Learning Consumer Preferences Using Semantic Similarity *

Reyhan Aydoğan
reyhan.aydogan@gmail.com

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of Computer Engineering
Boğaziçi University
Bebek, 34342, Istanbul,Turkey

ABSTRACT

In online, dynamic environments, the services requested by consumers may not be readily served by the providers. This requires the service consumers and providers to negotiate their service needs and offers. Multiagent negotiation approaches typically assume that the parties agree on service content and focus on finding a consensus on service price. In contrast, this work develops an approach through which the parties can negotiate the content of a service. This calls for a negotiation approach in which the parties can understand the semantics of their requests and offers and learn each other's preferences incrementally over time. Accordingly, we propose an architecture in which both consumers and producers use a shared ontology to negotiate a service. Through repetitive interactions, the provider learns consumers' needs accurately and can make better targeted offers. To enable fast and accurate learning of preferences, we develop an extension to Version Space and compare it with existing learning techniques. We further develop a metric for measuring semantic similarity between services and compare the performance of our approach using different similarity metrics.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Experimentation

Keywords

Negotiation, Inductive Learning, Ontology, Semantic Similarity

*This research is supported by Boğaziçi University Research Fund under grant BAP06A103 and Turkish Research and Technology Council CAREER Award under grant 105E073. This paper significantly extends a previous version that appeared in AAMAS 2006 BASeWEB Workshop. We thank Levent Akın, Çetin Meriçli, Nadia Erdoğan, and the anonymous reviewers for helpful comments.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

1. INTRODUCTION

Current approaches to e-commerce treat service price as the primary construct for negotiation by assuming that the service content is fixed [9]. However, negotiation on price presupposes that other properties of the service have already been agreed upon. Nevertheless, many times the service provider may not be offering the exact requested service due to lack of resources, constraints in its business policy, and so on [3]. When this is the case, the producer and the consumer need to negotiate the *content* of the requested service [15].

However, most existing negotiation approaches assume that all features of a service are equally important and concentrate on the price [5, 2]. However, in reality not all features may be relevant and the relevance of a feature may vary from consumer to consumer. For instance, completion time of a service may be important for one consumer whereas the quality of the service may be more important for a second consumer. Without doubt, considering the preferences of the consumer has a positive impact on the negotiation process. For this purpose, evaluation of the service components with different weights can be useful. Some studies take these weights as a priori and uses the fixed weights [4]. On the other hand, mostly the producer does not know the consumer's preferences before the negotiation. Hence, it is more appropriate for the producer to learn these preferences for each consumer.

Preference Learning: As an alternative, we propose an architecture in which the service providers learn the relevant features of a service for a particular customer over time. We represent service requests as a vector of service features. We use an ontology in order to capture the relations between services and to construct the features for a given service. By using a common ontology, we enable the consumers and producers to share a common vocabulary for negotiation. The particular service we have used is a wine selling service. The wine seller learns the wine preferences of the customer to sell better targeted wines. The producer models the requests of the consumer and its counter offers to learn which features are more important for the consumer. Since no information is present before the interactions start, the learning algorithm has to be incremental so that it can be trained at run time and can revise itself with each new interaction.

Service Generation: Even after the producer learns the important features for a consumer, it needs a method to generate offers that are the most relevant for the consumer among its set of possible services. In other words, the question is how the producer uses the information that was learned from the dialogues to make the best offer to the consumer. For instance, assume that the producer has learned that the consumer wants to buy a red wine but the producer can only offer rose or white wine. What should the producer's offer

978-81-904262-7-5 (RPS)

contain; white wine or rose wine? If the producer has some domain knowledge about semantic similarity (e.g., knows that the red and rose wines are taste-wise more similar than white wine), then it can generate better offers. However, in addition to domain knowledge, this derivation requires appropriate metrics to measure similarity between available services and learned preferences.

The rest of this paper is organized as follows: Section 2 explains our proposed architecture. Section 3 explains the learning algorithms that were studied to learn consumer preferences. Section 4 studies the different service offering mechanisms. Section 5 contains the similarity metrics used in the experiments. The details of the developed system is analyzed in Section 6. Section 7 provides our experimental setup, test cases, and results. Finally, Section 8 discusses and compares our work with other related work.

2. ARCHITECTURE

Our main components are consumer and producer agents, which communicate with each other to perform content-oriented negotiation. Figure 1 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 wines are available or not.

A shared ontology provides the necessary vocabulary and hence enables a common language for agents. This ontology describes the content of the service. Further, since an ontology can represent concepts, their properties and their relationships semantically, the agents can 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 [19] with some modification to illustrate our ideas and to test our system. The wine ontology describes different types of wine and includes features 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 data repository in Figure 1 is used solely by the producer agent and holds the inventory information of the producer. The data repository includes information on the products the producer owns, the number of the products and ratings of those products. Ratings indicate the popularity of the products among customers. Those are used to decide which product will be offered when there exists more than one product having same similarity to the request of the consumer agent.

