	{ (?, ?, ?) }	{ (French, Red, Dry) }
−	(French, Rose, Sweet)	{ (?, Red, ?),(?, ?, Dry) }	{(French, Red, Dry) }
+	(Italian, White, OffDry)	{ }	{ (?, ?, ?) }

Assume that the most specific set has a single hypothesis as *{(California, Red, Sweet)}* and a positive example *(California, Rose, Sweet)* comes. The original CEA will generalize this as *(California, ?, Sweet)*, meaning the color can take any value. However, in fact, we only know that the color can be *Red* or *Rose*. In the DCEA, we generalize the hypothesis as *{(California, [Red, Rose], Sweet)}*. Only when all the values exist in the list, they will be replaced by *?*. In other words, the algorithm generalizes specific hypotheses more slowly than before.

When a positive example comes, DCEA does not eliminate the general hypotheses in G not covering the sample anymore in order to support learning disjunctive concepts. Here, the intuition is that a general hypothesis does not have to cover all positive examples. This positive sample is added as a separate hypothesis into S unless there exists any hypothesis in S that can be merged with this sample. Otherwise, the algorithm combines this sample with that specific hypothesis.

When a negative sample comes, for each hypothesis in G that covers this negative sample, new hypotheses are generated by excluding each attribute value of the negative sample from the original hypothesis. For instance, assume that the negative example is *(Chianti, Rose, Sweet)* and there exists a general hypothesis in G such as *{(?, ReddishColor, ?)}*. Note that *ReddishColor* is a parent concept of *Rose* and *Red*. DCEA is able to use the hierarchical information about attribute values in generalization process. After the negative example, this hypothesis will be eliminated and three new hypotheses will be added. These hypotheses are *{ (?-Chianti, ReddishColor, ?), (?, ReddishColor-Rose, ?), (?, ReddishColor, ?-Sweet)}*. Consequently, all possible general hypotheses are generated. As seen from this example, this process is highly complex and expensive.

2.4 ID3 decision tree algorithm

ID3 is an inductive learning algorithm used in constructing decision trees in a top-down fashion from the observed examples represented in a vector with attribute-value pairs [24]. Unlike CEA, this algorithm supports learning a disjunctive concept.

A decision tree has two types of nodes: leaf node in which the class labels of the instances are kept and non-leaf nodes in which the test attributes are held. The test attribute in a non-leaf node is one of the attributes making up the concept description. Selection of test attributes is a crucial task in construction phase of the tree since it affects the size of tree. Smaller decision trees are preferable since it makes decision quicker. These test attributes are used to divide the examples into subsets by considering the (im)purity of examples in each group. ID3 uses information gain [27], which is a popular criterion for impurity test.

In detail, information gain is the difference of information before the split with that after the split and its value increases with the average purity of subsets. Therefore, we choose the test attribute whose information gain is higher than that of others. In estimation of information gain, an (im)purity measure called entropy [29] represents the homogeneity of examples in subsets after the split process.

After selection of a test attribute, tree splits in accordance with all possible values of that attribute. For instance, if the test attribute is *color* in our wine example, the node will be branched into three parts for the values: red, rose, and white. This operation will be performed for all new nodes recursively. The splitting will be finished when each subset is homogeneous; i.e., have the same class label or have no attribute left for testing.

The problem with this algorithm is that it is not an incremental algorithm, which means all the training examples should exist before learning. To overcome this problem, the system can keep the training instances at each time. After each new coming instances, the decision

Table 2 Example training data obtained during negotiation

Request or offer	Region	Color	Sugar
First request (+):	Chianti	Rose	OffDry
First counter offer (−):	Chianti	Rose	Sweet
Second request (+):	California	White	Sweet
Second counter offer (−):	US	Rose	OffDry
Third request (+)	Italian	Red	Dry

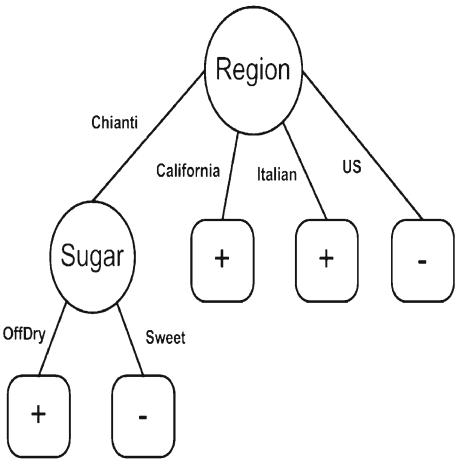

Fig. 1 Sample decision tree

tree can be rebuilt. Without doubt, there is a drawback of reconstruction such as additional process load. Nevertheless, this process load is not significant for our purposes since ID3 algorithm runs fast.

Table 2 shows an example negotiation scenario between agents. After the consumer's third request, the producer agent constructs the decision tree depicted in Fig. 1. The producer starts searching from the root to the leaf nodes in order to classify its available services. For example, according to the constructed tree in Fig. 1, *(US, Red, Sweet)* is classified as negative where *(California, Red, OffDry)* is classified as positive. In some cases, the values of a service may not be represented via branches. For instance, consider *(French, Rose, Dry)*. For the region, we do not have any branch for French. In such a case, ID3 algorithm classifies the service as unknown. Note that during the negotiation the producer only filters the available services, which are classified as negative because they would be possibly rejected by the consumer.

2.5 Bayes' classifier

In order to make decision under uncertainty, a classification based on probabilities can be applied [2]. In this classification, we estimate the probabilities of the classes with Bayes' rule (Eq. 1). In this equation, $P(C)$ denotes the prior probability of an issue belonging to class

C where $p(x|C)$ is the likelihood of observing x when the given issue belongs to C. Note that $p(x)$ is the evidence that is the probability of observing x where $P(C|x)$ is the posterior probability—the probability of the given issue, x belonging to C.

$$P(C|x) = \frac{P(C)p(x|C)}{p(x)} \tag{1}$$

In our negotiation domain, we have two classes: positive (+) and negative (−). As is customary, we assume that all issues are independent while we use Bayes' classifier [14]. Since an offer consists of several issues $(x_1, x_2, \ldots x_n)$, the probability of an offer belonging to a particular class (+ or −) is equal to $\prod_{i \in n} P(C|x_i)$. Instead of multiplying the posterior probabilities, we can take the logarithm of both sides. As a result, the posterior probability of an offer belonging to the given class is equal to $\sum_{i=1}^{n} \log P(C|x_i)$.

To estimate the individual posterior probabilities, the producer keeps all consumer's requests (positives) and its offers rejected by the consumer (negatives) during the negotiation. It is easy to estimate the prior probabilities because it is equal to the ratio of the number of examples belonging to that class to the number of the entire examples. For example, if we have three positive examples and two negative examples, $P(C = +)$ is equal to 3/5 and $P(C = -)$ is equal to 2/5. Since all negotiation issues are discrete, the likelihood is the ratio of how many times x belonging to C is observed to the number of examples belonging to C. To illustrate this, consider the training examples in Table 2. According to this table, $P(x_1|C = +)$ where x_1 has the value of "Chianti" is equal to 1/3 because we have three positive examples and only one of them has the value "Chianti". Finally, $p(x)$ is equal to $P(C = +)p(x|C = +) + P(C = -)p(x|C = -)$.

According to this classifier, the producer estimates posterior probabilities for both negative and positive classes. If the posterior of the negative class is higher than that of the positive class, this service is classified as negative. Note that the producer may have some services whose values have not been seen yet. For example, none of the examples contains *French* as a region. In this case, posterior probabilities for both class are ignored for that issue. As a result, if our service is (French, Rose, OffDry), we only consider the posteriors for color (Rose) and sugar level (OffDry).

2.6 Ontology

An ontology represents knowledge of a given domain [13,21]. Shortly, an ontology contains the specification of concepts and their meanings. We describe the concepts, specify their properties and establish some relationships among them by considering a domain of knowledge or interest. It can be thought as representation of knowledge with its semantics. For instance, consider wine domain [31]. The ontology contains the description of a wine concept including its properties such as color, body, winery and so on. In addition to these properties, the ontology includes some relationships such as "Has-a" and "Is-a". For example, *Chardonnay* is a *Wine* and *Wine* has *color* where color may be one of red, rose or white.

Relationships such as "Has-a" and "Is-a" contribute to reasoning of agents. We can represent a taxonomy or hierarchical information by using the relations, "Is-a" and "subclass". By using these relationships, agents can discover new knowledge from the existing knowledge. For instance, using the wine ontology the following reasoning can be made: *Bordeaux* is defined as a *Wine*, *Medoc* is defined as a *Bordeaux* and *Pauillac* is defined as a *Medoc*. When an agent wants to buy wine, another agent can offer any instance of *Bordeaux*, *Medoc* and *Pauillac* since it can reason that if *Medoc* is a *Bordeaux* and *Bordeaux* is a *Wine* then *Medoc* is a *Wine* and then if *Pauillac* is a *Medoc* and *Medoc* is a *Wine* then *Pauillac* is a wine.

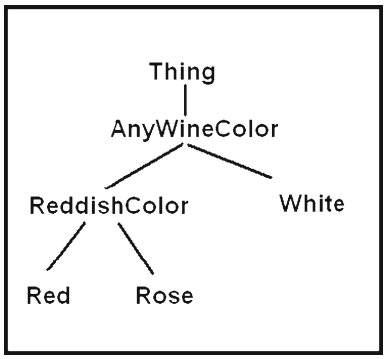

Fig. 2 Sample taxonomy for similarity estimation

2.7 Similarity estimation

During negotiation, an agent will need to compute similarities between service descriptions. To do this, it will need to compare similarities between values of an attribute. We establish this through a similarity metric. Any similarity metric would be sufficient for our architecture. However, our previous comparison of similarity metrics has shown that RP Semantic Similarity Metric works well with ontologies [3] and thus is used in this work. Based on the relative distance between two concepts in a taxonomy, RP metric measures how similar two concepts are. To do this, it exploits the following intuitions. Note that we use Fig. 2 to illustrate these intuitions.

- Parent versus grandparent: Parent of a node is more similar to the node than grandparents of that node. Generalization of a concept results in going further away from that concept. The more general concepts are, the less similar they are. For example, *AnyWineColor* is parent of *ReddishColor* and *ReddishColor* is parent of *Red*. Then, we expect the similarity between *ReddishColor* and *Red* to be higher than that of the similarity between *AnyWineColor* and *Red*.
- Parent versus sibling: A node would have higher similarity to its parent than to its sibling. For instance, *Red* and *Rose* are children of *ReddishColor*. In this case, we expect the similarity between *Red* and *ReddishColor* to be higher than that of *Red* and *Rose*.
- Sibling versus grandparent: A node is more similar to its sibling rather than to its grandparent. To illustrate, *AnyWineColor* is grandparent of *Red*, and *Red* and *Rose* are siblings. Therefore, we anticipate that *Red* and *Rose* are more similar than *AnyWineColor* and *Red*.

