EVALUATING AND RANKING SEMANTIC OFFERS ACCORDING TO USERS' INTERESTS

Wei Jiang
Computer Science Department
University of Regina
3737 Wascana Parkway, Regina, Saskatchewan
Canada, S4S 0A2
jiang20w@uregina.ca

Samira Sadaoui
Computer Science Department
University of Regina
3737 Wascana Parkway, Regina, Saskatchewan
Canada, S4S 0A2
sadaouis@uregina.ca

ABSTRACT

Semantic matchmaking systems are not successful in identifying the differences between users' interests. To address this weakness, we develop a user-oriented and personalized system to evaluate the offers that match the user's request. Our system evaluates and ranks the offers according to the user's specific interests to bring better results to each individual. The best offer represents the maximum satisfaction of the user. The proposed system extracts and analyzes the user's interests for multiple offer attributes. To evaluate the offers, we adapt the well-known economic model MultiNomial Logit to the field of semantic matchmaking. We show the benefits of our offer evaluation system through a detailed case study involving multiple, high-dimensional, concept and value-based attributes. In this study, we show how our system catches the differences between the purchasing interests of two buyers, and how it recommends a different best offer to each buyer. Furthermore, we assess the feasibility of the proposed system with a transport usage dataset. The experiment results demonstrate that our system can provide good result for each individual.

Keywords: MultiNomial Logit model, clustering technology SOM, semantic matchmaking, interest model, high-dimensional offer attributes.

1. Introduction

Semantic matchmaking is one of the most important processes of e-commerce. Nowadays with the blooming of e-commerce and e-services, users may obtain a large number of request-matched offers. It is really time consuming for users to evaluate and sort the entire candidate offers in order to find the best offer. Returning a ranked list of offers is a very important task for semantic matchmaking systems. And recent matchmakers [Di Noia et al. 2007a, Di Noia et al. 2007b, Bener et al. 2009, Skoutas et al. 2010] realize this importance. Our research work has the same goal as matchmakers: rank the offers to determine the best offer. As illustrated in Figure 1, we are not improving any matching work but only the ranking work for the matchmakers. Semantic matchmakers usually rely on common, standard ontologies to match and rank the offers. However in each domain, each individual may have different knowledge background i.e. different ontology to judge the offers. Using a common and public ontology does not take into account the users' differences. Furthermore, changes in the users' interests cannot be expressed in ontology. For instance, PC was the most sold item in the past, but during these last five years, lots of customers turned to Apple products**.** Ontology cannot track the shift of users' interests. The problems arising from semantic matchmaking and ranking are the following:

- Using common, general, un-personalized criteria [Di Noia et al. 2007a, Di Noia et al. 2007b, Bener et al. 2009**]** to evaluate the offers will not produce a personalized result. Searching a "common best offer" is not equal to solving each individual case since each individual is special. [Park & Gretzel 2010] shows that the individual's choice is related to his own personal style**.** Moreover, most semantic matchmakers are satisfied

with their research work with only one user, and they do not consider whether the common best result could satisfy other users or not.

- Evaluating the offers with multiple attributes is a challenging task. Bringing user-defined weights for the offer attributes into the evaluation function is not easy and not reliable [Skoutas et al. 2010]. In addition, users may have many trade-off situations when they evaluate complex offers [Skoutas et al. 2010]. Some trade-offs may occur between the attributes. For example, let us consider two computer attributes: category and price. For most semantic matchmakers, like [Di Noia et al. 2007a, Bener et al. 2009**],** both categories Mac and PC have equal ranking values, and the PC price is ranked better than the MAC price because it is lower. Still, some users prefer to pay more money to buy a Mac but not when buying a PC. This means there is a trade-off between category and price. In this case, matchmakers cannot explain why the user selected a higher price instead of a lower one. Trade-offs can also occur between the offer attributes and unobserved attributes [McFadeen 2001]. This issue is widely discussed in the economic choice area [McFadeen 2001] but still very few e-commerce technologies, like semantic matchmaking, have applied the economic ideas. For instance, the price attribute can be affected by other unobserved factors, such as the user's emotion, feeling and salary. As a result**,** the user is willing to pay more in a certain price range (for example $1 - $100) but not in other price ranges (like $900-$999) depending on his salary or other factors. All these complex cases cannot be considered by a simple weighing function [Skoutas et al. 2010]. Most semantic matchmakers still keep their experiments simple and with very few attributes, not really ready to the whole market.

- Dealing with high-dimensional offer attributes is not considered by existing semantic matchmakers, to the best of our knowledge.

In this paper, we address the problems above by providing a more user-oriented and personalized system to evaluate semantic offers (goods or web services). The proposed system determines the best offer by evaluating and sorting the offers according to the user's interests. We believe the user's specific interests are important for the evaluation because they are the real reasons explaining why an individual prefers a certain offer. We bring the interests into the ontology to be able to identify the differences between users' needs. Our system catches and analyzes the user's interest for each attribute separately. On one hand this greatly simplifies the offer evaluation with multiple attributes, and on the other hand we can avoid many trade-off situations. If the user is not satisfied with the results, he can always re-select better interests for the attributes. Last, our model can deal with high-dimensional attributes. Compared to existing semantic matchmakers, our evaluation system provides to the user a unique framework to handle multiple, high-dimensional, concept and value-based attributes.

To better illustrate the benefits of our system, we consider the following case: we have several users looking for the same service. In existing matchmakers, users who submit the same query get the exact same sorted list of offers. In Section 3, through a 5-attribute application, we demonstrate how our system catches the purchasing interests of each buyer, and how it recommends a different best offer to each buyer. Nowadays, matchmaking and ranking are important for advertisement matching [Essex 2009] and web-service matching, which are required in different business areas, such as Business-to-Business [Cheng et al. 2004] and Business-to-Customer [Guo 2008]. Our system may solve the unmatched-individual-interest problems by providing a personalized matchmaker that can help the user to find the offer that is the closest to his real needs.

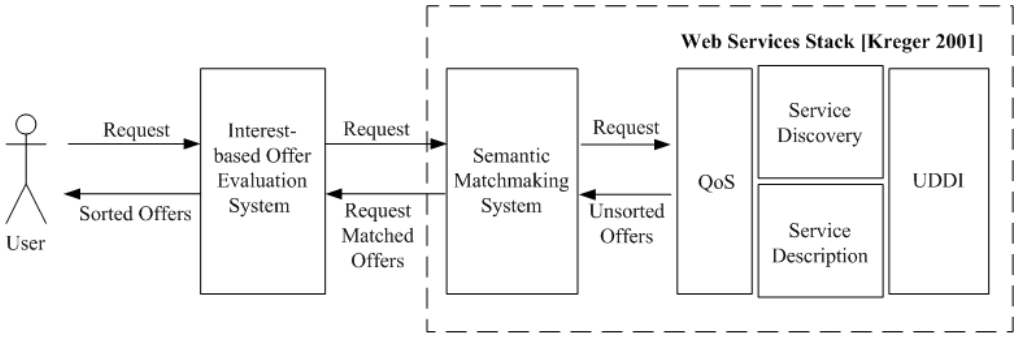


Figure 1: Semantic Matchmaker Improvement

In our system, first the user submits a request which is sent to the connected matchmaker. The latter returns a list of request-matched offers. To evaluate and sort these offers according to the user's interests, our system performs the following tasks: (1) converts the concept attributes to numeric information; (2) clusters the values of each attribute; (3) catches the user's interest for each attribute, (4) builds the user's interest model based on the interest weights and interest rates of the attributes; (5) applies the resulting model to evaluate the offers. We generate the attribute weights and rates according to the user's selected clustering (or interests). Different clustering selections will produce different interest models. In the case study of Section 3, our system catches the interests of two buyers, and generates a different interest model for each buyer. We may note that our system processes value-based attributes, and any concept attribute needs to be converted to a numeric information as it is done in some semantic matchmakers [Dong-wei et al. 2006, Qiu & Li 2008].

In this paper, we adopt the economic model called MultiNomial Logit (MNL) [McFadden 1974, McFadden 2001] to the field of semantic matchmaking. The MNL model is widely used in commerce and statistic areas to study human shopping behaviors [Yu et al. 2006, Xu et al. 2010]. Nevertheless, MNL focuses on the population taste and the individual taste is eliminated**.** We modify the MNL model to provide a new interest model that takes into consideration the individual taste i.e. the interests. This model evaluates the offers and returns the best offer w.r.t the individual's interests. The best offer has the highest interest value to the user. Furthermore, to take into account the user's interest for each attribute, we need to incorporate to our system the clustering technology called Self Organizing Map (SOM) [Kohonen 2001, Chen & Young 2005]. As a neural-network based approach, SOM is used to cluster high-dimensional inputs onto lower-dimensional outputs. The reason of using SOM in our work is that the offer attributes may be complex or contain high-dimensional data.

The rest of this paper is structured as follows. In Section 2, we present the distributed architecture of our system and also the different phases of our offer evaluation process. In Section 3, we explain in detail each phase through a 5-attribute application involving concept and high-dimensional attributes. In Section 4, we apply our system to a transport usage data set of 840 user selections of four types of travel transport. In Section 5, we describe some well known semantic matchmakers and compare them with our system. In Section 6, we report future work.

2. An Interest-based Offer Evaluation System

The proposed system is designed with a distributed architecture as depicted in Figure 2. First the user specifies his request with the web service language, such as WSDL [Chabeb et al. 2010] and WSML [Klusch et al. 2008], that is used by the connected matchmaker. The user submits the specified request via the GUI that transmits it to the matchmaker. The latter returns a list of candidate offers. On the server side, the *OfferManager* component: (1) collects the offers and stores them in the *OfferContent* database, (2) analyzes the request and offers to extract the attributes and their values, (3) stores the attribute data in the *OfferAttributes* database, and (4) calls *ConceptConvertor* to convert the concept attributes to numeric information. The *AttributeDataClustering* component clusters the values of each attribute and sends the resulting clustering to the client side. The user can now select his most interested clustering for each attribute. With these selections, our system will know the range and depth of the users' interests. These selections are then transmitted to *InterestWeightCalculator* and *InterestRateFunctionCreator* components. For each attribute, *InterestWeightCalculator* produces its interest weight and *InterestRateFunctionCreator* its interest rate function. The *OfferEvaluator* component: (1) builds the user's interest model based on the interest weights and interest rate functions, (2) evaluates all the candidate offers with the generated model, and (3) returns a list of sorted offers.