The negotiation takes place in a turn-taking fashion, where the consumer agent starts the negotiation with a particular service request. The request is composed of significant features of the service. In the wine example, these features include color, winery and so on. This is the particular wine that the customer is interested in purchasing. If the producer has the requested wine in its inventory, the producer offers the wine and the negotiation ends. Otherwise, the producer offers an alternative wine from the inventory. When the consumer receives a counter offer from the producer, it will evaluate it. If it is acceptable, then the negotiation will end. Otherwise, the customer will generate a new request 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.

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

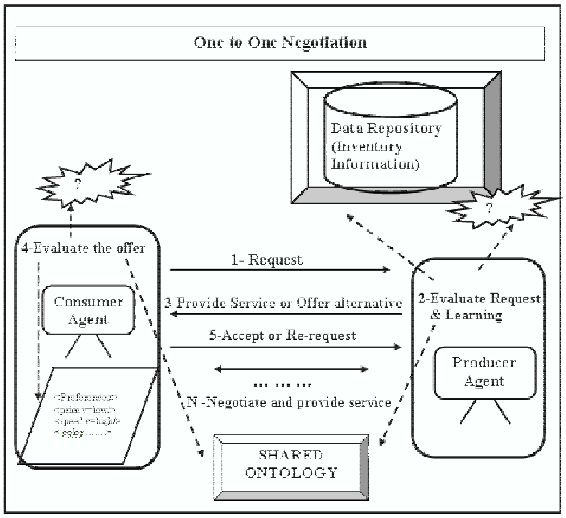

Figure 1: Proposed Negotiation Architecture

three important things: the current request, consumer 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. This challenge can be studied in three stages:

- Preference Learning: How can the producers learn about each customer's preferences based on requests and counter offers? (Section 3)
- Service Offering: How can the producers revise their offers based on the consumer's preferences that they have learned so far? (Section 4)
- Similarity Estimation: How can the producer agent estimate similarity between the request and available services? (Section 5)

3. PREFERENCE LEARNING

The requests of the consumer and the counter offers of the producer are represented as vectors, where each element in the vector corresponds to the value of a feature. The requests of the consumers represent individual wine products whereas their preferences are constraints over service features. For example, a consumer may have preference for red wine. This means that the consumer is willing to accept any wine offered by the producers as long as the color is red. Accordingly, the consumer generates a request where the color feature is set to red and other features are set to arbitrary values, e.g. *(Medium, Strong, Red)*.

At the beginning of negotiation, the producer agent does not know the consumer's preferences but will need to learn them using information obtained from the dialogues between the producer and the consumer. The preferences denote the relative importance of the features of the services demanded by the consumer agents. For instance, the color of the wine may be important so the consumer insists on buying the wine whose color is *red* and rejects all

Table 1: How DCEA works

Type	Sample	The most general set	The most specific set
+	(Full,Strong,White)	{(?, ?, ?)}	{(Full,Strong,White)}
-	(Full,Delicate,Rose)	{{(?-Full), ?, ? }, {?, (?-Delicate), ?}, {?, ?, (?-Rose)}}	{(Full,Strong,White)}
+	(Medium,Moderate,Red)	{{(?-Full), ?, ?}, {?,(?-Delicate), ?}, {?, ?, (?-Rose)}}	{{(Full,Strong,White)}, {(Medium,Moderate,Red)}}

the offers involving the wine whose color is *white* or *rose*. On the contrary, the winery may not be as important as the color for this customer, so the consumer may have a tendency to accept wines from any winery as long as the color is *red*.

To tackle this problem, we propose to use incremental learning algorithms [6]. This is necessary since no training data is available before the interactions start. We particularly investigate two approaches. The first one is inductive learning. This technique is applied to learn the preferences as concepts. We elaborate on Candidate Elimination Algorithm (CEA) for Version Space [10]. CEA is known to perform poorly if the information to be learned is disjunctive. Interestingly, most of the time consumer preferences are disjunctive. Say, we are considering an agent that is buying wine. The consumer may prefer red wine or rose wine but not white wine. To use CEA with such preferences, a solid modification is necessary. The second approach is decision trees. Decision trees can learn from examples easily and classify new instances as positive or negative. A well-known incremental decision tree is ID5R [18]. However, ID5R is known to suffer from high computational complexity. For this reason, we instead use the ID3 algorithm [13] and iteratively build decision trees to simulate incremental learning.