The relative distance between nodes c_1 and c_2 is estimated in the following way. Starting from c_1, the tree is traversed to reach c_2. At each hop, the similarity decreases since the concepts are getting farther away from each other. However, based on our intuitions, not all hops decrease the similarity equally.

Let m represent the factor for hopping from a child to a parent and n represent the factor for hopping from a sibling to another sibling. Since hopping from a node to its grandparent counts as two parent hops, the discount factor of moving from a node to its grandparent is m^2. According to the above intuitions, our constants should be in the form $m > n > m^2$ where the value of m and n should be between zero and one.

Table 3 Sample similarity estimation over sample taxonomy

$Similarity(ReddishColor, Rose) = 1 * (2/3) = 0.6666667$
$Similarity(Red, Rose) = 1 * (4/7) = 0.5714286$
$Similarity(AnyWineColor, Rose) = 1 * (2/3)^2 = 0.44444445$
$Similarity(White, Rose) = 1 * (2/3) * (4/7) = 0.3809524$

Some similarity estimations related to the taxonomy in Fig. 2 are given in Table 3. In this example, m is taken as 2/3 and n is taken as 4/7.

For all semantic similarity metrics in our architecture, the taxonomy for attributes is held in the shared ontology. In order to evaluate the similarity of the attribute vector, we firstly estimate the similarity for each attribute one by one and take the average sum of these similarities. Since we expect each attribute to make an equal contribution for estimating similarity, average sum is used to calculate the similarity of vectors. If instated, minimum or maximum value of the vector values are used, the impact of some issues may be lost and only one issue may come into prominence.

3 Negotiation architecture

We are interested in service negotiation, in which the consumer initiates the negotiation with a particular request consistent with her preferences and the producer tries to meet consumer's needs by making alternative offers. Here, neither the producer nor the consumer know each other's preferences. However, both can try to learn each other's preferences so that they can negotiate more effectively. That is, if the producer has too many services that it can offer, proposing each service one by one will be time consuming for both parties. It would be much useful if the producer can understand the customer's needs and propose offers that respect both its and the customer's preferences. This is also beneficial if the producer cannot fulfill the customer's request. Rather than proposing all possible offers and failing after the last one, if the producer learns customer's needs, it can decide that the customer's needs cannot be satisfied early on.

Our main components are consumer and producer agents, which communicate with each other to perform negotiation over the service itself (content-oriented negotiation). Figure 3 depicts our architecture. The consumer agent represents the customer and hence has access to the preferences of the customer. The consumer agent generates requests in accordance with these preferences and negotiates with the producer based on these preferences. Similarly, the producer agent has access to the producer's inventory and knows which services are available or not. Producer's service inventory holds the information such as the content of available services, the amount of services and their utility value for the producer. This utility value is used in determining which service will be offered to the consumer if there exist more than one possibly acceptable services for the consumer. The service having higher utility is always preferred by the producer.

A shared ontology provides the necessary vocabulary and hence enables a common language for the agents. If the agents do not have a shared ontology, their negotiation requests and offers will not make sense to each other or will be partially understood by the other agent. Since our focus is on the negotiation and learning of preferences, we assume that a shared ontology exists. This ontology describes the content of the service. Further, since an ontology can represent concepts, their properties and their relationships semantically, the agents can

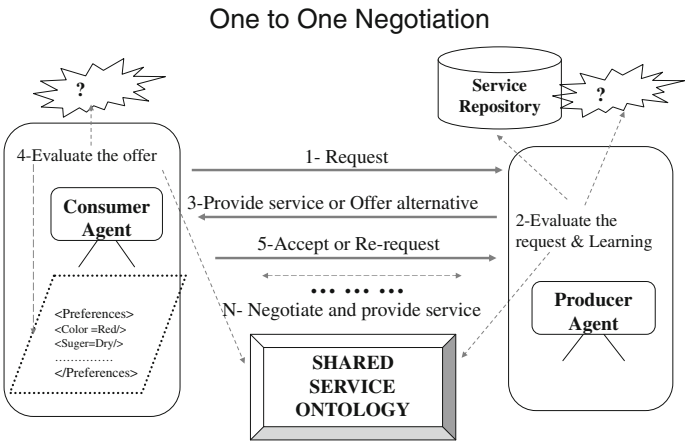

Fig. 3 Proposed negotiation architecture

reason the details of the service that is being negotiated. Since a service can be anything such as selling a car, reserving a hotel room, and so on, the architecture is independent of the ontology used. However, to make our discussion concrete, we use the well-known wine ontology [31] with some modification to illustrate our ideas and to test our system. The wine ontology describes different types of wine and includes attributes such as color, body, winery of the wine and so on. With this ontology, the service that is being negotiated between the consumer and the producer is that of selling wine.

The consumer agent initiates the negotiation with a service request, which is represented as a vector of attribute values. The consumer agent uses its preferences to generate this service request. In detail, it chooses a constraint from its disjunctive constraints randomly. It assigns the values, which are specified in this constraint and for other issues it chooses values randomly from their domain. For instance, consider that the consumer has the following preference: *{(?, Red, Dry)* and *(?, White, Sweet)}*. First, the consumer chooses one of the constraints randomly. If the consumer selects the first constraint, it will initialize the particular values specified in the constraint as *Red* and *Dry*. For the region issue, it will pick up one of the domain values randomly. A possible service request can be *(Chianti, Red, Dry)*.

If the producer has the requested wine, it provides this services and the negotiation finishes. Otherwise, the producer offers an alternative service from its inventory. In this phase, the producer tries to learn consumer's preferences from the bids exchanged during the negotiation. The producer considers both its available services and their utilities for the producer and consumer's learned preferences when deciding the alternative service. It does not offer a service, which is classified as a rejectable service by the learning algorithm. Among services classified as acceptable, producer chooses the service having the highest utility value for the producers. When the consumer receives a counter offer from the producer, it evaluates this offer in according to its preferences. If the offer satisfies consumer's preferences, then the negotiation will end up with a success. Otherwise, the customer generates a new request according to its preferences or stick to the previous request. This process will continue until some service is accepted by the consumer agent or all possible offers are put forward to the consumer by the producer. Of course, according to the agent's deadline, an agent may withdraw from the negotiation at any time.

One of the crucial challenges of the content-oriented negotiation is the automatic generation of counter offers by the service producer. When the producer constructs its offer, it

should consider three important things: the current request, consumer's preferences and the producer's available services. Both the consumer's current request and the producer's own available services are accessible by the producer. However, the consumer's preferences in most cases will not be available. Hence, the producer will have to understand the needs of the consumer from their interactions and generate a counter offer that is likely to be accepted by the consumer.

To generate the best offer, the producer agent uses its service repository and one of the inductive learning algorithm such as CEA, DCEA, RCEA, ID3 and Bayesian classifier. The service offering mechanism is the same for both the original CEA, DCEA and RCEA, but their methods for updating G and S are different.

The producer uses the hypotheses in G to filter out its stock. That is, if a service in a stock is not covered by G, then it is assumed that it will not be accepted by the consumer since it does not fit the modeled preferences. The producer assigns a utility value to each of its services and prefers to offer a service that is both an acceptable service as far as the consumer's preferences are concerned and a desired service whose utility is more than others for the producer. Among the services that are likely to be offered, an average similarity value is estimated with respect to the hypotheses in S. At the end, the most similar service is offered to the consumer.

If the producer learns the consumer's preferences with ID3, a similar mechanism is applied with two differences. First, since ID3 does not maintain G, the list of unaccepted services that are classified as negative by the decision tree are removed from the candidate service list. Second, the similarities of possible services are not measured with respect to S, but instead to all previously made requests. Note that ID3 is not an incremental algorithm, which means it requires all the training samples at the beginning of the training. To overcome this problem, the system keeps consumer's requests throughout the negotiation interaction as positive samples and all counter offers rejected by the consumer as negative examples. After each coming request, the decision tree is rebuilt.

Similarly, the producer using Bayesian eliminates the services that are classified as negative. After selecting the services having the highest utility for the producer, the most similar service to the positive sample set is offered by the producer.

4 Proposed learning algorithm (RCEA)

To learn the consumer's needs, we have developed Revisable Candidate Elimination Algorithm (RCEA), which is an incremental algorithm a la CEA in which the training samples become available only during the execution. A training sample x corresponds to a service request or a service offer and is a vector of attribute values such as $x = \{x[1], x[2], \ldots, x[m]\}$ where m is the number of attributes and $x[i]$ is the value of ith attribute. Possible domain values for each attribute are known and attributes can only be assigned values from their respective domain.

The consumer's requests are accepted as positive examples whereas the producer's counter offers that are rejected by the consumer are taken as negative examples. The concept to be learned should cover the positive samples but not cover the negatives. Using the learning algorithm, the producer decides which of its services would be more desirable for the consumer.

The main improvement to the original algorithm is that RCEA can *retract* its hypothesis about what it has learned as more interactions take place. Further, it uses an underlying ontology of service attributes for revising hypothesis as necessary. Since CEA does not support

learning disjunctives or making use of ontologies, we make the following changes to the algorithm.

First, according to CEA, all of the hypotheses in the most general set should cover the entire positive sample set. This rule prevents learning disjunctives since disjunctives are the union of more than one hypotheses and it cannot be covered by a single hypothesis with the condition of excluding negative samples. Therefore, in our learning algorithm we change this rule with that a positive sample should be covered by at least one of the hypotheses in the most general set. Furthermore, in some cases, a revision may be required when there is no more hypothesis in the most general set that is consistent with the incoming positive sample. In such a case, we need to add a new hypotheses covering this new positive sample while excluding all the negative samples. Hence, we require to keep the history of negative samples.

Second, when a positive sample comes, generalization of the specific set is performed in a controlled way with a threshold value, Θ. As far as disjunctive concepts are concerned, there should be more than one specific hypotheses in the most specific set. Deciding which hypothesis will be generalized is a complicated task. To do this, we estimate similarity of the current positive sample with respect to each hypothesis in the most specific set. The process of choosing the hypothesis that will be generalized uses this similarity information. Here, any similarity metric can be applied. During our study, we use the RP similarity (Sect. 2.7), which uses semantic information such as subsumption relations. At the end, the algorithm only generalizes the selected hypothesis.