2.1. Converting Concept Attributes

Since our system processes only numeric attribute data, any concept data needs to be converted to a digital number. Matchmakers utilize different conversion methods, such as converting by distance [Dong-wei et al. 2006] or by similarity [Qiu & Li 2008]. In Algorithm 1, we define the function convertConcept() to perform the conversion by distance . This function converts the current concept called C_{Child} to a high-dimensional position in order to keep the same relation to the other concepts like in the ontology**.** First we generate the weight of C_{Child} and get the distance between C_{Child} and its concept parent in the ontological tree. Afterwards, for each level in the tree we produce two data, called *al* and *bl*, to store the 2-dimensional position of C_{Child} . The function findConcept() determines the position of C_{Child} in the ontological tree. It returns four variables: C_{Parent} the parent concept of $C_{Child;}$ n the number of children concepts of $C_{Parent;}$ l the level of the position of C_{Child} in the tree; i the index of C_{Child} as the child of $C_{Parent.}$.

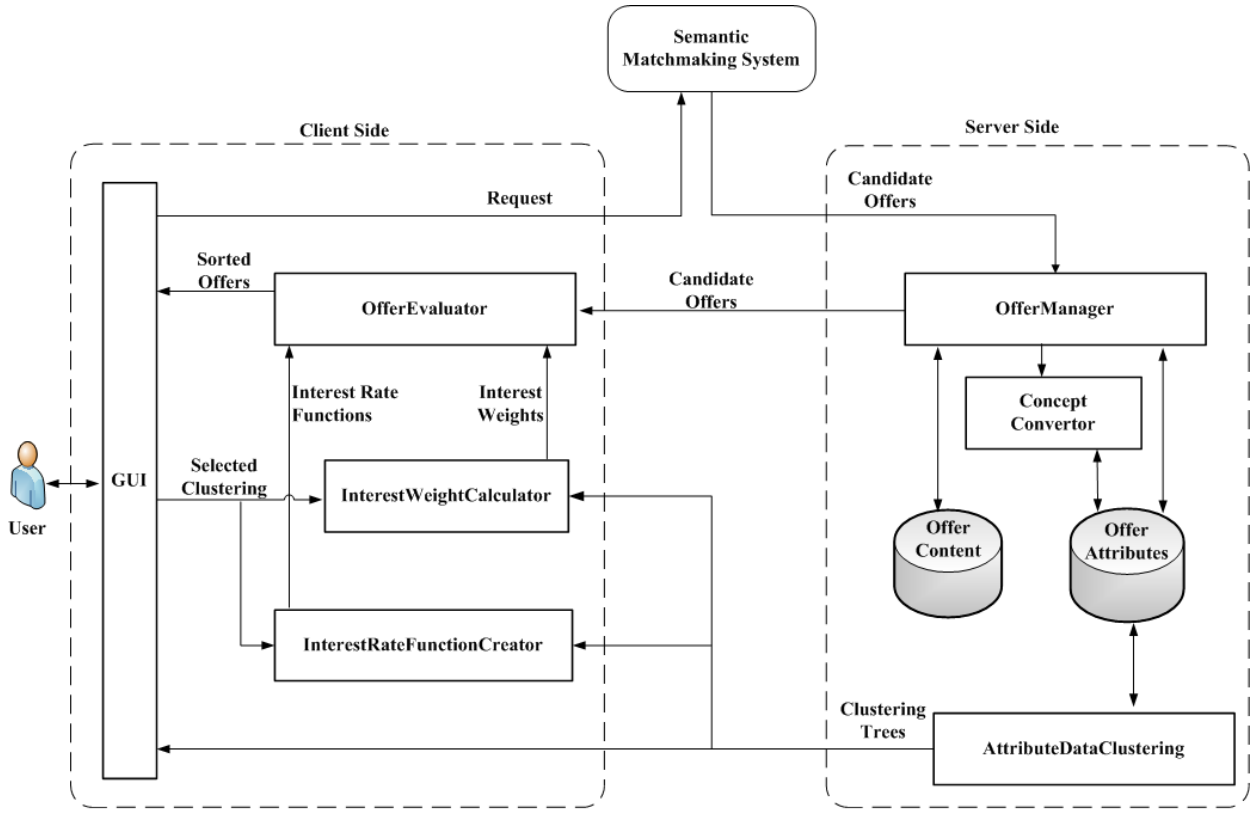


Figure 2: An Interest-based Offer Evaluation System

convertConcept(C_{Child}: String, Ontology: Tree, $hdpc_{Child}$:Array)
{ 1. $hdpc_{Child}$ = void; *//high-dimensional position of C_{child}*
do {

2. *findConcept* (C_{Child} , Ontology, C_{Parent} , n, l, i) ;
// determine the position of C_{Child}

3. $$weight(C_{Child}) = \begin{cases} 1 & \text{if } i = 0; \\ \dfrac{weight(C_{Child})}{n} & \text{if } i \neq 0; \end{cases}$$
// calculate the weight of C_{Child}

4. $\| C_{Child} - C_{Parent} \| = \| weight(C_{Child}) - weight(C_{Parent}) \|$;
// get distance between C_{Child} to C_{Parent}

5. $$\begin{cases} a_l(C_{Child}) = \| C_{Child} - C_{Parent} \| \cdot \sin(i \cdot \dfrac{-360}{n}); \\ b_l(C_{Child}) = \| C_{Child} - C_{Parant} \| \cdot \cos(i \cdot \dfrac{-360}{n}); \end{cases}$$
// get C_{Child} 2-dimensional position (a_l, b_l) at level *l*

6. add($hdpc_{Child}$, $a_l(C_{Child})$, $b_l(C_{Child})$)
// add the 2D position to hdpc

7. $C_{Child} = C_{Parent}$ // go up in the ontology

} while $l \neq 1$

8. return $hdpc_{Child}$;

}

Algorithm 1: Concept Conversion to a High-Dimensional Position

2.2. Clustering Attribute Data

The goal of *AttributeDataClustering* component is to cluster the values of each attribute and send the resulting clustering (represented as a tree structure) to the client side. Thanks to this clustering, our system is able to catch the user's interest for each attribute. The user is just required to select one of the attribute clustering to represent his most interested area. In Algorithm 2, we present the clustering function *clusterAttributeData()* that is based on the Self Organizing Map (SOM) [Kohonen 2001]. In our work, we apply SOM to recursively divide a large clustering into three sub-clustering until there are less data in each sub-clustering. The reason we let SOM divide a clustering into three groups is to arrange the attribute data as follows: a set of higher values, a set of lower values and a set of values in the middle. This division will help the user to better examine and understand his interests**.** We include the function *isLargeEnough()* to check whether a clustering A_i is still large enough or not. If the range of A_i is greater than the minimal range of the attribute data, then we need to cluster A_i.

In SOM of Algorithm 3, *createLVQ()* generates three random Learning Vector Quantizations (LVQ_i) in the range of *AttributeData*; *alpha (t)* controls the learning loop and decreases with the learning time. During the learning time, the function SOM() performs the competitive-learning given in step 3.1.1 and in step 3.1.3 updates the *LVQi* positions according to the competitive-learning results [Kohonen 2001]. Since SOM updates the neighborhood of each LVQ, we need a "neighborhood" function, *hci()*, to figure out how much the neighborhood of a LVQ can be updated. In Appendix A, we give the implementation of *SOM()* in C#.

```
clusterAttributeData (AttributeData: Array, ClusteringTree: Tree)
{
   1. MinRange = min (AttributeData);
     // calculate the minimum range
   2. SOM(AttributeData, A1, A2, A3);
      // cluster all data into 3 groups
   3. arrangeClustering (A1, A2, A3, ClusteringTree);
     // arrange clustering in ascending order
   4. for i = 1 to 3
         if ( isLargeEnough (Ai, MinRange) )
            clusterAttributeData(Ai, ClusteringTree);
            // cluster each sub-group
   5. return ClusteringTree;
 }
```

Algorithm 2: Attribute Data Clustering

```
SOM (AttributeData: Array, A1: Array, A2: Array, A3: Array)
{      1. t = 1;    //initialize learning time
       2. createLVQ (AttributeData, LVQ1, LVQ2, LVQ3);
          // create random LVQi
       3. while ( alpha(t) is not too small )
          // decrease alpha(t) by time t
          {  3.1 while ( pop (AttributeData, x) )
             // pop x from data set
             {  3.1.1 ||x - LVQc || = min{ ||x - LVQi ||};
                   // find closest LVQc to x
                3.1.2  add(x, Ac);
                   // add x into the closest cluster Ac
                3.1.3  LVQ(t+1) = LVQ(t) + hci(t)[x(t) - LVQ(t)];
                   // update LVQi to be closer to x
             }
             3.2   t = t+1; //increase time t
          }
       4. return A1, A2, A3;
 }
```

Algorithm 3: SOM

2.3. Computing Interest Weights of Offer Attributes

InterestWeightCalculator computes the interest weight of each attribute. This weight denotes the degree of importance of an attribute. The interest weight is related to the depth and range of the selected clustering of the user. The smallest and deepest clustering represents a more specific interest for the user. To produce the attribute's weight, we first determine the interest-weight coefficient of an attribute as shown in Formula **(1)** where: *k* is an attribute, *K* the attribute set, *AttributeData* the data of *k*, *SelectedClustering* the user's selected clustering for *k*, *SelectedLevelTree* the level of the selected clustering, and *TotalLevelTree* the number of levels in the clustering tree. The function *Length*() returns the length of the attribute interval i.e. the absolute difference between the two attribute's endpoints.

$$IW\_coe_k = \frac{\text{Length}(AttributeData_k)}{\text{Length}(SelectedClustering_k)} \cdot \frac{SelectedLevelTree_k}{TotalLevelTree_k} \quad k \in K \quad \textbf{(1)}$$

This coefficient has to be compared to the other attributes' coefficients to produce the interest weight for an attribute (cf. Formula **(2)**). An attribute with a larger coefficient gets a larger interest weight compared to the other attributes.

$$IW_k = \frac{IW\_coe_k}{\sum_{k=1}^{K} IW\_coe_k} \qquad k \in K \quad \textbf{(2)}$$

2.4. Computing Interest Rates of Offer Attributes

InterestRateFunctionCreator produces the interest rate function of an attribute according to the user's selection. A linear function is usually used to measure the attributes' rates [Ha & Park 2001, Yu et al. 2006, Huang et al. 2008]. Nevertheless, in some cases, linear utility functions cannot assign weights to attributes in order to make a certain offer as the best offer. We explain this issue with the example of Table 1. We are looking for the best stock among three available stocks: Stock1 is low risk with low return, Stock3 is high risk with high return and Stock2 is in the middle.

