Intelligent Product Brokering for E-Commerce: An Incremental Approach to Unaccounted Attribute Detection

Sheng-Uei Guan, Ping Cheng Tan and Tai Kheng Chan
Department of Electrical & Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260

ABSTRACT

This research concentrates on designing generic product-brokering agent to understand user preference towards a product category and recommends a list of products to the user according to the preference captured by the agent. The proposed solution is able to detect both quantifiable and non-quantifiable attributes through a user feedback system. Unlike previous approaches, this research allows the detection of unaccounted attributes that are not within the ontology of the system. No tedious change of the algorithm, database, or ontology is required when a new product attribute is introduced. This approach only requires the attribute to be within the description field of the product. The system analyzes the general product descriptions field and creates a list of candidate attributes affecting the user's preference. A genetic algorithm verifies these candidate attributes and excess attributes are identified and filtered off. A prototype has been created and our results show positive results in the detection of unaccounted attributes affecting a user.

Keywords: Unaccounted Attributes, Genetic Algorithm, Product-Brokering Agent

1 Introduction

E-Commerce activities have been growing at a tremendous rate of 100% per annum in the past few years [1]. The tremendous increase in E-Commerce activities gives rise to the need of creation of new software technology in many areas. A major area of concern is to develop a good business to consumer environment on the various online stores found. This can be done through the creation of intelligent software agents to fulfill the needs of consumers patronizing online E-Commerce stores. This includes intelligent filtering services and product brokering services to understand user's needs before alerting users of suitable products according to their needs and preference.

1.1 Current Situation and Motivation for Research

Although there is a tremendous increase in e-commerce activities, technology in enhancing consumers' shopping experience remains primitive. Online stores and websites today often only provide filtering and search services to the users. Unlike real life department stores, there are no sales assistants to aid consumers in selecting the most appropriate product for users. Consumers are further confused by the large options and varieties of goods available. Thus there is a need to provide on top of the provided filtering and search services, an effective piece of software in the form of a product brokering agent to understand their needs and assist them in selecting suitable products.

1.2 Definitions of Product Attributes

A user's choice in selecting a preferred product is often influenced by the product attributes that range from price to brand name. This research shall classify attributes as accounted, unaccounted, and detected. The same attributes may also be classified as quantifiable or non-quantifiable attributes.

Accounted attributes are predefined attributes that the system is specially catered to handle. A system may be designed to capture the user's choice in terms of price and brand name, making them accounted attributes. *Unaccounted attributes* have the vice versa definition and such attributes are not predefined in the ontology of the system. The system does not understand whether an unaccounted attribute represents a model or a brand name. Such attributes merely appear in the product descriptions field of the database. The system will attempt to detect the unaccounted attributes that affect the user's preference and consider them as *detected attributes*. Thus detected attributes are unaccounted attributes that are detected to be crucial in affecting the user's preference.

Quantifiable attributes contain specific numeric values (e.g. hard disk size) and thus their values are well defined. Non-quantifiable attributes on the other hand do not have any logical numeric values and their valuation may differ from user to user (e.g. brand name).

The proposed system shall define Price and Quality of a product in the ontology and consider it to be quantifiable, accounted attributes. All other attributes defined in the system and considered as unaccounted attributes that will be detected by the system.

1.3 Related Work on Handling Product Attributes in E-Commerce

A lot of research and work has been done to aid transactions in electronic commerce. One of the research aims found is to understand a user's needs before recommending products through the use of product brokering services. Due to the difference in complexity, different approaches are proposed to handle quantifiable and non-quantifiable attributes.

One of the main approaches to handling quantifiable attributes is to compile these attributes and assign weights representing their relative importance to the user. The

weights are adjusted to reflect the user's preference. One of the adopters of such an approach, "An Intelligent Product Brokering Agent for M-Commerce" [3] captures user preference by requesting the user to select the best product from a short list of products before adjusting the weights according to this feedback.

A lot of research aimed at creating an interface to understand user preference in terms of non-quantifiable attributes. This represents a more complex problem than its quantifiable cousins, as attributes are highly subjective with no discrete quantity to measure their values. Different users will give different values to a particular attribute. "MARI" (Multi- Attribute Resource Intermediary) by Software Agent Group in the MIT Media Labs [2] proposed a "word of mouth approach" to solve this problem. The project split up users into general groups and estimated their preference to a specific set of attributes through the group the user belongs to.

Another approach in the handling of non-quantifiable attributes involves specifically requesting the user for the preferred attributes. One project, "Intelligent Profile By Example" by MIT Labs [5] provides a learning tool for the user to explore his preference before requesting him to suggest desirable attributes.

Some of the main problems in related work lie in the handling of non-quantifiable attributes, as the approaches are too general. Most work so far only attempts to understand user preference through generalization and stereotyping instead of understanding specific user needs. Another main problem is that most works are only able to handle a specific set of attributes. The attributes that they are able to handle are hard-coded into the design of the system and the consequence is that they are not able to handle attributes that are unaccounted and beyond the pre-defined list. However the list of product attributes is

often large, possibly infinite. The approach used in related research may not be able to cover all the attributes, as they need to classify them into the ontology.

1.4 Objectives and Research Contribution

The main objective is to introduce an approach to capture user preference to a product category in terms of quantifiable and non-quantifiable attributes and produce a list of highly desired products for a user. After considering the flaws in current work and research, we propose an approach to capture user preference.