3.1 CEA

CEA [10] is one of the inductive learning algorithms that learns concepts from observed examples. The algorithm maintains two sets to model the concept to be learned. The first set is the most general set G. G contains hypotheses about all the possible values that the concept may obtain. As the name suggests, it is a generalization and contains all possible values unless the values have been identified not to represent the concept. The second set is the most specific set S. S contains only hypotheses that are known to identify the concept that is being learned. At the beginning of the algorithm, G is initialized to cover all possible concepts while S is initialized to be empty.

During the interactions, each request of the consumer can be considered as a positive example and each counter offer generated by the producer and rejected by the consumer agent can be thought of as a negative example. At each interaction between the producer and the consumer, both G and S are modified. The negative samples enforce the specialization of some hypotheses so that G does not cover any hypothesis accepting the negative samples as positive. When a positive sample comes, the most specific set S should be generalized in order to cover the new training instance. As a result, the most general hypotheses and the most special hypotheses cover all positive training samples but do not cover any negative ones. Incrementally, G specializes and S generalizes until G and S are equal to each other. When these sets are equal, the algorithm converges by means of reaching the target concept.

3.2 Disjunctive CEA

Unfortunately, CEA is primarily targeted for conjunctive concepts. On the other hand, we need to learn disjunctive concepts in the negotiation of a service since consumer may have several alternative wishes. There are several studies on learning disjunctive concepts via Version Space. Some of these approaches use multiple version space. For instance, Hong *et al.* maintain several version spaces by split and merge operation [7]. To be able to learn disjunctive concepts, they create new version spaces by examining the consistency between G and S.

We deal with the problem of not supporting disjunctive concepts of CEA by extending our hypothesis language to include disjunctive hypothesis in addition to the conjunctives and negation. 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 taken for that feature.

EXAMPLE 1. *Assume that the most specific set is {(Light, Delicate, Red)} and a positive example, (Light, Delicate, White) comes. The original CEA will generalize this as (Light, Delicate, ?), meaning the color can take any value. However, in fact, we only know that the color can be red or white. In the DCEA, we generalize it as {(Light, Delicate, [White, Red])}. Only when all the values exist in the list, they will be replaced by ?. In other words, we let the algorithm generalize more slowly than before.*

We modify the CEA algorithm to deal with this change. The modified algorithm, DCEA, is given as Algorithm 1. Note that compared to the previous studies of disjunctive versions, our approach uses only a single version space rather than multiple version space. The initialization phase is the same as the original algorithm (lines 1, 2). If any positive sample comes, we add the sample to the special set as before (line 4). However, we do not eliminate the hypotheses in G that do not cover this sample since G now contains a disjunction of many hypotheses, some of which will be conflicting with each other. Removing a specific hypothesis from G will result in loss of information, since other hypotheses are not guaranteed to cover it. After some time, some hypotheses in S can be merged and can construct one hypothesis (lines 6, 7).

When a negative sample comes, we do not change S as before. We only modify the most general hypotheses not to cover this negative sample (lines 11–15). Different from the original CEA, we try to specialize the G minimally. The algorithm removes the hypothesis covering the negative sample (line 13). Then, we generate new hypotheses as the number of all possible attributes by using the removed hypothesis.

For each attribute in the negative sample, we add one of them at each time to the exclusive list of the removed hypothesis. Thus, all possible hypotheses that do not cover the negative sample are generated (line 14). Note that, exclusive list contains the values that the attribute cannot take. For example, consider the color attribute. If a hypothesis includes red in its exclusive list and *?* in its inclusive list, this means that color may take any value except red.

Algorithm 1 Disjunctive Candidate Elimination Algorithm

```
1: G ←the set of maximally general hypotheses in H
2: S ←the set of maximally specific hypotheses in H
3: For each training example, d
4: if d is a positive example then
5:    Add d to S
6:    if s in S can be combined with d to make one element then
7:       Combine s and d into sd {sd is the rule covers s and d}
8:    end if
9: end if
10: if d is a negative example then
11:    For each hypothesis g in G does cover d
12:    * Assume : g = (x1, x2, ..., xn) and d = (d1, d2, ..., dn)
13:    – Remove g from G
14:    – Add hypotheses g1, g2, gn where g1= (x1-d1, x2,..., xn),
       g2= (x1, x2-d2,..., xn),..., and gn= (x1, x2,..., xn-dn)
15:    – Remove from G any hypothesis that is less general than
       another hypothesis in G
16: end if
```

EXAMPLE 2. *Table 1 illustrates the first three interactions and the workings of DCEA. The most general set and the most specific set show the contents of G and S after the sample comes in. After the first positive sample, S is generalized to also cover the instance. The second sample is negative. Thus, we replace (?, ?, ?) by three disjunctive hypotheses; each hypothesis being minimally specialized. In this process, at each time one attribute value of negative sample is applied to the hypothesis in the general set. The third sample is positive and generalizes S even more.*

Note that in Table 1, we do not eliminate *{(?-Full), ?, ?}* from the general set while having a positive sample such as *(Full, Strong, White)*. This stems from the possibility of using this rule in the generation of other hypotheses. For instance, if the example continues with a negative sample *(Full, Strong, Red)*, we can specialize the previous rule such as *{(?-Full), ?, (?-Red)}*. By Algorithm 1, we do not miss any information.