Table 1: Stock Data Example

Stock	**Risk**	**Return**
Stock1	0.1	0.1
Stock2	0.3	0.3
Stock3	1	1

Let us assume that the user considers Stock2 as the best offer. A linear utility function has to determine the weights for the two attributes "Risk" and "Return". However, we demonstrate below that these weights do not exist when we want to make Stock2 as the best offer. Linear utility functions cannot make sure that each offer has a chance to be the best one.

$$\begin{cases} \text{Stock2 is better than Stock1} \\ \text{Stock2 is better than Stock3} \end{cases}$$

$$\Rightarrow \begin{cases} -0.3W_{Risk} + 0.3W_{Return} > -0.1W_{Risk} + 0.1W_{Return} \\ -0.3W_{Risk} + 0.3W_{Return} > -1W_{Risk} + 1W_{Return} \end{cases}$$

$$\Rightarrow \begin{cases} 0.2\ W_{Risk} < 0.2W_{Return} \\ 0.7W_{Risk} > 0.7W_{Return} \end{cases}$$

$$\Rightarrow W_{Risk} \text{ and } W_{Return} \text{ do not exist}$$

To solve this problem, we use un-linear functions, such as the sigmoid function $\varsigma(x) = \frac{1}{1+\exp^{-x}}$. This function has few changes in the two intervals [-∞, -2] and [2, +∞]. Thus, in these two intervals, we can consider it as a linear function. However, the interval [-2, 2] is the quickly changeable area. The user's selected clustering represents his interest for an attribute. In order to consider such interest in our interest rate function, we need to bind the user's selected clustering into the changeable interval [-2, 2] (cf. Figure3).

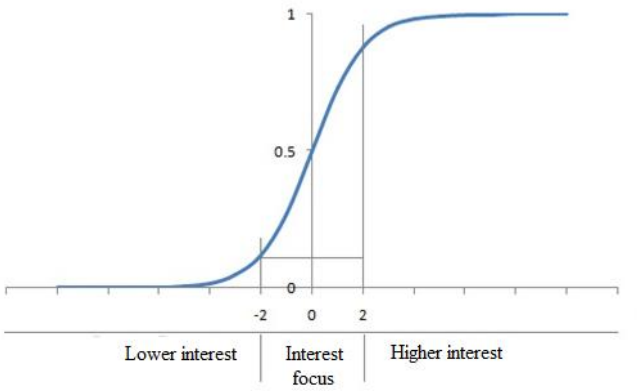


Figure 3**:** Sigmoid Function

Herein we explain how to produce the interest rate function for an attribute *k*, called $\varsigma_k()$. In Formula **(3)**, *x* indicates the value of an attribute; [AL, AR] the whole attribute data and [L, R] the selected clustering. We utilize LVQ as the center of the function $\varsigma_k()$**.** LVQ is computed with the learning loops of the SOM algorithm. In fact, LVQ point can be either in the selected clustering [L, R] or outside:

- If LVQ is inside [L, R], we decompose the interest rate function into two functions: $\varsigma_{k\_Right}()$ and $\varsigma_{k\_Left}()$ as shown in Formula **(3)** where: α_{Left} is generated when binding [L, LVQ] into the interval [-2, 0]; α_{Right} when binding [LVQ, R] into [0, 2]**.** α_{Left} controls the shape of the left side of the rate function, and α_{Right} its right side. *Sign* is +1 or-1 depending if the interest rate function is increasing or decreasing (cf. Table2).

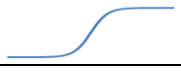
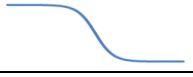
- If LVQ is outside of [L, R], the interest rate function can be either $\varsigma_{k\_Right}()$ or $\varsigma_{k\_Left}()$ depending on the LVQ position: left outside or right outside of the selected clustering. For example, if LVQ is on the right side of [L, R], we let $\varsigma_{k\_Left}()$ control the whole shape of the function since $\alpha_{Right}()$ cannot be generated.

$$\varsigma_k(x) = \begin{cases} \varsigma_{k\_Left}(x) = \dfrac{1}{1+\exp^{(\alpha_{Left}\cdot Sign\cdot|x-LVQ|)}} & x \in [AL, LVQ] \\ \quad and\ /\ or & \\ \varsigma_{k\_Right}(x) = \dfrac{1}{1+\exp^{(\alpha_{Right}\cdot Sign\cdot|x-LVQ|)}} & x \in [LVQ, AR] \end{cases} \quad \textbf{(3)}$$

$$\alpha_{Left}\cdot|x - LVQ| = -1\cdot(-2-0) \quad \text{when } x = L,\ L \text{ is binded to } -2$$
$$\Rightarrow \quad \alpha_{Left} = 2/|L - LVQ|$$

$$\alpha_{Right}\cdot|x - LVQ| = -1\cdot(2-0) \quad \text{when } x = R,\ R \text{ is binded to } 2$$
$$\Rightarrow \quad \alpha_{Right} = -2/|R - LVQ|$$

Table 2: Interest Rate Function Shape

Sign	Function and Image	
+	$\varsigma_k(x) = \dfrac{1}{1+\exp^{-x}}$	
-	$\varsigma_k(x) = \dfrac{1}{1+\exp^{x}}$	

2.5. Evaluating the Offers with the Interest Model

The MNL model represents the utility function of an individual in a population. It includes the deterministic and random components [McFadden 2001] as follows: $U_{nj} = V_{nj} + \varepsilon_{nj}, j \in C$

where:

- U_{nj} is the utility of the individual *n* selecting item *j;*
- V_{nj} is the "representative" or common taste of the population for item j;
- ε_{nj} is the individual taste for item *j*;
- C is the set of items.

In the formula below**,** the component V_{nj} consists of *K* observed deterministic features. V_{nj} contains the weight b_k for each feature x_{njk} [McFadden 2001]**:**

$$U_{nj} = \sum_{k=1}^{K} b_k \cdot x_{njk} + \varepsilon_{nj}, \quad j \in C$$

In our work, we still follow the idea of the MNL model. First we consider the deterministic features as the offer attributes. Furthermore, we can decompose ε_{nj} into *K* attributes since ε_{nj} is the total alternative with *K* features. This means the evaluation of the attributes may be different according to the interests of each individual. At the market level, the individual taste ε is brought into the population model as a random utility [McFadden 1974]. Since it comes from individuals and its value is random, ε is usually removed from the MNL function [Yu et al. 2006, Xu et al. 2010]. However in our model, the individual taste becomes important. The population taste cannot cover each user's specific interests. By including the individual taste, we have a better chance to produce real and personalized results. We propose Formula **(4)** to build our interest model.

$$\mathrm{IM}_{nj} = \sum_{k=1}^{K} (\mathrm{IW}_{njk} \cdot \mathrm{IR}_{njk}), \quad j \in O \quad \textbf{(4)}$$

where:

- IM_{nj} is the degree of interest of the user *n* selecting offer *j* with *K* attributes;
- IW_{jk} is the interest weight *(i.e.* b_k) of attribute k of offer j;
- IR_{jk} is the interest rate (i.e. $x_{jk} + \varepsilon_j$) of attribute k of offer j;
- O is the set of offers.

The *OfferEvaluator* component first builds the user's interest model with Formula **(4).** Then it evaluates all the candidate offers and gets their interest values. The offers are then sorted according to their interest values.

Table 3: Candidate Offers

Offer ID	Category	CPU (GHz)	RAM (GB)	Hard Drive (GB)	Price ($)
1	Desktop	2.2	2	320	(420, 400)*
2	Netbook	2.2	3	640	(470, 370) *
3	Laptop	2.5	4	500	(600, 500) *
4	Everyday	2.33	8	1310	(1100, 800) *
5	Performance	2.4	8	750	(999, 799) *
6	Game	2.5	6	750	(1030, 830) *
7	All-in-one	2.66	12	1024	(2500, 2200) *
8	Server	2.3	12	960	(1000, 880) *
9	iPad	2.5	6	820	(1100, 799) *
10	Tablet	2.4	4	1200	(950, 900) *
11	Apple	2.8	6	620	(1200, 1150) *
12	Mac min	(1.9, 1.9) *	10	1000	(800, 700) *
13	Mac Book	(3.0, 3.0) *	16	1500	(2900, 2600) *
14	Mac Pro	(1.8, 1.8)*	9	160	(680, 680) *
15	Mac Air	(3.2, 3.2)*	12	2000	(3200, 2500)*

*: Two-dimensional Data

3. A Case Study with two Buyers

This experiment consists of purchasing computers based on five attributes: Category, CPU, RAM, Hard Drive and Price. We assume here we have two buyers: Buyer1 and Buyer2, who submit the same request:

Purchase a computer with CPU > 1.5 GHz, RAM >= 2.0 GB, Hard Drive > 100 GB, Price < $3500 for the first ten purchased computers and Price < $3000 for the next ten.

We suppose the connected matchmaker returns the candidate offers given in Table 3 where Category is a concept attribute, CPU and Price contain high-dimensional values. In the next phases, we show how our system catches the purchasing interests of each buyer and determines the best offer.

3.1. Converting Concept Attributes

In Figure 4, we create the ontology of the attribute Category based on "bestbuy.com".

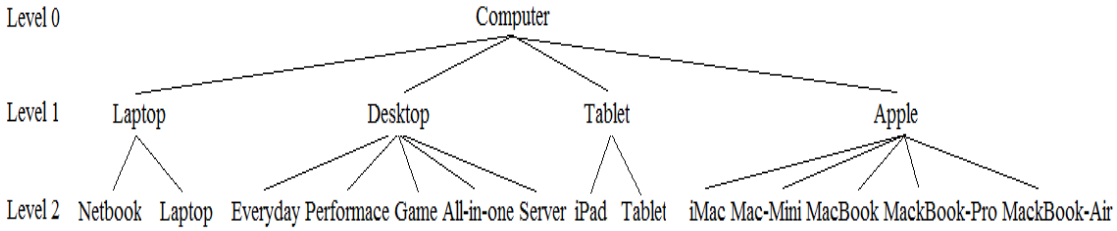


Figure 4: Computer Ontology

By following Algorithm 1 of Section 2, we first compute the weight of each computer category and then its distance to its parent node. In Table 4, we give the weight, distance and high-dimensional position of some categories. In Figure 5, we show how to produce the dimensional position of a concept.