Third, different from CEA, the generalization of a hypothesis is now controlled by another threshold value, Φ. This threshold value determines whether a generalization will be performed for each attribute. Generalization of hypothesis is different for each attribute in the hypothesis, in fact it depends on the property of the attributes. If there is an ontological information such that a hierarchy exists on the values of the attribute, the generalization depends on the ratio of the covered branches in the hierarchical tree. Otherwise, we apply a heuristic that favors generalization to *Thing* (?). Our heuristic is that we have a higher probability when more different values for the attribute exist. We estimate the dissimilarity by using semantic similarity that can be also fuzzy similarity. If the dissimilarity is higher than a predefined threshold value, we generalize it to ?. For instance, if the accepted values for the sweetness for the user are *Dry* and *Sweet*, it has more probability to generalize *Thing* (?) concept rather than *Dry* and *OffDry*.

4.1 Components of RCEA

Revised Candidate Elimination Algorithm (RCEA) manipulates four important sets.

Most specific set (S) contains the most specific hypotheses that cover the positive examples minimally. Formally, $S = \{H_1^S, H_2^S, \ldots, H_n^S\}$ where n is the number of specific hypotheses in S and each specific hypothesis is in the form of $H_j^S = \{H_j^S[1], H_j^S[2], \ldots, H_j^S[m]\}$ where m is the number of attributes and $H_j^S[y]$ is a vector of acceptable values for the y^{th} attribute.

Most general set (G) contains the most general hypotheses consistent with the positive samples. Formally, $G = \{H_1^G, H_2^G, \ldots, H_k^G\}$ where each hypothesis is of the form $H_j^G = \{H_j^G[1], H_j^G[2], \ldots, H_j^G[m]\}$ where m is the number of attributes and $H_j^G[y]$ is a value for the y^{th} attribute.

Negative Sample Set (N) contains the negative examples.

Positive Sample Set (P) includes the positive examples.

These sets are important in generating offers to the customers. If a service is not in G, it means that the service cannot be acceptable for the consumer, since G holds the most general hypotheses about the customer's service preferences. However, many services may fall in G, then the question is which one to offer. Among possibly acceptable services, finding the most satisfactory services for the consumer is crucial. By finding the most similar services to the most specific set (S) involving minimally acceptable services, the producer can find the most satisfactory services. Note that, here, we are not interested in the convergence of these sets. They are used independently: G for filtering unacceptable services and S for generating an alternative offer.

4.2 Properties of attributes

Attributes represent the components constructing the service in negotiation but it can be a part of any concept in other domains. Each attribute is defined in an ontology in which the domain information for these attributes and some relations associated with these attributes such as a hierarchy are kept. Reasoning on these relations would be useful in generalization and specialization of the hypotheses.

For a hypothesis to cover the sample, all attribute values in the hypothesis should be consistent with those of the sample. Consistency can be decided by subsumption relation. For each attribute x and y, $doesCover(x, y)$ means that the concept of x is an ancestor of y or $x = y$ in the case that the attribute has a hierarchy. Otherwise, $doesCover(x, y)$ means that the $x =?$ or $x = y$. Note that ? is a special value for an attribute that means any value is acceptable and it is at the top of the hierarchy.

In some cases, the generalization of a specific hypothesis is required and performed for each attribute of the hypothesis. The generalization of attributes depends on the ontological information. Some attributes have a hierarchical classification whereas some do not. According to whether the attribute has hierarchical information or not, a specific attribute is generalized.

- If an attribute has a hierarchical structure such as *WineRegion* as shown in Fig. 4, the algorithm generalizes the attribute value to the nearest common parent of the values for that attribute under a special condition depending on a threshold value. Here, the proportion of the number of values that the hypothesis involves to the number of values that the nearest common parent concept covers is estimated. For example, if we have *AnjouRegion* and *BordeuxRegion* values, this proportion for generalizing this feature to *FrenchRegion* is equal to 3/15. After estimating the proportion, it is compared with the predefined threshold value, Φ. If the proportion is greater than this threshold, the attribute will generalize to the nearest common parent. The reason for the nearest common parent is that minimal generalization of the hypothesis is desired.
- If the attribute does not have any hierarchical structure, we estimate the average similarity of the values that we have. At this point, our heuristic tells that if the attribute has more dissimilar values, we have more probability to generalize this attribute to *Thing*, ? which means every value is accepted for that attribute. For instance, for *Body* attribute we have three values: *Light*, *Medium* and *Full*. Table 4 shows the semantic similarities for these values. Note that these values should be assigned by a domain expert. However, in this study we assign these values in a way that there are three levels (light, medium, full) so we divide one by three ($1/3 = 0.3$). We give this ratio to the most different values (light and full) and for other values we add this to 0.3 so that the similarity medium and light becomes 0.6. Of course, we can use also fuzzy similarities [11] instead of semantic

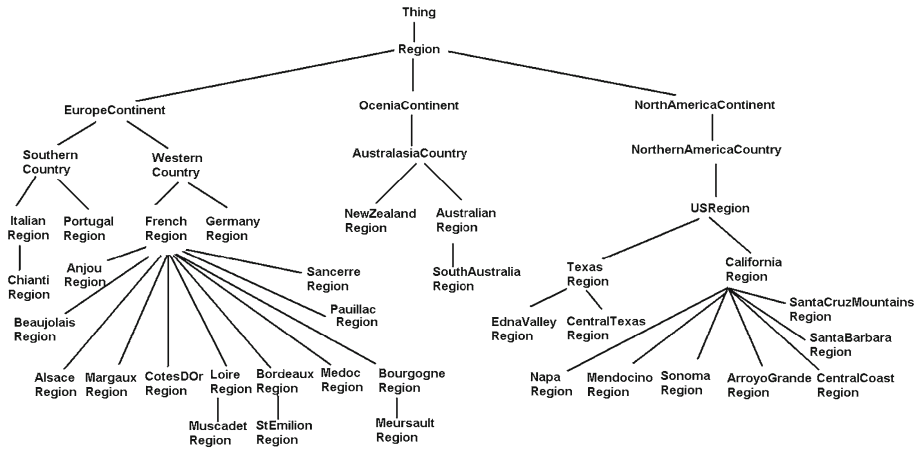

Fig. 4 Wine region hierarchy

Table 4 Semantic similarities for body

Body 1	Body 2	Similarity
Light	Full	0.3
Medium	Light	0.6
Full	Medium	0.6

similarities that we assign. According to our heuristic, if we have light and full, we are more likely to generalize to ? than the case when we have light and medium.

4.3 Learning algorithm

Algorithm 1: Revisable Candidate Elimination Algorithm (RCEA)[x]

1: **if** *x is positive* **then**
2: $P \leftarrow P + \{x\}$
3: $H_v^S \leftarrow$ updateSpecificSetForPositiveSample(x)
4: updateGeneralSetForPositiveSample(x,H_v^S)
5: **else**
6: $N \leftarrow N + \{x\}$
7: updateGeneralSetForNegativeSample(x)
8: removeLessGeneralFromGeneralSet()
9: updateSpecificSetForNegativeSample(x)
10: **end**

Each training sample is given as an input to the system. For each sample, both the most general and most specific sets are modified. Modifications depend on the type (positive or negative) of the sample. Algorithm 1 shows the general flow of the training process. If the sample is positive, it is added to the positive set, P (Line 2); otherwise it is added to the negative set, N (Line 6). Correspondingly, the most specific set (S) and the most general set (G) are updated. Note that in our algorithms, the following notation is used.

- x: current training sample
- H_i^S: ith hypothesis in the most specific set, S
- H_i^G:ith hypothesis in the most specific set, G
- N_i: ith negative sample in the negative set, N
- P_i: ith positive sample in the positive set, P
- $Sim_z(x, y)$: similarity of x to the hypothesis y using the similarity metric z
- $max_\Theta(Sim_z(x, H_j^S))$: returns the similarity values and the indices of the hypotheses in S, which are similar to the current sample by the similarity greater than Θ in descending order
- $x[i]$: the value of ith attribute of the current sample
- $Attr_i$: ith attribute
- ?: a special value for an attribute that means any value is acceptable
- $H_?$: the most general hypothesis consisting of "?" for each attribute

updateSpecificSetForPositiveSample: According to Algorithm 1, if the sample is positive, the algorithm updates the specific set to cover the current positive example (Line 3). The specific set should include at least one hypothesis that covers this positive sample. That is, one hypothesis in S needs to be chosen and generalized to cover this sample. The hypothesis that is chosen is the most similar hypothesis to the current sample. This is done to ensure that S is generalized as minimally as possible. The details of this process are explained in Algorithm 2.

Algorithm 2: updateSpecificSetForPositiveSample[x]

```
1:  found ← FALSE
2:  while !found do
3:      (maxSim, j) ← max_θ(Sim_z(x, H_j^S))
4:  if j ≠ null then
5:      H_new^S ← generalizeSpecificHypothesis(j, x)
6:      if doesCoverNegatives(H_new^S) ≡ FALSE then
7:          H_j^S ← H_new^S
8:          found ← TRUE
9:  else
10:     H_{n+1}^S ← {x}
11: end
12: end
```

That is, for each specific hypothesis, a similarity is estimated and the hypotheses whose similarity with the current sample is greater than Θ are sorted according to their similarity value in a descending order (Line 3). Starting with the first hypothesis in this list, we generalize the specific hypothesis to cover the new example (Line 5). Afterward, the algorithm tests whether the extended specific hypothesis covers any negative example in the negative sample set (Line 6). If this hypothesis covers a negative example, the algorithm interprets that this extension is invalid so it passes to the next specific hypothesis in the ordered list to generalize in order to cover the new positive sample. This process continues until the extended specific hypothesis does not cover any negative examples or the list is exhausted. If a hypothesis with a valid extension is found, the original hypothesis is replaced with its extended version (Line 7). On the other hand, if the list is exhausted or there is no hypothesis whose similarity is

greater than Θ, the new positive example is added as a separate hypothesis into most specific set (Line 10).

– *Complexity Analysis (Algorithm 2):* Let m denote the number of hypotheses in S and k denote the number of attributes. The computational complexity of estimating similarity of the positive example to each specific hypothesis is $O(k)$ and sorting them in descending order is $O(mlog\ m)$, yielding $O(\ kmlogm)$. Note that it is enough to do this computation once and to use this ordered list with estimated similarity values in the while loop (Line 3). If it is possible to generalize a particular specific hypothesis to cover the given sample, we do so and check whether the hypothesis covers any negative examples. The complexity of the generalization process is $O(kh)$ where h denotes the height of the hierarchy tree for a given issue (see *Complexity Analysis for (Algorithm 3*) for more detail). The computational complexity of checking all negative examples is $O(kn)$ where n is the number of negative examples. Overall this gives $O(k(h+n))$. We may end up repeating this procedure m times; i.e., until we find a specific hypothesis that does not cover the negative examples or until S is exhausted. The time complexity then is $O(\ km(h+n))$. Overall computational complexity would then be $O(\ km(log\ m+h+n))$.

generalizeSpecificHypothesis: $Generalize(H_j^S, x)$ results in H_i^S such that $H_i^S \supset H_j^S$ and $H_i^S \supset x$. Algorithm 3 indicates how the jth specific hypothesis is generalized when the positive sample x is received. For each attribute, we check if the kth attribute of the specific hypothesis covers the kth attribute of the current sample x (Line 2). If the value of the current positive sample is not covered, the kth attribute value is generalized.