3.3 ID3

ID3 [13] is an algorithm that constructs decision trees in a top-down fashion from the observed examples represented in a vector with attribute-value pairs. Applying this algorithm to our system with the intention of learning the consumer's preferences is appropriate since this algorithm also supports learning disjunctive concepts in addition to conjunctive concepts.

The ID3 algorithm is used in the learning process with the purpose of classification of offers. There are two classes: positive and negative. Positive means that the service description will possibly be accepted by the consumer agent whereas the negative implies that it will potentially be rejected by the consumer. Consumer's requests are considered as positive training examples and all rejected counter-offers are thought as negative ones.

The decision tree has two types of nodes: leaf node in which the class labels of the instances are held and non-leaf nodes in which test attributes are held. The test attribute in a non-leaf node is one of the attributes making up the service description. For instance, body, flavor, color and so on are potential test attributes for wine service. When we want to find whether the given service description is acceptable, we start searching from the root node by examining the value of test attributes until reaching a leaf node.

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 keeps consumer's requests throughout the negotiation interaction as positive examples and all counter-offers rejected by the consumer as negative examples. After each coming request, the decision tree is rebuilt. Without doubt, there is a drawback of reconstruction such as additional process load. However, in practice we have evaluated ID3 to be fast and the reconstruction cost to be negligible.

4. SERVICE OFFERING

After learning the consumer's preferences, the producer needs to make a counter offer that is compatible with the consumer's preferences.

4.1 Service Offering via CEA and DCEA

To generate the best offer, the producer agent uses its service ontology and the CEA algorithm. The service offering mechanism is the same for both the original CEA and DCEA, but as explained before their methods for updating G and S are different.

When producer receives a request from the consumer, the learning set of the producer is trained with this request as a positive sample. The learning components, the most specific set S and the most general set G are actively used in offering service. The most general set, G is used by the producer in order to avoid offering the services, which will be rejected by the consumer agent. In other words, it filters the service set from the undesired services, since G contains hypotheses that are consistent with the requests of the consumer. The most specific set, S is used in order to find best offer, which is similar to the consumer's preferences. Since the most specific set S holds the previous requests and the current request, estimating similarity between this set and every service in the service list is very convenient to find the best offer from the service list.

When the consumer starts the interaction with the producer agent, producer agent loads all related services to the service list object. This list constitutes the provider's inventory of services. Upon receiving a request, if the producer can offer an exactly matching service, then it does so. For example, for a wine this corresponds to selling a wine that matches the specified features of the consumer's request identically. When the producer cannot offer the service as requested, it tries to find the service that is most similar to the services that have been requested by the consumer during the negotiation. To do this, the producer has to compute the similarity between the services it can offer and the services that have been requested (in S).

We compute the similarities in various ways as will be explained in Section 5. After the similarity of the available services with the current S is calculated, there may be more than one service with the maximum similarity. The producer agent can break the tie in a number of ways. Here, we have associated a rating value with each service and the producer prefers the higher rated service to others.

4.2 Service Offering via ID3

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 are removed from the service list. Second, the similarities of possible services are not measured with respect to S, but instead to all previously made requests.

4.3 Alternative Service Offering Mechanisms

In addition to these three service offering mechanisms (Service Offering with CEA, Service Offering with DCEA, and Service Offering with ID3), we include two other mechanisms..

- Random Service Offering (RO): The producer generates a counter offer randomly from the available service list, without considering the consumer's preferences.
- Service Offering considering only the current request (SCR): The producer selects a counter offer according to the similarity of the consumer's current request but does not consider previous requests.

5. SIMILARITY ESTIMATION

Similarity can be estimated with a similarity metric that takes two entries and returns how similar they are. There are several similarity metrics used in case based reasoning system such as weighted sum of Euclidean distance, Hamming distance and so on [12].

The similarity metric affects the performance of the system while deciding which service is the closest to the consumer's request. We first analyze some existing metrics and then propose a new semantic similarity metric named *RP Similarity*.

5.1 Tversky's Similarity Metric

Tversky's similarity metric compares two vectors in terms of the number of exactly matching features [17]. In Equation (1), $common$ represents the number of matched attributes whereas $different$ represents the number of the different attributes. Our current assumption is that α and β is equal to each other.

$$SM_{pq} = \frac{\alpha(common)}{\alpha(common) + \beta(different)} \quad (1)$$

Here, when two features are compared, we assign zero for dissimilarity and one for similarity by omitting the semantic closeness among the feature values.