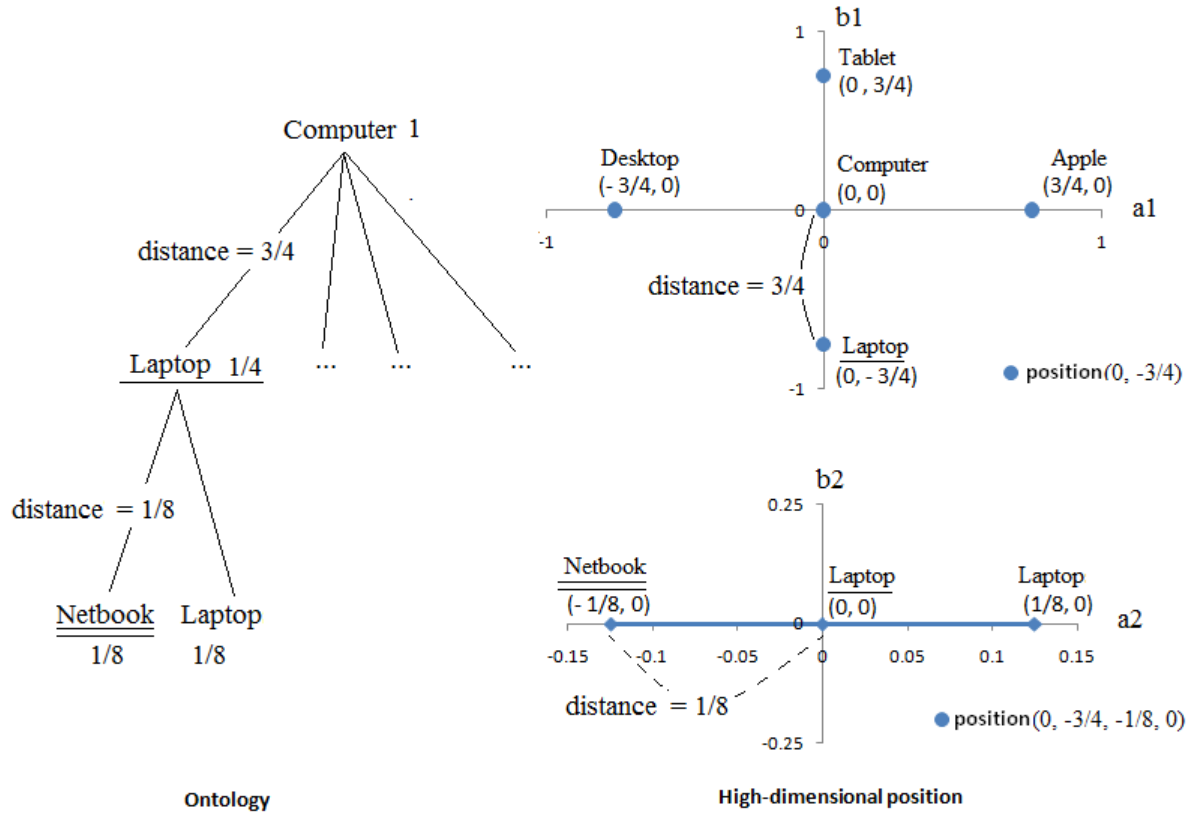


Figure 5: Concept Conversion

Table 4: Examples of Concept Conversion

Category	Weight	Distance	High-dimensional Position (a1, b1, a2, b2)
Computer	1	1 - 1 = 0	0, 0, 0, 0
Laptop	1/4	1- 1/4 = 3/4	0, -0.75, 0, 0
Desktop	1/4	1- 1/4 = 3/4	-0.75, 0, 0, 0
Tablet	1/4	1- 1/4 = 3/4	0, 0.75, 0, 0
Apple	1/4	1- 1/4 = 3/4	0.75, 0, 0, 0
Netbook	1/8	1/4-1/8 = 1/8	0, -0.75, -0.125, 0
Laptop	1/8	1/4-1/8 = 1/8	0, -0.75, 0.125, 0

3.2. Clustering Attribute Data

Our system displays via the GUI the clustering tree of each attribute (cf. Figure 9). Figures 6, 7 and 8 illustrate respectively the clustering of Hard Drive, CPU and Category. For the Category attribute, we give the concept view, and in Appendix B the position view. Since LVQs are randomly initialized (cf. Algorithm 3), sometimes one or more LVQs may be far away from the attribute data. Consequently, the corresponding clustering will not include any value. That is why in Figures 6, 7 and 8, some clustering have less than three sub-clustering.

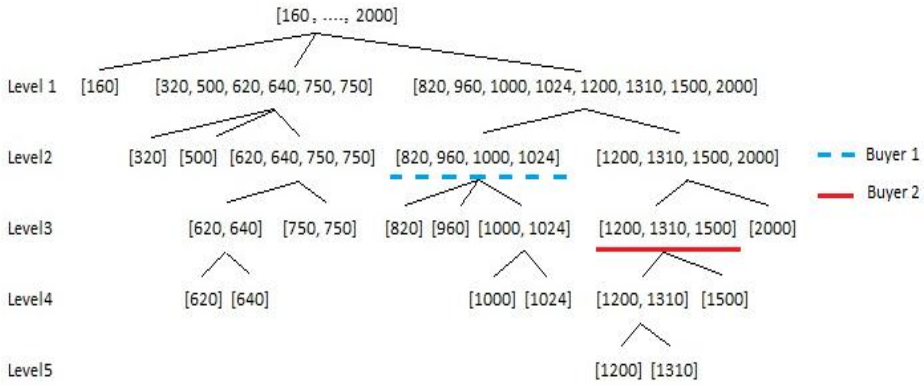


Figure 6: Hard-Drive Data Clustering

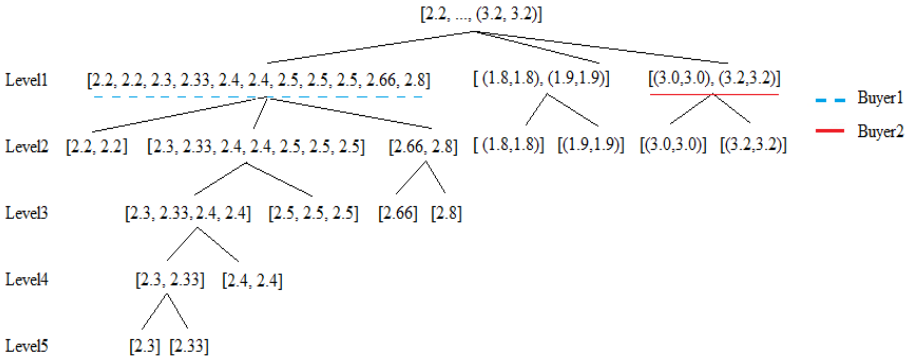


Figure 7: CPU Data Clustering

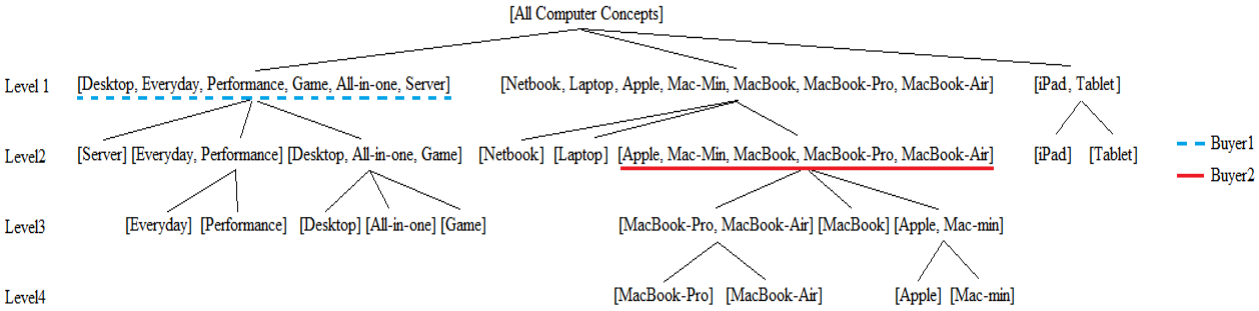


Figure 8: Category Data Clustering (Concept View)

Each buyer can now select his most interested clustering for each attribute as shown in Figure 9 for Buyer1. With these selections, our system will know the range and depth of each buyer's purchasing interests. Table 5 is an example of the interests of Buyer1 and Buyer2. For CPU and RAM, both buyers selected a clustering in level 1 because they have no special interests in these two attributes. Buyer1 may have a less interest in CPU than Buyer2 since he chose a larger clustering range. For the same reason, Buyer2 may have less interest in RAM. Buyer1 focuses on lower price and Hard Drive. Buyer2 is the opposite buyer who wants a higher Hard Drive with an acceptable price. Moreover, Buyer1 is looking for a desktop computer while Buyer2 is interested in Apple products.