Algorithm 3: generalizeSpecificHypothesis[j,x]

```
1:  for k ← 0 to Attr.size do
2:    if !doesCover(H_j^S[k], x[k]) then
3:      if hasHierarchy(Attr_k) then
4:        H_j^S[k] ← H_j^S[k] + x[k]
5:        comParent ← findCommonParent()
6:        proportion ← |H_j^S[k]|/|comParent|
7:        if Φ ≺ proportion then
8:          H_j^S[k] ← comParent
9:      else
10:       if Ω < (1 − Sim_z(x[k], H_j^S[k])) then
11:         H_j^S[k] ←?
12:       else
13:         H_j^S[k] ← H_j^S[k] + x[k]
14:       end
15:   end
16: end
```

This generalization for an attribute having a hierarchy is performed as explained in Sect. 4.2 (Lines 3–8). If there is no hierarchy for this attribute, the dissimilarity $(1-Sim_z(x[k],\ H_j^S[k]))$ is estimated between attribute values of hypothesis and current sample (Line 10). If this dissimilarity is greater than the threshold, the attribute will be generalized to ? (Line 11). Note

Table 5 Interaction between consumer and producer

Request or offer	Region	Color	Sugar
First request (R_1):	Chianti	Rose	OffDry
Current Version Space:	$S_0 = \{R_1\}$ $G_0 = \{(?, ?, ?)\}$		
First counter offer:	Chianti	Rose	Sweet
Current Version Space:	$S_1 = \{R_1\}$ G_1={(?, ?, OffDry)}		
Second request (R_2):	California	White	Sweet
Current Version Space:	$S_2 = \{R_1, R_2\}$ G_2={(?, ?, OffDry), (NorthAmerica, ?,?) , (?, White, ?) }		
Second counter offer:	US	Rose	OffDry
Current Version Space:	$S_3 = \{R_1, R_2\}$ G_3={(?, White, ?), (NorthAmerica, ?, Sweet), (California, ?, ?), (Europe, ?, OffDry) }		
Third request (R_3)	Italian	Red	Dry
Current Version Space:	S_4= {(Italian, Reddish, [OffDry,Dry]), R_2} $G_4 = G_3$		

that the dissimilarity shows the diversity of values that the attribute can have. This means that this attribute can be generalized to ?, which involves all values for that attribute. Otherwise, the value of x is added to the hypothesis (Line 13). Example 2 illustrates this process. Note that during the following examples the interaction specified in Table 5 will be used.

Example 2 Because of the limited space, to demonstrate our examples we use only three attributes instead of seven: Region, Color and Sugar. However, in our experimental results in Sect. 5, the all seven attributes are used. The former two attributes have a hierarchy and the last attribute can take three possible values: *Dry*, *OffDry* and *Sweet*. The color attribute can be *Red*, *Rose*, *White* and *Reddish*, which is the parent of *Rose* and *Red*. There are 41 possible values for the region attribute and its hierarchy is more complicated.

According to Table 5, the consumer agent asks for *(Chianti, Rose, OffDry)* as a first request. Since there is no hypothesis in S, the first request is added as a hypothesis into S. In the second turn, the consumer requests *(California, White, Sweet)*. The similarity between this request and the hypothesis in S is estimated as 0.28 according to RP similarity metric. Assume that the threshold value is equal to 0.5 during the learning process. Because the similarity is less than the threshold value (θ), the second request is added into system as a separate hypothesis. Next time, the consumer requests *(Italian, Red, Dry)*. The similarity for the first hypothesis in S is 0.68. Therefore, the algorithm first tries to generalize the first hypothesis and the extended hypothesis becomes *{(Italian), (Reddish), (OffDry, Dry)}*. The system checks whether any negative sample (producer's counter offer rejected by consumer) is covered by this generalized hypothesis. If it does not cover any negative sample, the generalization of specific set is completed. Otherwise, the algorithm undoes the generalization of the first hypothesis. Note that the nearest common parent for *Red* and *Rose* is *Reddish* and *Italian* subsumes *Chianti*. The dissimilarity of *OffDry* and *Dry* is 0.2. Since it is less than threshold Ω, for this attribute, generalization is not performed.

– *Complexity Analysis (Algorithm 3):* If the issue values are structured in a hierarchy, we need to find the nearest common ancestor of two values to generalize. Let k denote the number of attributes and h the height of the hierarchy for a given issue. The computational complexity of finding the common ancestor of two nodes is equal to $O(h)$ at worst

case. If there is no hierarchy for that issue, we need to estimate the dissimilarity of the issues whose complexity is $O(1)$. Since we perform the generalization for each issue, the complexity would be $O(kh)$ at worst case.

updateGeneralSetForPositiveSample: Since our learning algorithm allows disjunctives, there may be a variety of general hypotheses consistent with different positive samples. Unlike CEA, our algorithm does not eliminate any general hypotheses not covering the current positive sample. Instead, positive sample should be covered by at least one hypothesis in the general set. If none of the hypotheses cover the new positive sample, new general hypotheses covering this positive sample but not covering any negative samples are added into the most general set (Line 4 in Algorithm 1). This revision process is a new operation for Version Space and does not exist in CEA. Using this operation, our algorithm supports learning disjunctive concepts. Algorithm 4 shows the general picture for updating general set when a positive sample comes.

Algorithm 4: updateGeneralSetForPositiveSample[x, H_v^S]

```
1: if needRevision(x) ≡ TRUE then
2:     reviseGeneralSet(H_v^S)
3:     removeLessGeneralFromGeneralSet()
```

reviseGeneralSet: As specified in Algorithm 4, the algorithm first checks whether there is a need for revision. If so, new general hypotheses that cover the positive sample but not cover the previous negative samples are generated from the most general hypothesis by the help of the most similar specific hypothesis to current sample.

Algorithm 5 involves how the revision process is performed. This procedure takes the specific hypothesis (H_v^S) covering the current positive sample. First, the most general hypothesis consisting of ? is added into possible hypothesis set (Line 1). For each negative sample in the negative sample set (N), all possible hypotheses in possible hypothesis set is checked for covering one of the negative samples (Line 5). For each possible hypothesis covering the values of the negative sample, the values of the specific hypothesis, H_v^S, are used for specializing the possible hypotheses (Line 10). If there is a hierarchy for that attribute of the specific hypothesis, the most general parent concept in the hierarchy not covering the negative example is found (Line 12). Then, by setting this value to the current possible hypothesis a new hypothesis is created and added as a separate hypothesis into the possible hypothesis set (Lines 13–14). If there is no hierarchy for this attribute, the value of specific hypothesis is used directly (Lines 16–18). Example 3 explains this process in detail.

Example 3 In the second turn in Table 5, the consumer makes a request as *(California, White, Sweet)*. Since the current most general set G, *(?, ?, OffDry)* is consistent with only the wines whose sugar level is *OffDry*, the current request is not accepted by the current G. Therefore, revision is needed. According to our revision algorithm, first the most general hypothesis, *(?, ?, ?)* is generated as a possible hypothesis. In a loop, the set of possible hypotheses is checked to see whether the hypotheses in this set covers any negative sample. If a hypothesis covers a negative sample, the algorithm compares each attribute of the negative sample with the specific hypothesis involving the current positive sample. In detail, our negative sample set only includes *(Chianti, Rose, Sweet)*. The specific hypothesis involving the current positive sample is *(California, White, Sweet)*. The first two attributes are different. The possible hypotheses set includes only one hypothesis, *(?, ?, ?)*. Two new possible hypotheses may

Algorithm 5: reviseGeneralSet[H_v^S]

```
1:  HypoSet[0] ← H?
2:  for i ← 0 to |N| do
3:    for j ← 0 to |HypoSet| do
4:      H_t ← HypoSet[j]
5:      if doesCover(H_t, N_i) then
6:        HypoSet ← HypoSet − {HypoSet[j]}
7:        for k ← 0 to Attr.size do
8:          if doesCover(H_t[k], N_i[k]) then
9:            for r ← 0 to |H_v^S[k]| do
10:             if !doesCover(H_v^S[k][r], N_i[k]) then
11:               if hasHierarchy(Attr_k) then
12:                 g ← ParentNotCover(N_i[k])
13:                 H_t[k] ← g
14:                 HypoSet ← HypoSet + {H_t}
15:               else
16:                 H_tnew ← H_t
17:                 H_tnew[k] ← H_v^S[r]
18:                 HypoSet ← HypoSet + {H_tnew}
19:               end
20:           end
21:       end
22:   end
23: end
```

be generated as *(California,* ?, ?*)* and *(*?, *White,*?*)*. Since the hypotheses in G should be as general as possible, we make another revision. By using hierarchical information kept in the ontology, we can find the most general parent of this value not involving the negative value *Chianti*. Since *NorthAmerica* is the most general concept subsuming *California*, the new possible hypothesis becomes *(NorthAmerica,* ?, ?*)* and *(*?, *White,* ?*)*. If newly constructed possible hypotheses cover any negative samples, they are modified not to cover negatives any more. As a result, these two newly generated hypotheses *(NorthAmerica,* ?, ?*)* and *(*?, *White,* ?*)* are directly added into the most general set since they do not cover any negative samples.