Tversky's similarity metric is designed to compare two feature vectors. In our system, whereas the list of services that can be offered by the producer are each a feature vector, the most specific set S is not a feature vector. S consists of hypotheses of feature vectors. Therefore, we estimate the similarity of each hypothesis inside the most specific set S and then take the average of the similarities.

EXAMPLE 3. *Assume that S contains the following two hypothesis: { {Light, Moderate, (Red, White)} , {Full, Strong, Rose}}. Take service s as (Light, Strong, Rose). Then the similarity of the first one is equal to $1/3$ and the second one is equal to $2/3$ in accordance with Equation (1). Normally, we take the average of it and obtain $(1/3 + 2/3)/2$, equally $1/2$. However, the first hypothesis involves the effect of two requests and the second hypothesis involves only one request. As a result, we expect the effect of the first hypothesis to be greater than that of the second. Therefore, we calculate the average similarity by considering the number of samples that hypotheses cover.*

Let c_h denote the number of samples that hypothesis h covers and ($SM_{(h,service)}$) denote the similarity of hypothesis h with the given service. We compute the similarity of each hypothesis with the given service and weight them with the number of samples they cover. We find the similarity by dividing the weighted sum of the similarities of all hypotheses in S with the service by the number of all samples that are covered in S.

$$AVG\_SM_{(service,S)} = \frac{\sum_{|h|}^{|S|} (c_h * SM_{(h,service)})}{\sum_{|h|}^{|S|} c_h} \quad (2)$$

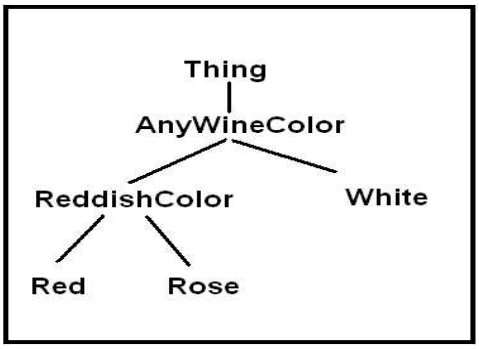

Figure 2: Sample taxonomy for similarity estimation

EXAMPLE 4. *For the above example, the similarity of (Light, Strong, Rose) with the specific set is $(2 * 1/3 + 2/3)/3$, equally $4/9$. The possible number of samples that a hypothesis covers can be estimated with multiplying cardinalities of each attribute. For example, the cardinality of the first attribute is two and the others is equal to one for the given hypothesis such as {Light, Moderate, (Red, White)}. When we multiply them, we obtain two $(2 * 1 * 1 = 2)$.*

5.2 Lin's Similarity Metric

A taxonomy can be used while estimating semantic similarity between two concepts. Estimating semantic similarity in a Is-A taxonomy can be done by calculating the distance between the nodes related to the compared concepts. The links among the nodes can be considered as distances. Then, the length of the path between the nodes indicates how closely similar the concepts are. An alternative estimation to use information content in estimation of semantic similarity rather than edge counting method, was proposed by Lin [8]. The equation (3) [8] shows Lin's similarity where c_1 and c_2 are the compared concepts and c_0 is the most specific concept that subsumes both of them. Besides, $P(C)$ represents the probability of an arbitrary selected object belongs to concept C.

$$Similarity(c_1, c_2) = \frac{2 \times \log P(c_0)}{\log P(c_1) + \log P(c_2)} \quad (3)$$

5.3 Wu & Palmer's Similarity Metric

Different from Lin, Wu and Palmer use the distance between the nodes in IS-A taxonomy [20]. The semantic similarity is represented with Equation (4) [20]. Here, the similarity between c_1 and c_2 is estimated and c_0 is the most specific concept subsuming these classes. N_1 is the number of edges between c_1 and c_0. N_2 is the number of edges between c_2 and c_0. N_0 is the number of IS-A links of c_0 from the root of the taxonomy.

$$Sim_{Wu\&Palmer}(c_1, c_2) = \frac{2 \times N_0}{N_1 + N_2 + 2 \times N_0} \quad (4)$$

5.4 RP Semantic Metric

We propose to estimate the relative distance in a taxonomy between two concepts using the following intuitions. We use Figure 2 to illustrate these intuitions.

- Parent versus grandparent: Parent of a node is more similar to the node than grandparents of that. Generalization of

a concept reasonably results in going further away 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 it's sibling then to its grandparent. To illustrate, *AnyWineColor* is grandparent of *Red*, and *Red* and *Rose* are siblings. Therefore, we possibly anticipate that *Red* and *Rose* are more similar than *AnyWineColor* and *Red*.