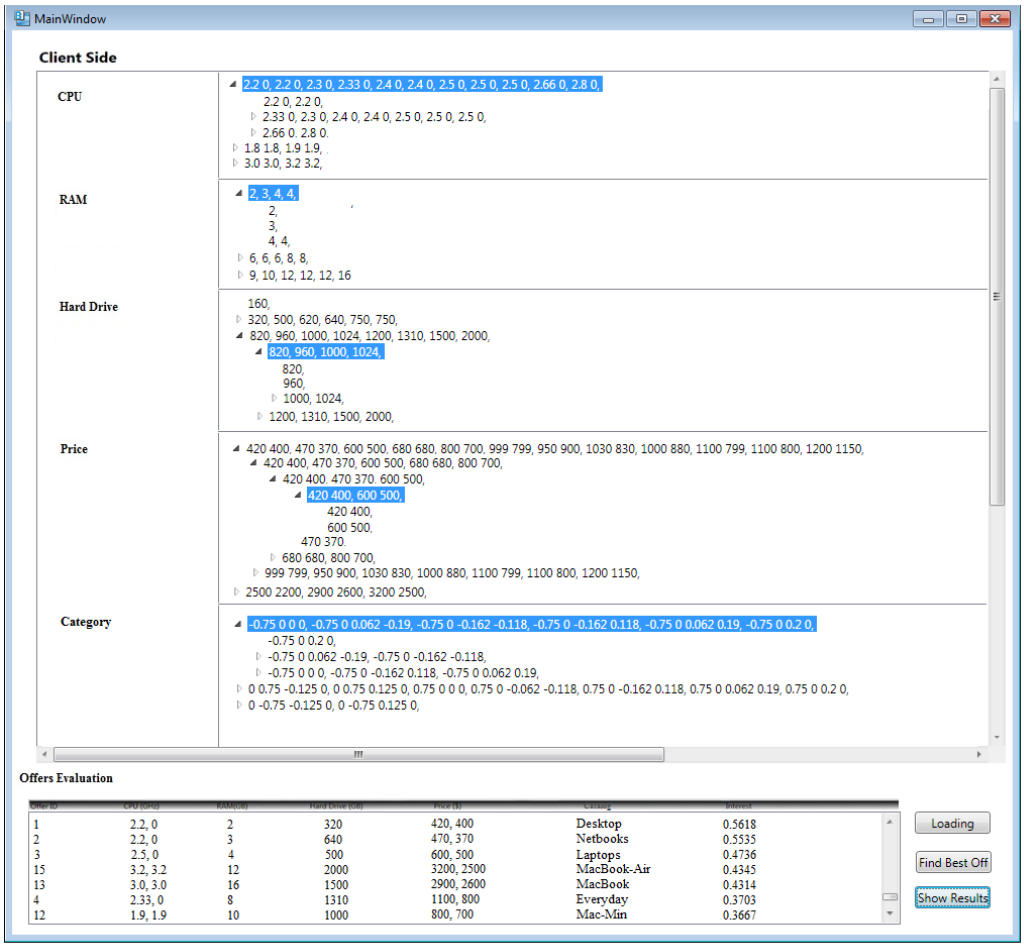


Figure 9: Selected Clustering of Buyer1

Table 5: Selections of Both Buyers

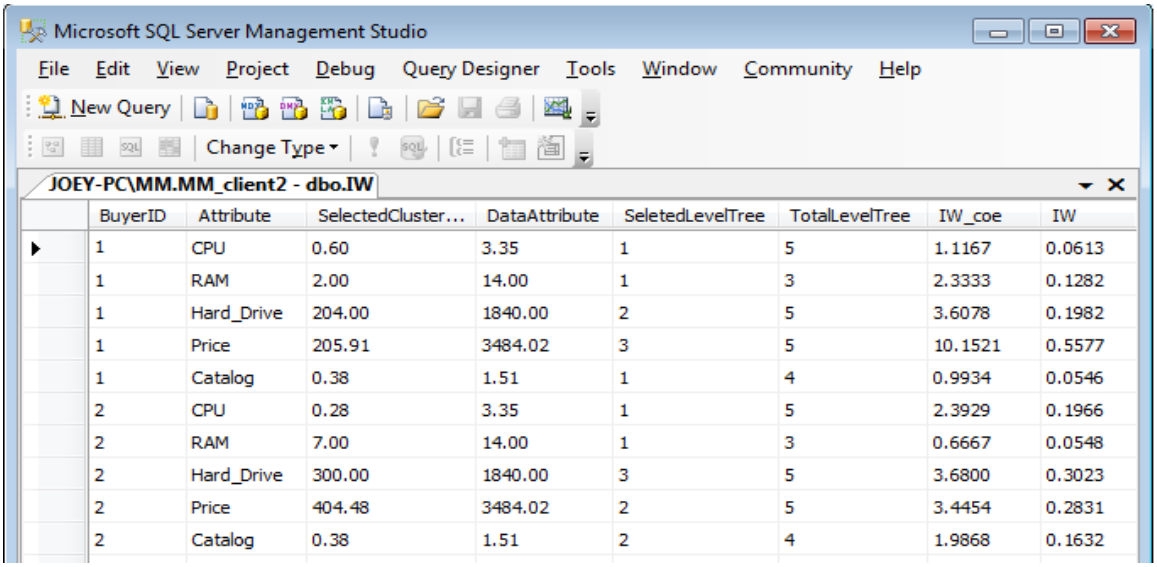
		Buyer1	Buyer2
CPU	**Clustering**	[2.2, 2.8]	[(3,3), (3.2, 3.2)]
	Level	1	1
RAM	**Clustering**	(2, 4)	(9, 16)
	Level	1	1
Hard Drive	**Clustering**	(820, 1024)	(1200, 1500)
	Level	2	3
Price	**Clustering**	[(420, 400), (600,	[(999, 799), (1200, 1150)]
	Level	3	2
Category	**Clustering**	[Desktop, …, Server]	[Apple, ..., MacBook-Air]
	Level	1	2

3.3. Computing Interest Weights of Offer Attributes

We give here an example on how to calculate the interest weights. Buyer1 selected the CPU clustering [2.2, 2.8] which is at level 1 of the 5-level CPU tree and all CPU data are in the range of [2.2, (3.2, 3.2)]**.** Thus, the interest-weight coefficient of CPU is 1.1167. We perform the same calculation for the other attributes: Price with a coefficient of 10.1521, Category with 0.9934, RAM with 2.3333 and Hard Drive with 3.6078. Buyer1's CPU interest weight for our 5-attribute application is as follows:

$$IW_{CPU\_Buyer1} = \frac{1.1167}{1.1167 + 10.1521 + 2.3333 + 3.6078 + 0.9934} \approx 0.0613$$

According to the interest-weight calculation performed by our system (cf. Figure 10), we can see that Price (with a weight of 0.5577) is the most important attribute for Buyer1**.** For the other attributes, Hard Drive (0.1982) is more important than RAM (0.1282) and CPU (0.0613). And CPU is less preferred than RAM. Such interests will help our system to understand Buyer1's needs. Buyer2 has another purchasing interest. If we compare the two buyers, we can see that Buyer2 is more interested in CPU and Hard Drive than Buyer1. This result corresponds to the buyers' selections: Buyer2 has selected a narrower clustering for these two attributes.



BuyerID	Attribute	SelectedCluster…	DataAttribute	SeletedLevelTree	TotalLevelTree	IW_coe	IW
1	CPU	0.60	3.35	1	5	1.1167	0.0613
1	RAM	2.00	14.00	1	3	2.3333	0.1282
1	Hard_Drive	204.00	1840.00	2	5	3.6078	0.1982
1	Price	205.91	3484.02	3	5	10.1521	0.5577
1	Catalog	0.38	1.51	1	4	0.9934	0.0546
2	CPU	0.28	3.35	1	5	2.3929	0.1966
2	RAM	7.00	14.00	1	3	0.6667	0.0548
2	Hard_Drive	300.00	1840.00	3	5	3.6800	0.3023
2	Price	404.48	3484.02	2	5	3.4454	0.2831
2	Catalog	0.38	1.51	2	4	1.9868	0.1632

Figure 10: Interest Weights for Both Buyers

3.4. Computing Interest Rates of Offer Attributes

For each buyer, we generate the interest rate functions for the five attributes by binding the selected clustering into the quickly changeable area. First we give an example on how to generate LVQs during the SOM learning time for high-dimensional attributes like CPU (cf. Table 6).

Table 6: LVQ Generation for CPU

Initial	LVQ1	(1.85, 0.5)		
	LVQ2	(2.98, 2.5)		
	LVQ3	(2.01, 1.54)		
Loop1	Input x	(2.2,0)	**…**	(3.2, 3.2)
	LVQ1	(1.8506, 0.4991)	**…**	(1.8565194, 0.4937947)
	LVQ2	(2.9799, 2.4999)	**…**	(2.9802438, 2.5018309)
	LVQ3	(2.0100, 1.5398)	**…**	(2.0097686, 1.53987435)
Loop2	LVQ1	(1.8565265, 0.49378457)	**…**	(1.8565933, 0.49372432)
	LVQ2	(2.9802438, 2.50183087)	**…**	(2.9802465, 2.50185127)
	LVQ3	(2.0097691, 1.53987249)	**…**	(2.0097661, 1.53987294)
Loop3	**…**	**…**	**…**	**…**
…				

To be able to produce the interest rate function for high-dimensional attributes, we need first to compute the distances between the attribute data and the best attribute value. For example in Table 7, we give the distances for CPU regarding Buyer1. Buyer1's selection of [2.2, 2.8] has a LVQ of (2.2654, 0.2979); the clustering [3.2, 3.2] is the best attribute value.

Table 7: Distances of High-Dimensional Data

x	**BestAttributeData**	**Distance(x, BestAttributeData)**
2.2 ***	(3.2, 3.2)	3.3526
2.8 **	(3.2, 3.2)	3.2249
(2.2654, 0.2979)*	(3.2, 3.2)	3.0489

***Min selection value (L), **Max selection value (R), *LVQ

In Table 8, we produce the interest rate function of CPU for Buyer1. This function is depicted in Figure 11. We may note that LVQ distance of 3.0489 is right outside of the selected clustering distance of [3.3526, 3.2249].

Table 8: $\varsigma_{CPU}()$ Generation for Buyer1

[AL, AR]	**Distance (LVQ, BestAttributeData)**	α_{Left}	α_{Right}	**Sign**
[2.2, (3.2,3.2)]	3.0489	6.5848	N/A	+1
[L, R]				
[2.2, 2.8]				

α_{Left} = 2 / | Distance(*L*, *BestAttributeData*) - Distance(*LVQ*, *BestAttributeData*) |
= 2 / | 3.3526 – 3.0489 | = 6.5848

$$\varsigma_{CPU}(x) = \varsigma_{CPU\_Left}(x) = \frac{1}{1+\exp^{[\alpha_{Left} \cdot Sign \cdot |Distance(x, BestAttributeData) - Distance(LVQ, BestAttributeData)|]}}$$

$$= \varsigma_{CPU\_Left}(x) = \frac{1}{1+\exp^{6.5848 \cdot |Distance(x,(3.2,3.2)) - 3.0489|}}, x \in [2.2, (3.2,3.2)\ ]$$

In Figure 11, most offers have lower CPU interest rates in the range of [0.11, 0.24]. Four offers have higher CPU interest rates: Offer15 (3.2, 3.2) with an interest rate of 0.9999; Offer 14 (1.8, 1.8) with 0.9991; Offer13 (3, 3) with 0.9999; Offer12 (1.9, 1.9) with 0.9997.

In Table 9, we produce the rate function for Buyer2. LVQ for the selected clustering [(3.0, 3.0), (3.2, 3.2)] is (2.8818, 2.9115). Thus the distance of LVQ to the best attribute data is 0.4295. LVQ is left outside of the selected clustering. Thus, the generated CPU interest rate function for Buyer2 has only $\varsigma_{Right()}$ (cf. Figure 12).