– *Complexity Analysis (Algorithm 5):* The complexity of checking whether a hypothesis covers a negative example is equal to the number of attributes, $O(k)$. In a loop, each candidate general hypothesis in $HypoSet$ is checked to see whether it covers any negative example in N. If it does, we generate possible specialization of that hypothesis in a way that it does not cover any negative example but covers the given specific hypothesis. To do this, for each candidate hypothesis in $HypoSet$, we compare the values of specific hypothesis with that of the current negative example for each attribute (Line 10). At worst case, the cost of specializing a single general hypothesis and generating new hypotheses not covering a negative example is $O(pk)$ where p denotes the maximum number of possible values for an attribute in the system. Note that $HypoSet$ includes only the most general hypothesis consisting of ? at the beginning and it grows up with respect to the content of the negatives and the given specific hypotheses. At worst case, we specialize a candidate hypothesis and generate (pk) different hypotheses. The computational complexity of specializing the hypothesis in $HypoSet$ would be $O(p^2k^2)$ if we assume that we have pk hypotheses in $HypoSet$. Note that the size of $HypoSet$ is dynamic and may

vary for each run. Since we traverse all of the negative examples in N to check whether any candidate general hypotheses covers any of them, the overall complexity would be $O(p^2k^2n)$, which means that the revision process will grow polynomially with the number of attributes, the number of values that a specific hypothesis contains for each issue, and the number of negative samples.

updateGeneralSetForNegativeSample: The aim of this method is to specialize the hypotheses in G, which cover the negative example, in a minimal fashion. Minimal specialization is crucial since it is desired that the hypotheses remain as general as possible. Therefore, Algorithm 6 checks all the hypotheses in the most general set to see if they cover the negative sample (Line 2). Each hypothesis covering the negative sample is backed up as an abandoned hypothesis (Line 3) since the information kept in those hypothesis should not be lost before removing them. Then, these hypotheses are removed from the most general set (Line 4). By only taking the hypotheses in the most specific set that are covered by the abandoned hypotheses (Line 6) into account, a minimal specification of the abandoned hypotheses that do not cover the negative sample are generated. To accomplish this, the algorithm compares the value of each attribute of specific hypothesis with that of the negative sample (Line 11 & Line 16). If the values are different and there is no hierarchy for the attribute, the value of attribute of the specific hypothesis is replaced in the abandoned hypothesis (Lines 17–18). If the values are different and there is a hierarchy for that attribute, the most general common parent of the value of the specific hypothesis, which does not cover the negative sample, is found (Line 12) and this value is replaced in the abandoned hypothesis (Lines 13–14). Example 4 shows how this process is performed.

Example 4 As specified in Table 5, at first G is equal to the most general hypothesis, *(?, ?, ?)*. After first negative example *(Chianti, Rose, Sweet)*, there is a need for specialization of hypotheses in G to make them not cover the negative samples. To accomplish this, for each hypothesis in G covering negatives, we use the hypotheses in S. Some hypotheses in G may not cover some particular hypothesis in S. Therefore, when updating the hypothesis in G, we should only consider the related hypotheses in S. In this example, S includes only *(Chianti, Rose, OffDry)* and this is covered by *(?, ?, ?)*. Now, the algorithm compares the values of the attributes in specific hypothesis with those of the negative sample one by one and it selects the attributes whose values are different. For example, the sweetness degree in specific hypothesis is *OffDry* whereas that for negative sample is *Sweet*. From this information, the algorithm modifies general hypothesis as *(?, ?, OffDry)*. Since sugar level does not have hierarchy, we modify the general hypothesis directly. In the case of hierarchy, we find the most general concept that does not cover the value of the negative sample.

– *Complexity Analysis (Algorithm 6):* We traverse each general hypothesis and check if it covers the given negative example, yielding a time complexity of $O(tk)$ where t is the number of general hypotheses in G and k is the number of attributes that a hypothesis involves. Next, for each hypothesis covering the given example (maximum is t), we compare the values of the specific hypothesis to those of the negative example for each attribute. If the values are different, we specialize the current general hypothesis. Let p denote the number of possible values for an attribute. The complexity of the comparison and specialization process is $O(pk)$. We repeat this process for each specific hypothesis, yielding $O(pkm)$ (where m is the number of specific hypotheses in S), yielding an overall complexity of $O(pkmt)$. Since the complexity of the first part is insignificant ($O(tk)$), we conclude that the overall complexity of updating general set is $O(pkmt)$ at worst case.

Algorithm 6: updateGeneralSetForNegativeSample[x]

```
for i ← 0 to |G| do
  if doesCover(H_i^G, x) then
    abandoned ← H_i^G
    G ← G − {H_i^G}
    for j ← 0 to |S| do
      if doesCover(abandoned, H_j^S) then
        for k ← 0 to Attr.size do
          for t ← 0 to |H_j^S[k]| do
            temp ← abandoned
            if hasHierarchy(Attr_k) then
              if !doesCover(H_j^S[k][t], x[k]) then
                gnew ← ParentNotCover(x[k])
                temp[k] ← gnew
                G ← G + {temp}
            else
              if !doesCover(H_j^S[k][t], x[k]) then
                temp[k] ← H_j^S[k][t]
                G ← G + {temp}
            end
          end
        end
      end
    end
  end
end
```

removeLessGeneralFromGeneralSet: If there are some hypotheses in G less general than the other hypotheses in G, these are removed from the system. Because, the aim is to keep the most general hypotheses in G. $LessGeneral(H_i, H_j)$ means that for each k, $H_j[k] \supseteq H_i[k]$ and $H_i \neq H_j$. To achieve this, the hierarchical information can be used. In the hierarchy, a child concept is less general than its ancestors. If there is no hierarchy for that attribute, any value is less general than ?.

updateSpecificSetForNegativeSample: If there are hypotheses in the most specific set that cover a negative sample, they are removed. Since each positive sample should be covered by at least one of the hypotheses in S, each positive sample in P is checked whether it is covered by S. If none of the specific hypotheses in S covers a positive sample, this positive sample is added as a separate hypothesis to S. Consequently, each positive sample in P will be covered by the most specific set.

5 Evaluation

To evaluate our algorithm, we construct an environment in which one producer tries to negotiate with a consumer on the service of wine selling as explained before. Our test environment is written in Java and Jena [15] is used as ontology reasoner.

We could not compare our algorithm directly with the existing approaches in negotiation in which preferences are represented as utility functions because of diversity in settings. First, our system requires a service ontology holding semantic similarities and hierarchical

information about the issue domain whereas other approaches do not use a service ontology. Second, some studies need additional knowledge about the domain. For instance, fuzzy similarities are needed for [11] where we use semantic similarities in our work. Third, in contrast to other approaches ([14,19]), in our setting the producer agent has a service repository and it can only generate a counter offer from its available services in its repository. However, other approaches usually do not consider a limited service repository as we did.

We compare the performance of our learning algorithm with other four alternatives, namely Candidate Elimination Algorithm (CEA), Disjunctive Candidate Elimination Algorithm (DCEA), ID3 and Bayesian Classifier. The comparisons are made such that the same consumer agent negotiates with five producers that have the same service inventory but employ a different learning algorithm to learn the consumer's preferences. After each request and counter offer, these producers train their own learning element accordingly.

If the producer uses one of the Version Space based learning algorithms such as CEA, DCEA and RCEA, it uses G to filter out its service stock. Note that if the service is not covered by G, this means that service will possibly be rejected by the consumer. After filtering available services, the producer chooses the service whose utility is the highest for the producer. Among these services, an average similarity value is estimated with respect to the hypotheses in S because S represents the consumer's predicted preferences. At the end, the most similar service whose utility for the producer is the highest, is offered to the consumer. Consequently, the producer offers a service that is beneficial for both the consumer and the producer agent.

Similarly, the producer using ID3 and Bayesian eliminates services that are classified as negative. After selecting the services having the highest utility for the producer, the producer offers the service that is most similar to the positive sample set. In our settings, RP semantic similarity metric (Sect. 2.7) is used as a similarity metric.

As far as the performance of the negotiation is concerned, the number of interactions between the consumer and the producer is the main factor. Obviously, completing the negotiation in as few interactions as possible is desirable for both parties. The number of interactions depends on the producer type, producer's stock, the preference of the customer and the order and content of consumer's request. In our experiments, in order to analyze the effects of these factors and the performance of the producer agent's learning algorithm, simulation of negotiation is executed for different combinations of these parameters.

The producer's available services and the consumer's order of requests affect the negotiation time. To compare different producers fairly, 100 different producer stocks are generated randomly. Each stock contains 50 different wine services. The utility of these services varies between one and five and are equally distributed in the stock. Further, for each scenario the negotiation process is repeated 10 times varying the request orders. To construct the consumer profiles (preferences), we asked 20 people (students and faculty from the Department of Computer Engineering at Bogazici University) to create consumer profiles. First two preference profiles only consist of conjunctives but other 18 preferences are in the form of both conjunctives and disjunctives. Table 6 shows five representative preference profiles. The preferences of the consumers are taken from the user by using a simple interface. The output of this interface is a preference file including the preference information and it is used only by the consumer agent. That is, the producer agents cannot access this file.

The satisfiability of the consumer's preferences over the stocks also varies. For example, the second preference is not satisfiable in 55 of the 100 stocks. In the remaining 45 stocks, the number of desired services is one. For the seventh preference, all stocks have some services consistent with user preferences. On average, 16.62 of the 50 services in a stock are desirable

Table 6 Selected customer preferences selected customer preferences

Id	Preference—[Sugar, Flavor, Body, Color, Region, Winery, Grape]—
1	[OffDry, ?, ?, White, ?, Cheap, ?]
2	[?, Moderate, ?, ?, US, ?, WhiteGrape]
4	[?, Moderate, ?, White, US, ?, ?] or [?, Delicate, ?, Red, Italian, ?, ?]
6	[Dry, Moderate, Medium, ?, (Italian, Portugal), ?, ?] or [Dry, ?, Light, Rose, ?, ?, ?]
7	[Sweet, ?, Light, ?, ?, ?, ?] or [Dry, Strong, ?, ?, ?, ?, ?] or [OffDry, ?, ?, ?, ?, Expensive, RedGrape]

for the customer. Compared to the second preference, it is highly possible to find a desired service.

We first test the performance of the producers using Version Space learning algorithms (RCEA, CEA and DCEA). Then, we compare our proposed algorithm (RCEA) with adaptations of other well-known classifiers, mainly ID3 and Bayesian.

5.1 Case 1: Comparing RCEA, CEA and DCEA

We experiment with three separate producers: Producer with Candidate Elimination Algorithm (CEA), Producer with Disjunctive (DCEA) and Producer with Revisable Candidate Elimination Algorithm (RCEA). We have ten runs for 100 different stocks, resulting in 1,000 negotiations for each preference profile. In each of the ten runs, the consumer starts the negotiation with a random request that is compatible with her preferences. We repeat these experiments for 20 different consumer profiles. We evaluate the behavior of the producers using different success and performance criteria.

5.1.1 Success of producers: RCEA, CEA and DCEA

We first study the number of negotiations that are finalized successfully. Table 7 shows the number of successful negotiations for each preference (out of 1,000 trials). The last column shows the number of trials in which success was possible; i.e., there was a possible service that could satisfy the consumer. The figures show that DCEA always carries out a successful negotiation if there is a satisfying service. The reason for these perfect results is that DCEA cannot filter the services except for the counter offers rejected by the consumer. Therefore, it never gives up offering services unless there are no remaining service in its stock or its offer is accepted by the consumer. On the other hand, RCEA's success is close but may miss up to 8.2% of the time. CEA, on the other hand, has varying performance and may miss up to 85% of the possible services. This is mainly due to the fact that CEA cannot handle disjunctives properly.