As a taxonomy is represented in a tree, that tree can be traversed from the first concept being compared through the second concept. At starting node related to the first concept, the similarity value is constant and equal to one. This value is diminished by a constant at each node being visited over the path that will reach to the node including the second concept. The shorter the path between the concepts, the higher the similarity between nodes.

Algorithm 2 Estimate-RP-Similarity(c_1,c_2)

Require: The constants should be $m > n > m^2$ where $m, n \in R[0, 1]$
1: $Similarity \leftarrow 1$
2: **if** c_1 is equal to c_2 **then**
3: Return $Similarity$
4: **end if**
5: $commonParent \leftarrow findCommonParent(c_1, c_2)$ {$commonParent$ is the most specific concept that covers both c_1 and c_2}
6: $N1 \leftarrow findDistance(commonParent, c_1)$
7: $N2 \leftarrow findDistance(commonParent, c_2)$ {$N1$ & $N2$ are the number of links between the concept and parent concept}
8: **if** ($commonParent$ == c_1) or ($commonParent$ == c_2) **then**
9: $Similarity \leftarrow Similarity * m^{(N1+N2)}$
10: **else**
11: $Similarity \leftarrow Similarity * n * m^{(N1+N2-2)}$
12: **end if**
13: Return $Similarity$

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. Algorithm 2 shows the distance calculation.

According to the algorithm, firstly the similarity is initialized with the value of one (line 1). If the concepts are equal to each other then, similarity will be one (lines 2-4). Otherwise, we compute the common parent of the two nodes and the distance of each concept to the common parent without considering the sibling (lines 5-7).

If one of the concepts is equal to the common parent, then there is no sibling relation between the concepts. For each level, we multiply the similarity by m and do not consider the sibling factor in the similarity estimation. As a result, we decrease the similarity at each level with the rate of m (line9). Otherwise, there has to be a sibling relation. This means that we have to consider the effect of n when measuring similarity. Recall that we have counted N_1+N_2 edges between the concepts. Since there is a sibling relation, two of these edges constitute the sibling relation. Hence, when calculating the effect of the parent relation, we use N_1+$N_2 - 2$ edges (line 11).

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

Table 2: Sample similarity estimation over sample taxonomy

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

For all semantic similarity metrics in our architecture, the taxonomy for features is held in the shared ontology. In order to evaluate the similarity of feature vector, we firstly estimate the similarity for feature one by one and take the average sum of these similarities. Then the result is equal to the average semantic similarity of the entire feature vector.

6. DEVELOPED SYSTEM

We have implemented our architecture in Java. To ease testing of the system, the consumer agent has a user interface that allows us to enter various requests. The producer agent is fully automated and the learning and service offering operations work as explained before. In this section, we explain the implementation details of the developed system.

We use OWL [11] as our ontology language and JENA as our ontology reasoner. The shared ontology is the modified version of the *Wine Ontology* [19]. It includes the description of wine as a concept and different types of wine. All participants of the negotiation use this ontology for understanding each other. According to the ontology, seven properties make up the wine concept. The consumer agent and the producer agent obtain the possible values for the these properties by querying the ontology. Thus, all possible values for the components of the wine concept such as color, body, sugar and so on can be reached by both agents. Also a variety of wine types are described in this ontology such as *Burgundy*, *Chardonnay*, *CheninBlanc* and so on. Intuitively, any wine type described in the ontology also represents a wine concept. This allows us to consider instances of *Chardonnay* wine as instances of *Wine* class.

In addition to wine description, the hierarchical information of some features can be inferred from the ontology. For instance, we can represent the information *Europe Continent* covers *Western Country*. *Western Country* covers *French Region*, which covers some territories such as *Loire*, *Bordeaux* and so on. This hierarchical information is used in estimation of semantic similarity. In this part, some reasoning can be made such as if a concept X covers Y and Y covers Z, then concept X covers Z. For example, *Europe Continent* covers *Bordeaux*.

For some features such as body, flavor and sugar, there is no hierarchical information, but their values are semantically leveled. When that is the case, we give the reasonable similarity values for these features. For example, the body can be *light*, *medium*, or *strong*. In this case, we assume that *light* is 0.66 similar to *medium* but only 0.33 to *strong*.

WineStock Ontology is the producer's inventory and describes a product class as *WineProduct*. This class is necessary for the producer to record the wines that it sells. Ontology involves the individuals of this class. The individuals represent available services that the producer owns. We have prepared two separate *WineStock* ontologies for testing. In the first ontology, there are 19 available wine products and in the second ontology, there are 50 products.

7. PERFORMANCE EVALUATION

We evaluate the performance of the proposed systems in respect to learning technique they used, *DCEA* and *ID3*, by comparing them with the *CEA*, *RO* (for random offering), and *SCR* (offering based on current request only).