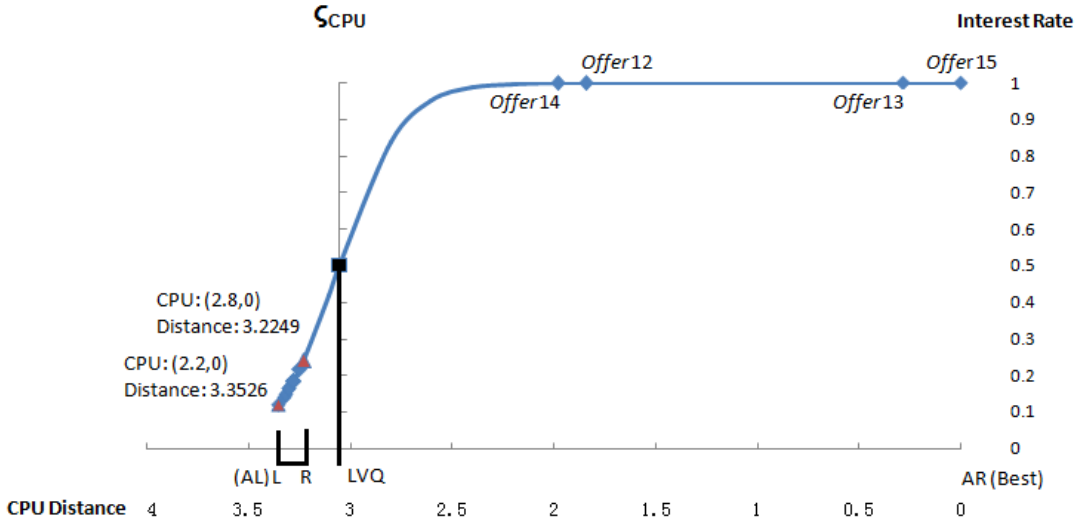


Figure 11: CPU Interest Rate Function for Buyer1

Table 9: $\varsigma_{CPU}()$ Generation for Buyer2

[AL, AR]	**Distance (LVQ, BestAttributeData)**	α_{Left}	α_{Right}	**Sign**
[2.2, (3.2,3.2)]	0.4295	N/A	-4.6564	+1
[L, R]				
[(3.0,3.0), (3.2, 3.2)]				

α_{Right} = -2 / | Distance(*R*, *BestAttributeData*) - Distance(*LVQ*, *BestAttributeData*) |
= -2 / | 0 – 0.4295 | = -4.6564

$$\varsigma_{CPU\_Buyer\,2}(x) = \varsigma_{CPU\_Right}(x) = \frac{1}{1+\exp^{[\alpha_{Right}\cdot Sign\cdot|Distance(x,BestAttributeData)-Distance(LVQ,BestAttributeData)|]}}$$

$$= \varsigma_{CPU\_Right}(x) = \frac{1}{1+\exp^{-4.6564\cdot|Distance(x,(3.2,3.2))-0.4295|}} \quad x \in [2.2,\ (3.2,3.2)]$$

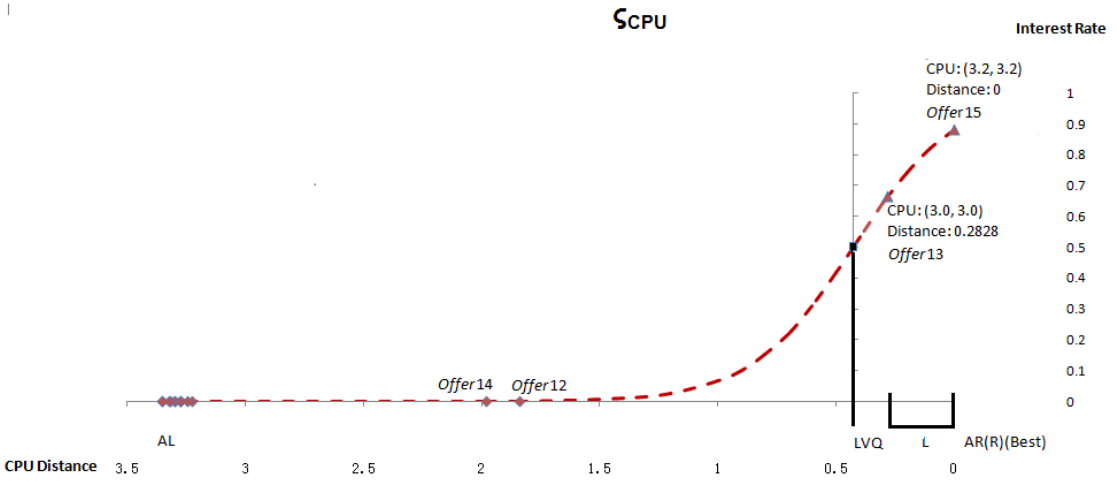


Figure 12: CPU Interest Rate Function for Buyer2

As we can see in Figure 12, only two offers have higher CPU interest rates: Offer15 (3.2, 3.2) with a rate of 0.8808 and Offer 13 (3, 3) with 0.6644. The other offers have lower CPU interest rates, less than 0.1, because they are far away from the selected clustering.

If we compare the two interest rate functions of Figure 11 and 12, we find out that most CPU attributes in offers are acceptable for Buyer1, while Buyer2 only interests the largest CPU. CPU interest rates of Offer12 and Offer14 for both buyers are obviously shown their different interest focus.

3.5. Evaluating the Offers with the Interest Model

After our system gets the interest weights and interest rate functions for the five attributes, it generates the interest model *(IM*) for both buyers as follows.

$$IM_{Buyer1}(Offer) = 0.0613 \cdot \varsigma_{CPU}(CPU) + 0.1282 \cdot \varsigma_{RAM}(RAM) + 0.1982 \cdot \varsigma_{HardDrive}(HardDrive) + 0.5577 \cdot \varsigma_{Price}(Price) + 0.0546 \cdot \varsigma_{Category}(Category)$$

$$IM_{Buyer2}(Offer) = 0.1966 \cdot \varsigma_{CPU}(CPU) + 0.0548 \cdot \varsigma_{RAM}(RAM) + 0.3023 \cdot \varsigma_{HardDrive}(HardDrive) + 0.2831 \cdot \varsigma_{Price}(Price) + 0.1632 \cdot \varsigma_{Category}(Category)$$

For instance, we show below how the interest model calculates the interest value for the first offer:

$$IM_{Buyer1}(Offer1) = 0.0613 \cdot \varsigma_{CPU}(2.2) + 0.1282 \cdot \varsigma_{RAM}(2) + 0.1982 \cdot \varsigma_{HardDrive}(320) + 0.5577 \cdot \varsigma_{Price}[(420,400)] + 0.0546 \cdot \varsigma_{Category}([Desktop, \ldots, Server])$$

$$= 0.0613 \cdot 0.119202 + 0.1282 \cdot 0.119207 + 0.1982 \cdot 0.0015 + 0.5577 \cdot 0.8800 + 0.0546 \cdot 0.8808$$
$$= \mathbf{0.5618}$$

$$IM_{Buyer2}(Offer1) = 0.1966 \cdot \varsigma_{CPU}(2.2) + 0.0548 \cdot \varsigma_{RAM}(2) + 0.3023 \cdot \varsigma_{HardDrive}(320) + 0.2831 \cdot \varsigma_{Price}[(420,400)] + 0.1632 \cdot \varsigma_{Category}([Apple, \ldots, MacBook\text{-}Air])$$

$$= 0.1966 \cdot 3.93E\text{-}07 + 0.0548 \cdot 1.15E\text{-}08 + 0.3026 \cdot 1.3E\text{-}09 + 0.2831 \cdot 0.9994 + 0.1632 \cdot 0.8702$$
$$= \mathbf{0.4250}$$

So, we obtained an interest value of 0.5618 for Buyer1 and 0.4250 for Buyer2. Based on these values, we can conclude that Buyer1 has a higher interest for Offer1 than Buyer2. With our interest model, the system evaluates all the candidate offers of Table 3. Our system returns a ranked list of offers for both buyers as shown in Tables 10: Offer1 is the best offer for Buyer1 and Offer15 the best offer for Buyer2.

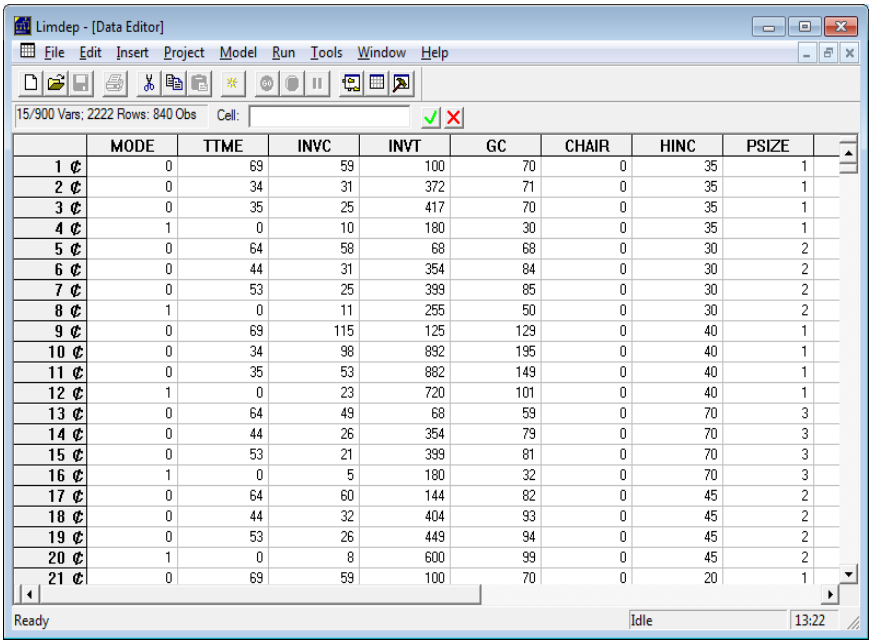
Table 10: Offer Evaluation and Ranking

Offer ID	IM_{Buyer1}
1	0.5618
2	0.5535
3	0.4736
15	0.4345
13	0.4314
4	0.3703
12	0.3667
14	0.3460
10	0.3356
7	0.2797
8	0.2537
9	0.2135
5	0.2069
6	0.2056
11	0.1946

Offer ID	IM_{Buyer2}
15	0.6553
13	0.5852
4	0.5292
12	0.4488
14	0.4302
2	0.4254
1	0.4250
3	0.4248
10	0.4219
8	0.4183
5	0.3924
6	0.3804
9	0.3672
7	0.1796
11	0.1775

4. Transport Usage Application

For the second experiment, we consider a database containing 840 user selections of four types of travel transport: by AIR, TRAIN, BUS and CAR [Greene 2010, Multinomial Choice 2010] (cf. Figure 13). In total, there are 210 users and each of them selected his best transport. The selected transport is represented by the attribute MODE with value 1, and the unselected one with value 0. For example, the first four rows are the choices of User1: the first three rows are respectively the unselected transport AIR, TRAIN and BUS; the fourth row is the selected transport CAR. For our experiment, we only focus on four attributes: TTME (Terminal TIME), INVC (IN Vehicle Cost), INVT (IN Vehicle Time) and GC (Generalized Cost of travel model) among the fourteen observed attributes given in [Multinomial Choice 2010].



	MODE	TTME	INVC	INVT	GC	CHAIR	HINC	PSIZE
1 ¢	0	69	59	100	70	0	35	1
2 ¢	0	34	31	372	71	0	35	1
3 ¢	0	35	25	417	70	0	35	1
4 ¢	1	0	10	180	30	0	35	1
5 ¢	0	64	58	68	68	0	30	2
6 ¢	0	44	31	354	84	0	30	2
7 ¢	0	53	25	399	85	0	30	2
8 ¢	1	0	11	255	50	0	30	2
9 ¢	0	69	115	125	129	0	40	1
10 ¢	0	34	98	892	195	0	40	1
11 ¢	0	35	53	882	149	0	40	1
12 ¢	1	0	23	720	101	0	40	1
13 ¢	0	64	49	68	59	0	70	3
14 ¢	0	44	26	354	79	0	70	3
15 ¢	0	53	21	399	81	0	70	3
16 ¢	1	0	5	180	32	0	70	3
17 ¢	0	64	60	144	82	0	45	2
18 ¢	0	44	32	404	93	0	45	2
19 ¢	0	53	26	449	94	0	45	2
20 ¢	1	0	8	600	99	0	45	2
21 ¢	0	69	59	100	70	0	20	1

Figure 13: 840 Observations on Transports Usage

We now apply our interest model to the database of Figure 13 in order to generate the best transport for each user. We consider each row as an offer. Thus, we have 840 offers. Our system clusters 840 data for each attribute, and produces four clustering trees for the attributes TTME, INVC, INVT and GC. Figure 14 depicts the clustering tree of TTME.