We also apply the Friedman statistical test [12], which is often used for comparison of multiple classifiers in order to see whether the success of producers is statistically significantly different from each other. In Friedman test, the algorithms are ranked for each data set separately from the best performing algorithm to the worst performing algorithm [10]. According to the average ranks of the algorithms, this test compares the multiple classifiers. In our experiments, each negotiation is considered as a different data set because different training sets are obtained in each negotiation. We take alpha as 0.01 where alpha is the significance level of the statistical test. According to this statistical test, there is no statistically significant difference between the number of successful negotiations of RCEA and that of

Table 7 Case-1: Number of negotiations that end with consensus

	# of success			Poss. Successful
Preference	RCEA	CEA	DCEA	Negotiations
1	970	970	970	970
2	450	450	450	450
3	998	476	1,000	1,000
4	786	119	790	790
5	1,000	694	1,000	1,000
6	894	349	920	920
7	1,000	675	1,000	1,000
8	1,000	562	1,000	1,000
9	968	201	970	970
10	960	193	990	990
11	1,000	324	1,000	1,000
12	1,000	331	1,000	1,000
13	1,000	335	1,000	1,000
14	1,000	579	1,000	1,000
15	999	336	1,000	1,000
16	1,000	615	1,000	1,000
17	529	140	530	530
18	1,000	616	1,000	1,000
19	989	255	990	990
20	817	306	890	890
Avg.:	918.0	426.3	925.0	925.0

DCEA under 99 per cent confidence level. However, there is a statistically significant difference between those and that of the CEA. Thus, it can be said that RCEA is successful as DCEA on average.

Recall that, for each of the 100 stocks, we test the performance with ten different consumer request orders. It is also important for a producer to negotiate consistently in all ten orders. That is, a producer that negotiates well with a certain ordering of requests but not too well when the ordering changes is really not useful, since nothing can be guaranteed about its negotiation. This is what we investigate next. We let a producer win a point if it completes all ten trials successfully. This is estimated for all stocks and summed up at the end. Therefore, it indicates the consistency of producer's overall success. Since the consistency performance of the algorithms can be at most the maximum consistency performance, we find the ratio of each producer to the maximum and take the average of these ratios to see the average consistency performance of each producer. According to Table 8, DCEA and RCEA have the most consistent performance whereas the performance of CEA is much below. For the first two preference profiles, the consistency performance of CEA is the same with those of DCEA and RCEA because these preferences include only conjunctives.

When we apply the Friedman test to these results, it can be said that there is no significant difference between the consistency performance of RCEA and DCEA but there is a statistically significant difference between those and the consistency performance of CEA under 99 per cent confidence level.

Table 8 Case-1: Consistency performance of the producers

Preference	RCEA	CEA	DCEA	Max*
1	97	97	97	97
2	45	45	45	45
3	98	4	100	100
4	75	0	79	79
5	100	8	100	100
6	74	6	92	92
7	100	6	100	100
8	100	7	100	100
9	95	0	97	97
10	74	1	99	99
11	100	0	100	100
12	100	0	100	100
13	100	1	100	100
14	100	3	100	100
15	99	2	100	100
16	100	6	100	100
17	52	0	53	53
18	100	4	100	100
19	98	1	99	99
20	59	8	89	89
Avg.:	95.42	12.93	100.00	100.00

* The maximum consistency performance value

5.1.2 Performance of producers: RCEA, CEA and DCEA

The results presented in Sect. 5.1.1 show that CEA fails in significant number of negotiations whereas both DCEA and RCEA are successful in most of the negotiations. The result is exemplified if we also consider the consistency. Between DCEA and RCEA, we see that DCEA is slightly more successful in providing a service that will satisfy the consumer. Now, we study the performance of these algorithms.

For performance estimation, the metric is the average number of interactions for successful termination in the case that all producers have a success at the same run. We select a subset of the negotiations in which all three producers have been successful and study the average number of interactions needed for termination. Table 9 shows that when CEA is successful in negotiation, then the number of interactions it requires is quite low. (Note that CEA is unsuccessful in many interactions as seen in Table 7). This shows that CEA either completes a negotiation quickly or fails (mostly when the preference is in the form of disjunctives). When RCEA and DCEA are compared, we see that DCEA needs more interactions to end a negotiation successfully. The reason stems from the fact that DCEA does not filter the candidate services in accordance with the consumer's preferences as well as RCEA does. Hence, it requires more time to complete the negotiation.

Note that in 8,525 out of 20,000 negotiation runs all of the producers (RCEA, CEA and DCEA) are successful. We apply Friedman test to the results of the 8,525 runs. Under 99 per

Table 9 Case-1: Avg. # of interactions when all are successful

Preference	RCEA	CEA	DCEA
1	5.14	3.13	11.62
2	7.42	3.98	19.63
3	2.50	1.76	2.61
4	7.36	2.40	9.21
5	1.93	1.56	2.07
6	4.25	2.35	5.56
7	1.63	1.40	1.64
8	1.76	1.41	1.66
9	4.64	2.12	5.26
10	4.22	1.98	5.66
11	2.33	1.57	2.38
12	2.84	1.74	2.95
13	2.16	1.50	2.16
14	2.26	1.64	2.37
15	2.98	1.70	3.63
16	1.49	1.36	1.50
17	5.86	2.09	9.14
18	1.79	1.49	1.76
19	3.17	1.85	3.73
20	5.42	2.56	9.23
Avg.:	3.18	1.97	4.94
StDev.:	4.02	1.34	8.20

cent confidence level, the results of RCEA, CEA and DCEA are statistically significantly different to each other and according to the number of interactions when all are successful, the ordering can be represented as $CEA \prec RCEA \prec DCEA$. This means that RCEA negotiates faster than DCEA and when all producers have a success, CEA negotiates faster than RCEA and DCEA. Note that when the preferences are in the form of both conjunctives and disjunctives, CEA fails.

Next, we study the number of interactions needed to decide that a negotiation will not terminate successfully because the producer stock does not have a useful service for the consumer. This is important since ideally the negotiation algorithm should detect that the preference cannot be satisfied as early as possible, so as to not tire the consumer out. Table 10 shows the average number of interactions needed to decide that the consumer's preferences cannot be satisfied by the existing services in the stock. For some preference profiles, all of the stocks have some satisfactory service so we are interested in cases for other preferences. As seen in Table 10, DCEA has 50 interactions for each case, which is the worst case. This is due to the nature of DCEA. Since it only filters out the rejected offers of consumers, all remaining services are still plausible. On the other hand, RCEA generalizes the rejected offers and eliminates others that are similar to those that were rejected. Hence, RCEA ends the negotiation in a reasonable time, without showing the user all possible services in its stock. This is an important advantage of RCEA over DCEA. Meanwhile, CEA completes the negotiations even earlier than RCEA. However, this situation does not show that CEA

Table 10 Case-1: Avg # of interactions when stocks do not have the desired service

Preference	RCEA	CEA	DCEA
1	12.70	5.40	50.00
2	12.89	5.40	50.00
4	33.27	3.15	50.00
6	33.09	4.33	50.00
9	36.60	3.27	50.00
10	29.30	2.90	50.00
17	27.24	3.46	50.00
19	40.40	3.90	50.00
20	22.83	3.85	50.00
Avg.:	27.59	3.96	50.00

performs better because the number of successful runs for CEA is much less than that for RCEA.

These results show that CEA is not convenient in our negotiation domain, since it does not support disjunctives. DCEA is one of the alternative learning algorithms that can be used in learning preferences but the number of interactions that are necessary to complete the negotiation when none of the producer's available services is high when compared to RCEA. We conclude that among the three Version space algorithms, RCEA performs better than both CEA and DCEA, when different success and performance criteria are considered.

5.2 Case 2: Comparing RCEA, ID3 and Bayesian

In this section, we compare RCEA with other well-known classifiers such as the famous decision tree algorithm ID3, and a Bayes' classifier. We experiment with three separate producers, Producer with Revisable Candidate Elimination Algorithm (RCEA), Producer with ID3 (ID3) and Producer with Bayes' classifier (Bayesian). We use 20 consumer profiles for evaluation and we have ten runs for 100 different stocks, resulting in 1,000 negotiations.

5.2.1 Success of producers: RCEA, ID3 and Bayesian

We first study the number of negotiations that are finalized successfully. Table 11 shows the number of successful negotiations for each preference (out of 1,000 trials). The last column shows the number of trials in which success was possible; i.e., there was a possible service that could satisfy the consumer. The figures show that RCEA may miss up to 8.2% of the time where ID3 may miss up to 5.3%. But, on average the number of successful negotiations for RCEA and ID3 are similar (918 versus 916.8). Bayesian may miss up to 25.4% and on average the number of negotiations that it completes successfully is less than that of RCEA and ID3.

5.2.2 Performance of producers: RCEA, ID3 and Bayesian

The results presented in Sect. 5.2.1 show that success of the algorithms are close to each other. We see that RCEA is slightly more successful in providing a service that will satisfy the consumer. Now, we study the performance of these algorithms. Table 12 compares the

Table 11 Case-2: Number of negotiations that end with consensus

	# of success			Poss. Successful
Preference	RCEA	ID3	Bayesian	Negotiations
1	970	969	963	970
2	450	437	401	450
3	998	1,000	998	1,000
4	786	790	739	790
5	1,000	1,000	1,000	1,000
6	894	871	813	920
7	1,000	1,000	1,000	1,000
8	1,000	1,000	999	1,000
9	968	958	899	970
10	960	983	946	990
11	1,000	1,000	986	1,000
12	1,000	999	989	1,000
13	1,000	994	981	1,000
14	1,000	1,000	999	1,000
15	999	988	977	1,000
16	1,000	1,000	1,000	1,000
17	529	526	395	530
18	1,000	1,000	999	1,000
19	989	972	965	990
20	817	849	782	890
Avg.:	918.00	916.80	891.55	925.00

three algorithms in terms of the number of interactions needed to complete a negotiation successfully. Accordingly, RCEA needs fewer interactions to end a negotiation successfully when its performance is compared to others.

Note that in 17,665 out of 20,000 negotiation runs all of the producers (RCEA, ID3 and Bayesian) are successful at the same run. We apply Friedman test to the results of the 17,665 runs. Under 99 per cent confidence level, the results of RCEA, ID3, and Bayesian are statistically significantly different to each other and according to the number of interactions when all are successful, the ordering can be represented as $RCEA \prec Bayesian \prec ID3$, which means that RCEA negotiates faster than Bayesian, which negotiates faster than ID3 when all producers complete the negotiation successfully.