We apply a variety of scenarios on this dataset in order to see the performance differences. Each test scenario contains a list of preferences for the user and number of matches from the product list. Table 3 shows these preferences and availability of those products in the inventory for first five scenarios. Note that these preferences are internal to the consumer and the producer tries to learn these during negotiation.

Table 3: Availability of wines in different test scenarios

ID	Preference of consumer	Availability (out of 19)
1	Dry wine	15
2	Red and dry wine	8
3	Red, dry and moderate wine	4
4	Red and strong wine	2
5	Red or rose, and strong	3

7.1 Comparison of Learning Algorithms

In comparison of learning algorithms, we use the five scenarios in Table 3. Here, first we use Tversky's similarity measure. With these test cases, we are interested in finding the number of iterations that are required for the producer to generate an acceptable offer for the consumer. Since the performance also depends on the initial request, we repeat our experiments with different initial requests. Consequently, for each case, we run the algorithms five times with several variations of the initial requests. In each experiment, we count the number of iterations that were needed to reach an agreement. We take the average of these numbers in order to evaluate these systems fairly. As is customary, we test each algorithm with the same initial requests.

Table 4 compares the approaches using different learning algorithm. When the large parts of inventory is compatible with the customer's preferences as in the first test case, the performance of all techniques are nearly same (e.g., Scenario 1). As the number of compatible services drops, RO performs poorly as expected. The second worst method is SCR since it only considers the customer's most recent request and does not learn from previous requests. CEA gives the best results when it can generate an answer but cannot handle the cases containing disjunctive preferences, such as the one in Scenario 5. ID3 and DCEA achieve the best results. Their performance is comparable and they can handle all cases including Scenario 5.

Table 4: Comparison of learning algorithms in terms of average number of interactions

Run	DCEA	SCR	RO	CEA	ID3
Scenario 1:	1.2	1.4	1.2	1.2	1.2
Scenario 2:	1.4	1.4	2.6	1.4	1.4
Scenario 3:	1.4	1.8	4.4	1.4	1.4
Scenario 4:	2.2	2.8	9.6	1.8	2
Scenario 5:	2	2.6	7.6	1.75+ No offer	1.8
Avg. of all cases:	**1.64**	**2**	**5.08**	**1.51**+No offer	**1.56**

7.2 Comparison of Similarity Metrics

To compare the similarity metrics that were explained in Section 5, we fix the learning algorithm to DCEA. In addition to the scenarios shown in Table 3, we add following five new scenarios considering the hierarchical information.

- The customer wants to buy wine whose winery is located in California and whose grape is a type of white grape. Moreover, the winery of the wine should not be expensive. There are only four products meeting these conditions.
- The customer wants to buy wine whose color is red or rose and grape type is red grape. In addition, the location of wine should be in Europe. The sweetness degree is wished to be dry or off dry. The flavor should be delicate or moderate where the body should be medium or light. Furthermore, the winery of the wine should be an expensive winery. There are two products meeting all these requirements.
- The customer wants to buy moderate rose wine, which is located around French Region. The category of winery should be Moderate Winery. There is only one product meeting these requirements.
- The customer wants to buy expensive red wine, which is located around California Region or cheap white wine, which is located in around Texas Region. There are five available products.
- The customer wants to buy delicate white wine whose producer in the category of Expensive Winery. There are two available products.

The first seven scenarios are tested with the first dataset that contains a total of 19 services and the last three scenarios are tested with the second dataset that contains 50 services.

Table 5 gives the performance evaluation in terms of the number of interactions needed to reach a consensus. Tversky's metric gives the worst results since it does not consider the semantic similarity. Lin's performance are better than Tversky but worse than others. Wu Palmer's metric and RP similarity measure nearly give the same performance and better than others. When the results are examined, considering semantic closeness increases the performance.

8. DISCUSSION

We review the recent literature in comparison to our work. Tama *et al.* [16] propose a new approach based on ontology for negotiation. According to their approach, the negotiation protocols used in e-commerce can be modeled as ontologies. Thus, the agents can perform negotiation protocol by using this shared ontology without the need of being hard coded of negotiation protocol details. While

Table 5: Comparison of similarity metrics in terms of number of interactions

Run	Tversky	Lin	Wu Palmer	RP
Scenario 1:	1.2	1.2	1	1
Scenario 2:	1.4	1.4	1.6	1.6
Scenario 3:	1.4	1.8	2	2
Scenario 4:	2.2	1	1.2	1.2
Scenario 5:	2	1.6	1.6	1.6
Scenario 6:	5	3.8	2.4	2.6
Scenario 7:	3.2	1.2	1	1
Scenario 8:	5.6	2	2	2.2
Scenario 9:	2.6	2.2	2.2	2.6
Scenario 10:	4.4	2	2	1.8
Average of all cases:	**2.9**	**1.82**	**1.7**	**1.76**

Tama *et al.* model the negotiation protocol using ontologies, we have instead modeled the service to be negotiated. Further, we have built a system with which negotiation preferences can be learned.