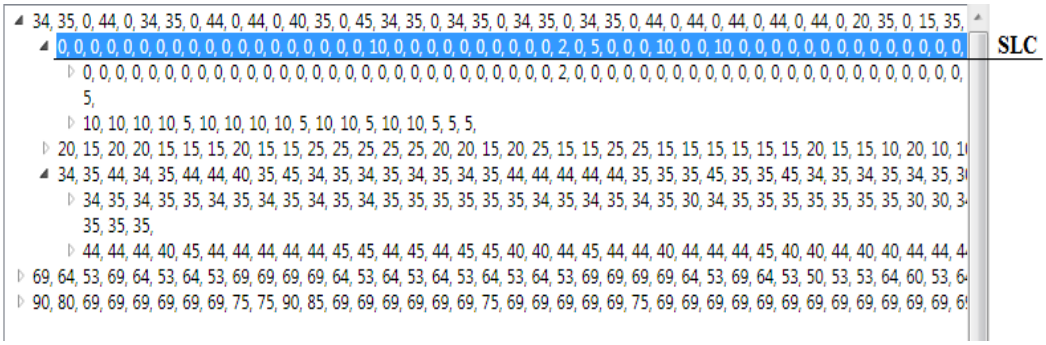


Figure 14: TTME Clustering Tree

Each user is required to select his most interested clustering for the four attributes in order to produce his interest model. For this experiment, we need in total 820 clustering selections. Since we cannot acquire all the users' selections, we assume here that each user chooses the largest clustering which contains the attribute values corresponding to the selected transport. We call this largest clustering SLC. In Figure 14, the highlighted clustering

represents SLC for User1 since it is the largest clustering containing the value 0 and not the values 69, 34 and 35 for the attribute TTME. The value 0 of TTME (fourth row of Figure 13) corresponds to User1's selected transport.

For our experiment, we only employ SLC as the users' selections for the four attributes. Table 11 shows the results returned by our evaluation system for 50 users. We need to manually select 200 SLCs for the 50 users. Our interest model accuracy rate is 40% when selecting the largest clustering. We may note that the accuracy is strongly dependent on the users' selections. Our interest model may have better evaluation results if we have the real users' selections. In addition, the smallest and deepest clustering in the tree will provide better results.

Table 11: Best Transports for 50 Users

Users	**User Selected Transport**	**IM Best Transport**
User1	Car	Car
User2	Car	Car
User3	Car	Car
User4	Car	Car
User5	Car	Car
User6	Train	Air
User7	Air	Train
User8	Car	Bus
User9	Car	Car
User10	Car	Car
User11	Car	Car
User12	Car	Car
User13	Car	Car
User14	Car	Car
User15	Car	Car
...		
User17	Train	Train
...		
User20	Train	Train
...		
User23	Air	Air
...		
User25	Air	Air
...		
User41	Air	Air
...		
User43	Air	Air
...		
User46	Air	Air
...		
User49	Air	Air
User50	Air	Bus

In Table 12, we measure the performance of our system for the applications of sections 3 and 4. All running time includes the interaction with the database system. We can see that our system runs with a very good performance. We cannot experimentally compare the efficiency between our system and other ranking methods since there are no standard and public datasets for semantic matchmaking and ranking.

Table 12: System Performance

	15 Offers (5 attributes)	**840 Offers (4 attributes)**
Clustering (Server)	580.2476 ms	110,284.265 ms
IW (Client)	13.4154 ms	51.6797 ms
IR (Client)	219.3452 ms	4,794.4967 ms
Evaluation (Client)	200.5748 ms	10,678.3938 ms
Total Time	1013.583 ms	12,5808.8 ms

5. Related Work

In the early time of e-commerce, matchmakers matched buyers with suppliers by mapping the attributes of the request and offers [Ha & Park 2001]. On the Internet, requests and offers are expressed in different schemas and words even when representing the same semantic meaning. This caused the early matchmakers to be blind to some potential offers. To solve this problem, semantic matching models, based on the ontologies, have been proposed. [Kawamura & Hasegawa 2005] matches services with the request by using a semantic filtering supported by the ontology. [Qiu & Li 2008] proposes an ontology similarity table to identify all the similar concepts of the request and offers**.** [Dong-wei et al. 2009] defines logical relationships in the ontology to map the concepts of request and offers**.**

Nevertheless, these matchmakers return an unranked list of candidate offers. So, the user needs to spend a lot of time evaluating the offers in order to find the best offer. Several ranking models have been introduced to sort the offers. [Dong-wei et al. 2006] introduces a distance measurement between the request concept node to the offer concept node in the ontology. [Di Noia et al. 2007a] presents a penalty function to convert the logical relationships in the ontology to digital values. The penalty values are then used to rank the offers. [Bener et al. 2009] assigns scores to the logical relationships. [Huang et al. 2008] utilizes a ranking function based on the user's attribute weights. [Shen et al. 2006] combines both concept ranking and constraint ranking. It analyzes each keyword and constraint appearing in the request and offer, and then calculates the offer fitness rate. In [Hahn et al. 2008], matching and ranking are based on the semantic similarity of the request model and offer model. We may note that most ranking criteria are defined by the system developers. Indeed, these ranking models dismiss the user's own ranking criteria. Users may have to accept a system believed best offer. In our research work, we take into consideration the user's interests to have a better chance to produce real and personalized results.

Furthermore, the above ranking methods map the functional properties of offers with the request's functional description. However, these methods cannot explain why sometimes the user prefers an offer that is different from the best recommended offer**.** [Yu & Reiff-Marganiec 2008] argues that the user's choice may be affected by other criteria often referred to as non-functional properties. Some methods have been proposed to handle the non-functional properties [Wang et al. 2006, Liu et al. 2009]. Since these non-functional properties functions are linear, they cannot make sure that each offer has a chance to be the best offer. In our work, we avoid this problem with our un-linear interest model.

In [Skoutas at al. 2010], the authors point out that any single matching criterion is not good enough to determine the best offer with multiple attributes. In addition, they are complex trades-off situations between attributes. The existing ranking models are only based on ranking the values of query words, functional or non-functional parameters, but not on ranking offers with individual's interests. The latter are the real reasons explaining why an individual selects a specific offer. Our research goal is to develop a more user-oriented and personalized system to evaluate the offers.

Most recommender systems utilize the collaborative filtering method to find the best item for the user [Zhan et al. 2010, Chandrashekhar & Bhasker 2011**,** Shambour & Lu 2011]. This method evaluates an item according to the other users' ratings. Other research work, like [Coleho at al. 2010], integrates the user's personality into the recommender system. Creating such user model requires a large set of user information, and sometimes the user does not leave any information. Recommender and matchmaking systems share the same goal: determine the best item for the user. However, there are major differences between these two systems: (1) matchmaking systems compute how much an offer is close to the user's request. They rank the offers based on the semantic matching degree between the query and offer [Di Noia et al. 2007a, Skoutas et al. 2010]. Matchmakers discover offers by mapping one by one the attributes of the offer and request; (2) recommender systems try to predict the user's rating for an item based on other users' ratings for this item. They do not examine each item attribute but consider an item as a whole.

Compared to our work, the accuracy of the results returned by the recommender systems is much lower since they recommend an item that is preferred by the other users with similar tastes. Our offer evaluation system determines the best offer by taking into consideration only the interests of the user. Furthermore, the recommender systems process a large database of user information. To evaluate the offers, our system requires very few inputs from the user. For each attribute, the user just needs to click on one clustering to make a selection.

6. Conclusion and Future Work

In this paper, we showed the benefits of sorting the request-matched offers according to the user's interests. By adopting the well-known economic MNL model, we produced an automated interest model that determines the best offer based on to the user's specific needs. The proposed system provides a more user-oriented and personalized solution for evaluating the offers and avoids the linear matching problems.

There are several possible directions of this work. The first one is to compare our interest model with the MNL model by using the transport usage application. The second direction is to include the interest learning [Wei et al. 2005] in our offer evaluation system. The main purpose of this learning is to update the interest model to fit the user's interests instantly. A learned interest model will be able to determine the best offer in these two following situations: the user shifts his interests, or new matched offers are added in our system database. Last but not least, we are interested in producing personalized attribute clustering trees specifically for each buyer because buyers may have different knowledge, experience, attitude, etc. Also, using users' feedbacks [Palanivel & Sivakumar 2010] may improve the system results**.** Our system will be able to better understand each buyer's interests with these features and feedbacks**.**

Acknowledgment

The authors wish to thank the NSERC Individual Discovery Grant of Canada for its support of this research.

REFERENCES

Bellur, U., and H. Vadodaria, "Web Service Ranking Using Semantic Profile Information," *International Conference on Web Services*, IEEE, pp 872-879, 2009.

Bener, A. B., V. Ozadali, and E. S. Ilhan, "Semantic matchmaker with precondition and effect matching using SWRL", *Expert Systems with Applications*, Vol. 36, No 5: 9371-9377, 2009.

Chabeb , Y., S. Tata, and A. Ozanne, "YASA-M: A Semantic Web Service Matchmaker," *International Conference on Advanced Information Networking and Applications*, IEEE, pp 966-973, 2010.

Chandrashekhar, H., and B. Bhasker, "Personalized Recommender System Using Entropy Based Collaborative Filtering Technique," *Journal of Electronic Commerce Research*, Vol. 12, No. 3, pp 214-237, 2011.

Chen, Y. Y., and K. Y. Young, "Applying SOM as a Search Mechanism for Dynamic System," *44th IEEE Conference on Decision and Control and European Control Conference*, pp 4111- 4116, 2005.

Cheng, T.W., W.L. Wang, and A. P. Chen, "E-marketplace using artificial immune system as matchmaker" , *International Conference on e-Commerce Technology*, IEEE, pp 358-361, 2004.

Coleho, B., C. Martins, and A. Almeida, "Web Intelligence in Tourism: User Modeling and Recommender System", *Web Intelligence and Intelligent Agent Technology*, (WI-IAT), IEEE/WIC/ACM, pp 619-622, 2010.

Di Noia, T., E. Di Sciascio, and F. M. Donini, "Semantic matchmaking as non-monotonic reasoning: a description logic approach", *Journal of Artificial Intelligence Research*, Vol. 29: 269-307, 2007a.