Moreover, we investigate how many times each producer negotiates as fast as the fastest producer out of 17,665 negotiations. The first part of Fig. 5 shows this information for each producer graphically. It is seen that RCEA completes negotiations early in most of the cases (12,030), which supports the above results. The number of times that ID3 reaches a consensus early is higher than the number of times that Bayesian reaches a consensus ($9,686 > 7,562$), but still less than that of RCEA. We also calculate the proportion of the number of minimum interactions to the number of interactions that producer needs for each negotiation. This proportion shows the relative speed (i.e., in terms of number of interactions) of an algorithm compared to the best algorithm for a given negotiation instance. Higher proportion means that

Table 12 Case-2: Average # of interactions when all are successful

Preference	RCEA	ID3	Bayesian
1	5.10	9.08	8.12
2	6.97	15.21	10.68
3	4.68	4.87	5.27
4	14.21	14.52	12.99
5	2.79	2.88	3.26
6	8.64	9.98	8.79
7	2.48	2.53	2.66
8	3.27	3.27	3.39
9	11.61	13.04	11.51
10	11.24	11.92	11.66
11	6.88	7.50	7.07
12	7.00	7.47	7.21
13	6.87	7.23	6.83
14	3.56	3.68	4.06
15	7.64	9.32	8.46
16	2.38	2.48	2.49
17	11.69	13.73	11.47
18	2.88	3.06	3.46
19	8.26	9.73	8.30
20	9.07	11.68	10.65
Avg.:	6.48	7.50	6.97
StdDev.:	6.66	7.78	6.53

the number of interactions that the producer needs is closer to the fastest producer. The second part of Fig. 5 draws the proportions for 17,665 negotiations. It is seen that the proportion for RCEA is best, while ID3 is higher than Bayesian (nearly $0.77 > 0.72$). Note an interesting relation between ID3 and Bayesian. From Table 12, we know that on average the number of interactions required to complete a negotiation successfully is less in Bayesian than that in ID3. However, in Fig. 5, we see that the relative speed of ID3 is higher than Bayesian. We observe that this stems from the fact that sometimes Bayesian completes a negotiation much earlier than ID3—the difference between the number of interactions between ID3 and Bayesian is significantly higher for this case.

Next, we study the number of interactions needed to decide that a negotiation will not terminate successfully because the producer stock does not have a useful service for the consumer. Table 13 shows the average number of interactions needed to decide to end a negotiation in the case of the stock does not have any desired service for the consumer. (Thus, the table only shows cases when a stock does not contain services that will satisfy the consumer's preferences.) Both RCEA and Bayesian end the negotiation in a reasonable time, without showing the user all possible services in their stocks. Compared to RCEA and Bayesian, ID3 needs more interaction to complete the negotiation. As a result, RCEA can facilitate faster negotiation of service descriptions. If no consensus can be found, RCEA and Bayesian signal this much earlier than ID3.

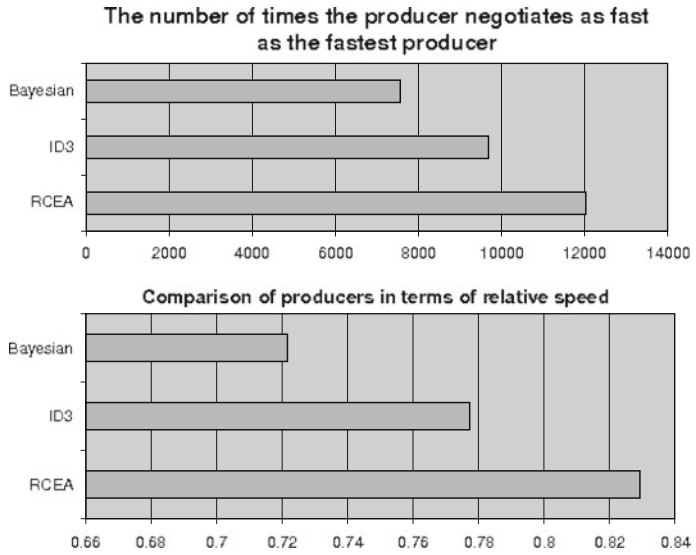

Fig. 5 Performance of RCEA, ID3 and Bayesian

Table 13 Case-2: Average # of interactions when stocks do not have the desired service

Pref	RCEA	ID3	Bayesian
1	12.70	26.43	24.70
2	12.89	35.43	21.81
4	33.27	30.45	26.83
6	33.09	34.68	26.21
9	36.60	42.67	27.97
10	29.30	30.20	31.10
17	27.24	36.73	26.27
19	40.40	44.10	29.00
20	22.83	29.03	22.60
Avg.:	27.60	34.41	26.28

6 Discussion

In this section, we give a brief description of our negotiation framework and explain important aspects of our learning approach (Sect. 6.1). Further, we compare our work with the related works (Sect. 6.2) and conclude with future directions (Sect. 6.3).

6.1 Summary

We present a multi-issue negotiation model in which the producer tries to learn the consumer's preferences and generate offers accordingly. Understanding consumer's needs is a significant matter in negotiation since generating well-targeted offers for an agent depends on its knowledge about the negotiation partner. Since the preferences of the participants are almost private, the best action is to learn these preferences by using the requests exchanged during the negotiation.

As far as preference elicitation phase is concerned, it would be more intuitive for the human user to give her preferences in a qualitative way rather than quantifying the preferences with utility functions. One alternative representation for the preference is a set of constraints in the form of conjunctives and disjunctives. During this study, the consumer's preferences consist of such kind of constraints. The producer agent tries to learn these unknown preferences from the bids during the negotiation. Since training data such as positive and negative examples come throughout the interaction, an incremental learning algorithm in which the training instances come over time is required.

Therefore, we develop a learning algorithm called RCEA supporting learning disjunctives in an incremental way. As humans use their common knowledge about the domain when they predict someone's private information, this learning algorithm also is able to use domain knowledge. In other words, our learning algorithm is enhanced with ontology reasoning. The producer agent performs learning after each interaction with the consumer. While the consumer generates its request according to its preferences, the producer considers both its own utility and the learned consumer's preferences during the negotiation. Consequently, the counter offer would be beneficial for both participants.

Our results show that the participants reach early agreement when our proposed learning algorithm is used. Moreover, if consensus is not possible, the producer can realize this earlier than other approaches. Since CEA does not support learning disjunctives, it fails in most of the cases, especially when the consumer's preferences involves disjunctives. When the producer does not have any acceptable services for the consumer, the producer using DCEA does not realize this so it leads to many unnecessary interactions between agents, which prolongs the negotiation time. Although the number of successful negotiations of ID3 is closer to that of our algorithm, the number of interactions it needs to reach a consensus is worse than ours. On the other hand, Bayesian negotiates relatively faster than ID3, but it does not lead to as many successful negotiations as ID3 or RCEA. To sum up, RCEA outperforms other algorithms in terms of the number of successful negotiations, the number of interactions that is required to come up an agreement or to decide the negotiation will be unsuccessful.

6.2 Discussion of related work

We review related work from the literature. A variety of learning and modeling techniques are used in the literature. We first review leading research that represent preferences by utility functions [18]. A sequential negotiation model in which a Bayesian learning is used to update the beliefs about the participants is presented by Zeng and Syraca [32]. Their proposed negotiation model is a generic framework allowing multiple participants and multiple issues. The Bayesian rule is used to update the agent's belief about the opponent agent such as the opponent's reservation values. When we focus on the learning part of this study and their experiments, it can be said that they mostly consider the negotiation in which there is only a single issue, particularly the price. The agent tries to learn the opponent's reservation value for the price by using its observations. After each new coming information about the opponent or environment, the agent's belief is updated. Note that to be able to apply the Bayesian update rule, a priori knowledge is required. The quality of the priori knowledge will possibly affect the performance of the learning. According to our approach, learning can be performed without any knowledge about the opponents since in a open environment the agents may meet each other for the first time. Thus, our approach does not require a priori knowledge. In contrast to single issue negotiation, we mainly consider multi issues to be negotiated.

Hindriks and Tykhonov also represent preferences using utility functions and they apply Bayesian learning to learn the opponent's preferences from bid exchanges [14]. In order to apply Bayesian learning, they make some assumptions about preference structure and opponent's negotiation strategy. The preferences are represented as a weighted sum of utility functions (linearly additive functions). Three types of evaluation functions are defined: downhill, uphill and triangular. Their hypothesis space consists of these predefined function types. A probability distribution is associated with these functions. If the prior knowledge does not exist, a uniform distribution is used for the prior probabilities. The opponent's offers are used as an evidence and the opponent's strategy is assumed to be a concession-based strategy. Thus, the conditional probability of the hypotheses with the given evidence is updated by using the Bayes' rule. These probabilities are used to decide the agent's counter offer. If we consider the negotiation between the buyer and the seller, the assumptions about preference structure may not be applicable, because, the preferences of the buyer can include interdependencies. For instance, the buyer may prefer dry wine if the color of the wine is red whereas she may prefer sweet wine when the color is white. In contrast to that study, our negotiation approach allows the agent to have such preferences. Hindriks and Tykhonov model the user's preferences as a utility function. However, we keep preferences as a set of concepts. Therefore, we do not deal with finding a numerical model.

Buffett and Spencer propose a multi-object negotiation in which a Bayesian classification method is applied for learning the opponent's preference relation [8]. There are predefined classes for preference relations and the aim is to find the class to which the opponent's preference relation belongs. The aim is to narrow down the search space. It is assumed that the opponent applies a concession-based strategy. In this negotiation model, the agents negotiate over the subsets of a set of objects. Here, one participant prefers more objects whereas other prefers less. It is known that if the agent prefers more, a superset of an offer is preferred by this agent. In that study, the authors are not concerned with determining the initial preference classes or deciding on the prior probabilities. The evidence is estimated by using the bid exchanges. Their experiment results mainly give information about the performance of the classifier. However, there are no experimental results about the negotiation such as joint utility, number of rounds and so on. We do not deal with multi-object negotiation. Instead, we study a multi-issue negotiation in which a concept learning method is applied to learn the consumer's preferences. Also, our experiment results are related to the negotiation performance and success.