Sadri *et al.* study negotiation in the context of resource allocation [14]. Agents have limited resources and need to require missing resources from other agents. A mechanism which is based on dialogue sequences among agents is proposed as a solution. The mechanism relies on observe-think-action agent cycle. These dialogues include offering resources, resource exchanges and offering alternative resource. Each agent in the system plans its actions to reach a goal state. Contrary to our approach, Sadri *et al.*'s study is not concerned with learning preferences of each other.

Brzostowski and Kowalczyk propose an approach to select an appropriate negotiation partner by investigating previous multi-attribute negotiations [1]. For achieving this, they use case-based reasoning. Their approach is probabilistic since the behavior of the partners can change at each iteration. In our approach, we are interested in negotiation the content of the service. After the consumer and producer agree on the service, price-oriented negotiation mechanisms can be used to agree on the price.

Fatima *et al.* study the factors that affect the negotiation such as preferences, deadline, price and so on, since the agent who develops a strategy against its opponent should consider all of them [5]. In their approach, the goal of the seller agent is to sell the service for the highest possible price whereas the goal of the buyer agent is to buy the good with the lowest possible price. Time interval affects these agents differently. Compared to Fatima *et al.* our focus is different. While they study the effect of time on negotiation, our focus is on learning preferences for a successful negotiation.

Faratin *et al.* propose a multi-issue negotiation mechanism, where the service variables for the negotiation such as price, quality of the service, and so on are considered traded-offs against each other (i.e., higher price for earlier delivery) [4]. They generate a heuristic model for trade-offs including fuzzy similarity estimation and a hill-climbing exploration for possibly acceptable offers. Although we address a similar problem, we learn the preferences of the customer by the help of inductive learning and generate counter-offers in accordance with these learned preferences. Faratin *et al.* only use the last offer made by the consumer in calculating the similarity for choosing counter offer. Unlike them, we also take into account the previous requests of the consumer. In their experiments, Faratin *et al.* assume that the weights for service variables are fixed a priori. On the contrary, we learn these preferences over time.

In our future work, we plan to integrate ontology reasoning into the learning algorithm so that hierarchical information can be learned from subsumption hierarchy of relations. Further, by using relationships among features, the producer can discover new knowledge from the existing knowledge. These are interesting directions that we will pursue in our future work.

9. REFERENCES

[1] J. Brzostowski and R. Kowalczyk. On possibilistic case-based reasoning for selecting partners for multi-attribute agent negotiation. In *Proceedings of the 4th Intl. Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 273–278, 2005.

[2] L. Busch and I. Horstman. A comment on issue-by-issue negotiations. *Games and Economic Behavior*, 19:144–148, 1997.

[3] J. K. Debenham. Managing e-market negotiation in context with a multiagent system. In *Proceedings 21st International Conference on Knowledge Based Systems and Applied Artificial Intelligence, ES'2002:*, 2002.

[4] P. Faratin, C. Sierra, and N. R. Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *Artificial Intelligence*, 142:205–237, 2002.

[5] S. Fatima, M. Wooldridge, and N. Jennings. Optimal agents for multi-issue negotiation. In *Proceeding of the 2nd Intl. Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 129–136, 2003.

[6] C. Giraud-Carrier. A note on the utility of incremental learning. *AI Communications*, 13(4):215–223, 2000.

[7] T.-P. Hong and S.-S. Tseng. Splitting and merging version spaces to learn disjunctive concepts. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):813–815, 1999.

[8] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[9] P. Maes, R. H. Guttman, and A. G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, 1999.

[10] T. M. Mitchell. *Machine Learning*. McGraw Hill, NY, 1997.

[11] OWL. OWL: Web ontology language guide, 2003. http://www.w3.org/TR/2003/CR-owl-guide-20030818/.

[12] S. K. Pal and S. C. K. Shiu. *Foundations of Soft Case-Based Reasoning*. John Wiley & Sons, New Jersey, 2004.

[13] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[14] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: Agent varieties and dialogue sequences. In *ATAL 2001, Revised Papers*, volume 2333 of *LNAI*, pages 405–421. Springer-Verlag, 2002.

[15] M. P. Singh. Value-oriented electronic commerce. *IEEE Internet Computing*, 3(3):6–7, 1999.

[16] V. Tamma, S. Phelps, I. Dickinson, and M. Wooldridge. Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence*, 18:223–236, 2005.

[17] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977.

[18] P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[19] Wine, 2003. http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine.rdf.

[20] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133 –138, 1994.