Di Noia, T., E. Di Sciascio, and F. M. Donini, "A nonmonotonic approach to semantic matchmaking and request refinement in E-marketplaces", *International Journal of Electronic Commerce*, Vol. 12, No 2: 127-154, 2007b**.**

Dong-wei, B., F. Ai-guo, and C. Shan-fa, "Semantic Matchmaking of Web Services Constraint Conditions", *5th International Conference on Wireless Communications, Networking and Mobile Computing*, IEEE, pp 1-5, 2009.

Dong-wei, B., L. Chuan-Chang, P. Yong, and C. Jun-liang, "Web Services Matchmaking with Incremental Semantic Precision", *International Conference on Wireless Communications, Networking and Mobile Computing*, IEEE, pp 1-4, 2006.

Essex, D., "Matchmaker, matchmaker", *Communications of the ACM*, Vol. 52, No. 5: 16-17, 2009.

Greene, W., *"*Discrete Choice Modeling*"*, http://pages.stern.nyu.edu/~wgreene/DiscreteChoice/Lectures/HEDGPart8-MultinomialLogit.ppt, 2011.

Guo, W., "Ontology Design for Supporting Matchmaking in E-commerce", *International Symposium on Information Science and Engineering*, pp 410-413, 2008.

Ha, S. H., and S. C. Park, "Matching users and suppliers: an intelligent dynamic exchange model", *Intelligent Systems*, IEEE, Vol. 16, No. 4: 28- 40, 2001.

Hahn, C., S. Nesbigall, S. Warwas, I. Zinnikus, M. Klusch, and K. Fischer, "Model-Driven Approach to the Integration of Multi-Agent Systems and Semantic Web Services", *12th Enterprise Distributed Object Computing Conference Workshops*, IEEE, pp 314-324, 2008.

Huang, R., Y. W. Zhuang, J. L. Zhou, and Q. Y. Cao, "Semantic Web-based Context-aware Service Selection in Task-computing", *International Workshop on Modelling, Simulation and Optimization*, pp 97-101, 2008.

Kawamura, T., and T. Hasegawa, A. Ohsuga, M. Paolucci, and K. Sycara, "Web services lookup: a matchmaker experiment," *IT Professional*, IEEE, Vol. 7, No. 2: 36-41, 2005.

Klusch, M., P. Kapahnke, and F. Kaufer, "Evaluation of WSML Service Retrieval with WSMO-MX", *International Conference on Web Services*, IEEE, pp 401-408, 2008.

Kohonen, T., "The Self-Organizing Map", 3rd Edition, Springer –Verlag, New York Inc., 2001.

Kreger, H., "Web Services Conceptual Architecture (WSCA 1.0)", IBM Software Group, 2001.

Liu, Y., N. A. H. H. Ngu, and L. Z. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection", *International Conference on Service Oriented Computing (ICSOC)*, Stockholm, Sweden, pp 23-27, 2009.

McFadden, D., "Conditional Logit Analysis of Qualitative Choice Behavior", *Frontiers in Econometrics*. Edition P. Zarembka, Academic Press: New York, pp 105-142, 1974.

McFadden, D., "Economic Choices, ``*American Economic Review''*, American Economic Association, Vol. 91, No. 3: 351-378, 2001.

Palanivel, K., and R. Sivakumar, "A Study on Implicit Feedback in Multicriteria E-Commerce Recommender System", *Journal of Electronic Commerce Research*, Vol. 11, No. 2, pp 140-156, 2010.

Park, A. Y., and U. Gretzel, "Influence of Consumers' Online Decision-Making Style On Comparison Shopping Proneness And Perceived Usefulness of Comparison Shopping Tools," *Journal of Electronic Commerce Research*, Vol. 11, No. 4, pp 342-354, 2010.

Qiu, T., and P. F. Li, "Web Service Discovery Based on Semantic Matchmaking with UDDI," *9th International Conference for Young Computer Scientists*, pp1229-1234, 2008.

Qu, C.T., F. Zimmermann, K. Kumpf, R. Kamuzinzi, V. Ledent, and R. Herzog, "Semantics-Enabled Service Discovery Framework in the SIMDAT Pharma Grid", *Transactions on Information Technology in Biomedicine*, IEEE , Vol. 12, No 2: 182-190, 2008**.**

Reiff-Marganiec, S., H.Q. Yu, and M. Tilly, "Service selction based on non-functional properties", *In Proceedings of Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, Springer, 2007.

Shambour, Q., and J. Lu, "Integrating Multi-Criteria Collaborative Filtering and Trust Filtering for Personalized Recommender Systems", *Computational Intelligence in Multicriteria Decision-Making (MDCM), IEEE Symposium*, pp 44-51, 2011.

Shen, X., X. Jin, R. F. Bie, and Y. C. Sun, "MSC: A Semantic Ranking for Hitting Results of Matchmaking of Services", *30th Annual International Computer Software and Applications Conference*, IEEE, pp 291-296, 2006.

Skoutas, D., D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and Clustering Web Services Using Multicriteria Dominance Relationships", *Services Computing*, IEEE, Vol.3, No 3: 163-177, 2010.

Wang, H., and Z. Z. Li, "A Semantic Matchmaking Method of Web Services Based on SHOIN^+ (D)*," *Asia-Pacific Services Computing Conference*, IEEE, pp 26-33, 2006

Wang, X., T. Vitvar, M. Kerrigan, and I. Toma, "A QoS-aware selection model for semantic web services," *4th International Conference on Service-Oriented Computing*, pp 390–401, 2006.

Wei, Y. Z., L. Moreau, and N. R. Jennings, "Learning users' interests by quality classification in market-based recommender systems", *Transactions on Knowledge and Data Engineering*, IEEE, Vol. 17, No. 12: 1678-1688, 2005.

Xu, C., W. Wang, Z. B. Li, and C. Yang, "Comparative study on drivers' route choice response to travel information at different departure time", *2nd International Asia Conference on Informatics in Control, Automation and Robotics*, IEEE, No. 3: 97-100, 2010.

Yu, H.Q., and S. Reiff-Marganiec, "Non-Functional Property based Service Selection: A Survey and Classification of Approaches", *Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, ECOWS, IEEE, 2008.

Yu, S. H., Y. J. Li, and X. B. Yan, "A Spatial Neural Network Application in Consumer Spatial Behavior Modeling", *International Conference on Machine Learning and Cybernetics*, IEEE, pp 3044-3047, 2006.

Zhan, J., C. L. Hsieh, I. C. Wang, T. S. Hsu, C. J. Liau, and D. W. Wang, "Privacy-Preserving Collaborative Recommender Systems", *Systems, Man, and Cybernetics, Part C: Applications and Reviews*, IEEE, Vol. 40, No 4: 472-476, 2010.

*"*Multinomial Choice Travel Mode Data Set*"*, http://pages.stern.nyu.edu/~wgreene/DiscreteChoice/Data/clogit.lpj, 2011.

Appendix A: SOM ( ) Function

```
/* Related class member variables */
private double[] A1_LVQ;            // LVQ1
private double[] A2_LVQ;            // LVQ2
private double[] A3_LVQ;            // LVQ2
private ArrayList A1_offer_id = new ArrayList();      // A1
private ArrayList A2_offer_id = new ArrayList();      // A2
private ArrayList A3_offer_id = new ArrayList();      // A3

private void SOM ( DataTable attribute_table, int dimension, DataTable root_attribute_table, int level )
{  /* Initialize Data Values */
        double LVQ1_distance =0;
        double LVQ2_distance =0;
        double LVQ3_distance =0;
        int learn_time =1;
        this.CreateLVQ (dimension, attribute_table); /* Create Random LVQ */
        /* Learning Loop */
        while (this.GetLearningRate (learn_time) > 0.0001)
         //Check if it need learning again
        {  // Pick up value x
                for ( int i=0; i<attribute_table.Rows.Count; i++ ) {
                for ( int j=1; j<=dimension; j++ ) {
                    // Calculating distance between x to LVQ1 through the loop
                    LVQ1_distance += Math.Pow (
                            (System.Convert.ToDouble (attribute_table.Rows[i][j])
                            - this.A1_LVQ[j-1] ), 2.0 );
          // Calculating distance between x to LVQ2 through the loop
          LVQ2_distance += Math.Pow (
                  (System.Convert.ToDouble (attribute_table.Rows[i][j])
                            -this.A2_LVQ[j-1] ), 2.0 );
          // Calculating distance between x to LVQ3 through the loop
                    LVQ3_distance += Math.Pow (
                  (System.Convert.ToDouble (attribute_table.Rows[i][j])
                            - this.A3_LVQ[j-1] ), 2.0 );
                    }
                    // Get each LVQ distance to x
                    LVQ1_distance = Math.Round ( Math.Sqrt (LVQ1_distance), 2 );
                    LVQ2_distance = Math.Round ( Math.Sqrt (LVQ2_distance), 2 );
                    LVQ3_distance = Math.Round ( Math.Sqrt (LVQ3_distance), 2 );

                // Find the closest LVQ to x
                this.FindWinnerLVQ ( LVQ1_distance, LVQ2_distance,
                        LVQ3_distance, System.Convert.ToInt32 (attribute_table.Rows[i][0]) );
                    this.updateLVQ ( i, dimension, attribute_table,
                            LVQ1_distance, LVQ2_distance, LVQ3_distance,
                            this.GetLearningRate (learn_time), root_attribute_table );

                // Update LVQi
                LVQ1_distance = 0;
                LVQ2_distance = 0;
                LVQ3_distance = 0;  // Clear LVQ distance for next input x
                }
                learn_time ++;        // Learning time update
                if ( this.GetLearningRate (learn_time) > 0.01 )
                {   // Prepare for next learning
```

```
            A1_offer_id = new ArrayList();
                    A2_offer_id = new ArrayList();
            A3_offer_id = new ArrayList();
                    }
    }
}

/*  SOM sub-function
       * Find the closest LVQc to x
       * Put x into LVQc related cluster  */

private void FindWinnerLVQ ( double distance1, double distance2, double distance3, int current_row ) {
            int winner = 0;
            /* Find the closest LVQ based on distance */
            if (distance1 <= distance2){
       if (distance1 <= distance3) { winner = 1; }
            else { winner = 3; }   }
            else{
       if (distance2 <= distance3) { winner = 2; }
            else { winner = 3; }   }
            /* Put x in Ac
            * The Ac could be temporary.
            * It depends on when the learning loop stop.
            * The last time generated Ac is the final result */
            if (winner == 1) { this.A1_offer_id.Add (current_row); }
            if (winner == 2) { this.A2_offer_id.Add (current_row); }
            if (winner == 3) { this.A3_offer_id.Add (current_row); }
}
```

Appendix B: Clustering Category (position view)