Another line of related research is in generating counter offers. Evolutionary learning approaches, especially Genetic Algorithms are used in negotiation to generate counter offers [9,19]. The candidate offers are represented by the chromosomes and their fitness function captures both agent's own utility and opponent's last offer. The agent decides the next offer according to the result of its genetic algorithm. A parameter controls the agent's attitude (the amount of selfishness and benevolence). Moreover, the effect of time pressure is reflected by another parameter. Lau et al. prefer learning opponent's preferences indirectly rather than finding an exact model for the preferences in their study [19]. Similar to the work of Faratin, Sierra and Jennings [11], they take the opponent's last offer into account. Their fitness function considers the agent's own utility and the opponent's last offer. The aim is to find a solution whose distance is minimal to them. Contrary to their study, we do not only consider the opponent's last offer but also its all previous offers. By using all bids exchanged during the negotiation, our aim is to generate offers, which are possibly acceptable as far as the opponent's behavior is concerned. To do this, we apply an inductive learning enhanced with ontology reasoning about the negotiation domain.

Boutilier, Regan and Viappiani use concept learning to learn the user's feature definition in preference elicitation [7] whereas in our study concept learning is used to understand the opponent agent's preferences and more importantly to generate counter offers. While our aim is to learn other agent's preference descriptions, they mainly concentrate on reducing relevant concept uncertainty. They do not try to find the accurate concept definitions. They use membership queries to obtain negative and positive examples. Whereas we handle learning both disjunctive and conjunctive concepts, they use only conjunctive concepts.

In our approach, we assume that the consumer's preferences are private and the producer can only discover them over interactions. Hence, the producer has to apply a learning algorithm to learn the preferences over interaction. However, in some approaches in the literature, it is assumed that the consumer reveals part of its preferences. Hence, the amount of learning that needs to be done is less. We review two such approaches next.

Jonker, Robu and Treur propose a negotiation architecture in which two agents negotiate over multiple attributes under incomplete preference information [17]. As the privacy of the participants is concerned, the agents are free to either reveal their preferences for some attributes or to keep some private. A target utility of the current bid is estimated by determining the concession step. This target utility is distributed over the attributes. To distribute this target utility over the attributes, agent's own preference weights and opponent's preference weights are taken into account. However, the agent has only limited information about its opponent. The opponent may reveal some of its preference parameters. For opponent's unknown weights, they apply a heuristic to guess these unknown weights by using the history of the bids that the opponent offers. According to the target value of the each attribute, agent chooses a convenient value for each attribute separately. At the end, an approximation is done to the target utility. As far as modeling of the opponent's preferences are concerned, they only focus on predicting the negotiation partner's preference weights (that indicate the importance degree of the attributes). Apart from the weights, opponent's preferences on the values of the attributes are also important. Thus, in our study, we focus on finding an approximate preference model (a set of constraints in our application), which is used for avoiding unacceptable offers for the opponent and selecting offers, which are more likely to be acceptable for the consumers. Whereas Jonker, Robu and Treur are mainly concerned about the efficiency of the outcome for both agents, we concentrate on how fast the producer can make offers that will satisfy the consumer.

Luo et al. propose a prioritized fuzzy constraint based negotiation model in which the buyer agent specifies its offers by using constraints [20]. According to the proposed model, the agent reveals its partial preference information (some parts of its constraints) in a minimal fashion. That is, the buyer agent sends its constraints to the seller. The seller agent checks the constraints and if it cannot satisfy this constraints, it requests the buyer to relax them. In our model, the consumer does not reveal its partial preferences. Further, the seller tries to predict what the buyer would like if her initial request cannot be satisfied. Hence, the challenge in our study is to find out a model for consumer's private preferences.

The following three approaches deal with aspects that are complementary to our work. In order to shorten the negotiation time for better agreements, Ramchurn et al. propose a negotiation algorithm using rewards [28]. They show that their negotiation tactic based on the generation of rewards improves the utility of the deals. The authors formalize their repeated negotiation games by defining a set of issues and a set of values for them. For each agent, there is a private utility function over each issue. The utility over a contract is the weighted sum of the individual utility of all issues. The utility is discounted with a rate depending on the time between each transmitted offer and delay between two games. In our work, we do not consider rewards. We concentrate on constructing an approximation to the consumer's

preferences. By eliminating the offers possibly to be rejected by the consumer, we increase the chance of generating a mutually acceptable offer.

Rahwan et al. propose an argumentation-based negotiation approach that is based on the exchange of meta-information about negotiation [25]. As a result, an agent may influence other agents' states. Thus, it provides more sophisticated interactions involving promise, threat, justification of a proposal and so on. With the help of arguments, the probability of the acceptance of other agent's offer is expected to increase. For example, the agent can generate an argument explaining what its aim is. The other agent may offer an alternative proposal helping the agent reach its goal. Our negotiation model does not involve arguments but it would be interesting to extend our model by adding arguments.

Somefun and Poutré propose a learning method to learn customer's preferences by using anonymous data on passed negotiations [30]. In their system, there is one agent negotiating with several customers about a collection of services that are associated with a price. They consider the inter-dependencies between the services. We do not keep the information obtained in the past negotiations and we deal with a single service rather than a collection of services. It would be interesting to negotiate over a set of services and using past information to improve the negotiation process.

6.3 Future work

Our main direction for future research is the addition of preference prioritization to our negotiation framework. That is, for the user, one constraint may be more important than another one. For example, the user may express her preferences in a way that she firstly prefers a red and dry wine and she secondly prefers a white and sweet wine. It would be useful if our algorithm could realize that the first constraint is more preferred and would generate offers accordingly. Furhter, with disjunctive preferences, simple conditional preferences can be shown. To capture more complex cases, other representations, such as CP-net [6] may be used. We started applying RCEA to learn preferences that are represented with other preference models, such as CP-nets [5].

Acknowledgments This paper significantly extends a paper that appeared in IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2009). This research has been supported by Boğaziçi University Research Fund under grant BAP09A106P and The Scientific and Technological Research Council of Turkey by a CAREER Award under grant 105E073. We thank Mehmet Gönen for useful comments.

References

1. Abedin, F., Chao, K.-M., Godwin, N., & Arochena, H. (2009). Preference ordering in agenda based multi-issue negotiation for service level agreement. In *WAINA '09: Proceedings of the 2009 international conference on advanced information networking and applications workshops*, (pp. 19–24). Washington, DC, USA: IEEE Computer Society.
2. Alpaydın, E. (2004). *Introduction to machine learning*. Cambridge, MA: The MIT Press.
3. Aydoğan, R., & Yolum, P. (2007). Learning consumer preferences using semantic similarity. In *Proceedings of sixth international joint conference on autonomous agents and multiagent systems (AAMAS)* (pp. 1293–1300). Honolulu, Hawaii.
4. Aydoğan, R., & Yolum, P. (2009). Ontology-based learning for negotiation. In *IEEE/WIC/ACM international conference on intelligent agent technology (IAT 2009)* (pp. 177–184). Milan, Italy.
5. Aydoğan R., & Yolum, P. (2010). The effect of preference representation on learning preferences in negotiation. In *The third international workshop on agent-based complex automated negotiations (ACAN 2010)* (pp. 1–8). Toronto, Canada.

6. Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., & Poole, D. (2004). Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR), 21*, 135–191.
7. Boutilier, C., Regan, K., & Viappiani, P. (2009). Preference elicitation with subjective features. In *RecSys '09: Proceedings of the third ACM conference on recommender systems* (pp. 341–344). New York, NY, USA: ACM
8. Buffett, S., & Spencer, B. (2005). Learning opponents' preferences in multi-object automated negotiation. In *ICEC '05: Proceedings of the 7th international conference on Electronic commerce* (pp. 300–305). New York, NY, USA: ACM
9. Choi, S. P. M., Liu, J., & Chan, S. (2001). A genetic agent-based negotiation system. *Computer Networks, 37*(2), 195–204.
10. Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*, 1–30.
11. Faratin, P., Sierra, C., & Jennings, N. R. (2002). Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence, 142*, 205–237.
12. Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics, 11*(1), 86–92.
13. Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition, 5*(2), 199–220.
14. Hindriks, K., & Tykhonov, D. (2008). Opponent modelling in automated multi-issue negotiation using bayesian learning. In *7th international joint conference on autonomous agents and multiagent systems (AAMAS)* (pp. 331–338).
15. Jena, (2009). http://jena.sourceforge.net/.
16. Jennings, N. R., Faratin, P., Lomuscio, A. R., Parsons, S., Sierra, C., & Wooldridge, M. (2001). Automated negotiation: Prospects, methods and challenges. *International Journal of Group Decision and Negotiation, 10*(2), 199–215.
17. Jonker, C. M., Robu, V., & Treur, J. (2007). An agent architecture for multi-attribute negotiation using incomplete preference information. *Autonomous Agents and Multi-Agent Systems, 15*(2), 221–252.
18. Lai, G., Li, C., Sycara, K., & Giampapa, J.A. (2004). *Literature review on multi-attribute negotiations*. Technical report, Robotics Institute, Pittsburgh, PA.
19. Lau, R. Y. K., Tang, M., Wong, O., Milliner, S. W., & Chen, Y. P. (2006). An evolutionary learning approach for adaptive negotiation agents. *International Journal of Intelligent Systems, 21*(1), 41–72.
20. Luo, X., Jennings, N. R., Shadbolt, N., Leung, H., & Lee, J. H. (2003). A fuzzy constraint based model for bilateral, multi-issue negotiations in semi-competitive environments. *Artifical Intelligence, 148* (1–2), 53–102.
21. McGuinness, D. L. (2003). Ontologies come of age. In *Spinning the semantic web* (pp. 171–194). Cambridge: MIT Press.
22. Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence, 18*(2), 203–226.
23. Mitchell, T. M. (1997). *Machine learning*. New York: McGraw Hill.
24. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.
25. Rahwan, I., Ramchurn, S. D., Jennings, N. R., Mcburney, P., Parsons, S., & Sonenberg, L. (2003). Argumentation-based negotiation. *Knowledge Engineering Review, 18*(4), 343–375.
26. Raiffa, H. (1982). *The art and science of negotiation*. Cambridge: Harvard University Press.
27. Raileanu, L. E., & Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence, 41*(1), 77–93.
28. Ramchurn, S. D., Sierra, C., Godo, L., & Jennings, N. R. (2006). Negotiating using rewards. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems* (pp. 400–407).
29. Shannon, C., Petigara, N., & Seshasai, S. (1948). A mathematical theory of. *Communication, Bell System Technical Journal, 27*, 379–423.
30. Somefun, D. J. A., & Poutré, J. A. L. (2007). A fast method for learning non-linear preferences online using anonymous negotiation data. In *Agent-mediated electronic commerce. Automated negotiation and strategy design for electronic markets*. Lecture notes in computer science (pp. 118–131). Springer
31. Wine, (2009). http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf.
32. Zheng, D. D., & Sycara, K. (1998). Bayesian learning in negotiation. *International Journal of Human-Computers Studies, 48*(1), 125–141.