Previous approaches concentrate on defining a specific solution to capture the reaction of particular users towards specific pre-defined quantifiable attributes (e.g. Price). As demonstrated in the project "MARI", non-quantifiable attributes are often neglected or estimated by the general group each user belongs to. There is no capturing of the specific user's response towards a non-quantifiable attribute. The solutions proposed are also specific to individual attributes or restricted by the underlying ontology defined. Addition of new product attributes is extremely tedious, often requiring re-designing of the system.

This approach unlike previous ones presents a generic approach to capture individual user responding towards product attributes including non-quantifiable ones. The proposed solution does not generalize or stereotype user preference but capture the user's unique taste and recommend a list of products to the user. Under the proposed generic approach, the system is able to handle the inclusion of any unaccounted attribute that is not predefined in the system, without re-programming the system. The system is able to cater for any unaccounted attribute through a general description field found in most product database. This is extremely useful as hundreds of new attributes of products

emerge each day making any complex analysis impossible. In addition, the system is self-adjusting in nature and can adapt to changes in user's preference.

1.5 Proposed Approach

The proposed approach attempts to capture user preference on the basis of two quantifiable accounted attributes, Price and Quality. These attributes affect the preference of almost all users and thus are constantly checked in the system. The proposed approach incrementally learns and detects any unaccounted attribute that affects the user's preference. If any unaccounted attribute is suspected, the system attempts to come up with a list of highly controversial attributes and verify their importance through a genetic algorithm. Thus vital attributes that are unaccounted for previously will be considered. The unaccounted attributes are derived from the general description field of a product. Hence unlike most related work, the approach is generic in nature, as the system is not restricted by the attributes it is designed to cater to.

2 GENERAL DESIGN ISSUES

A product-brokering agent often has the ability to capture user preference. In the process of capturing consumer taste in our application, we shall attempt to inspect user preference towards two quantifiable attributes - price and quality. The system shall attempt to detect other attributes that affect user preference. As these attributes to be detected are not pre-defined or accounted by the system, they are considered as unaccounted attributes. The system shall attempt to detect the presence of these unaccounted attributes and account for them in the analysis of the particular user's preference. After the unaccounted attributes are detected, they become detected attributes.

2.1 Overall Design Architecture

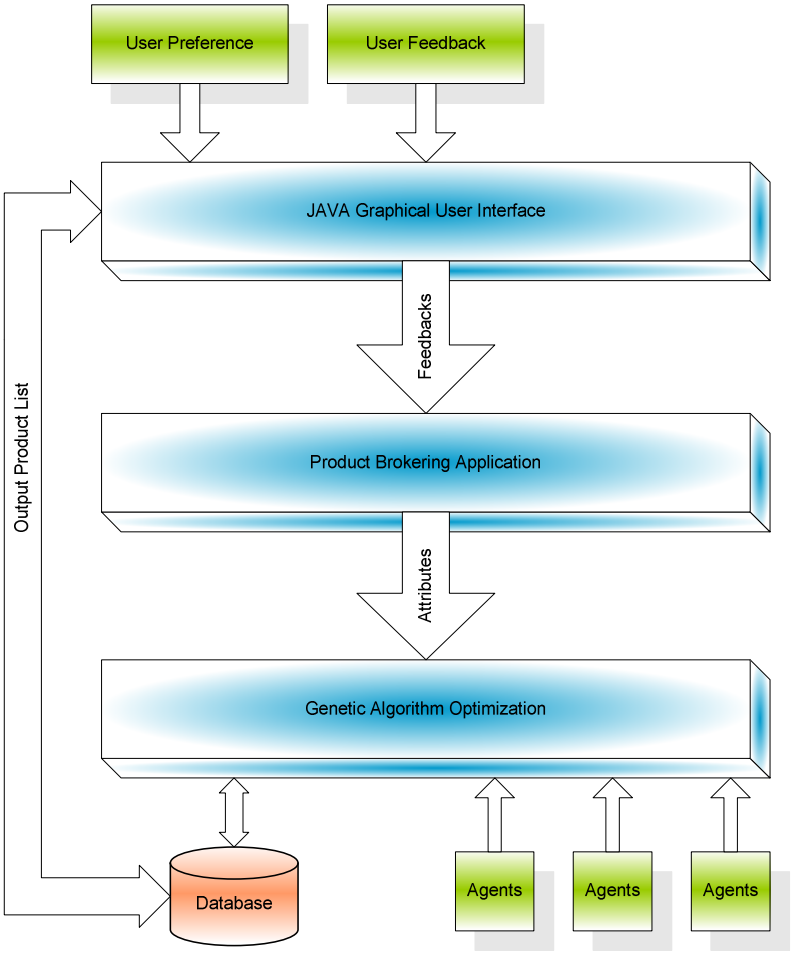

Fig. 1a: System Architecture

The overall architecture is as shown in Fig.1a. User preference and feedback are gathered from a GUI based Java program at the UI layer. Following that, the product brokering application layer will process the feedback and preference in search of attributes. The genetic algorithm optimization layer will interface between the database and the intelligent software agents. With genetic algorithm, the agents will process the attributes and store relevant information in the database which allows the user to view from the GUI. All agents will evolve according to their fitness and some will survive and some will die.

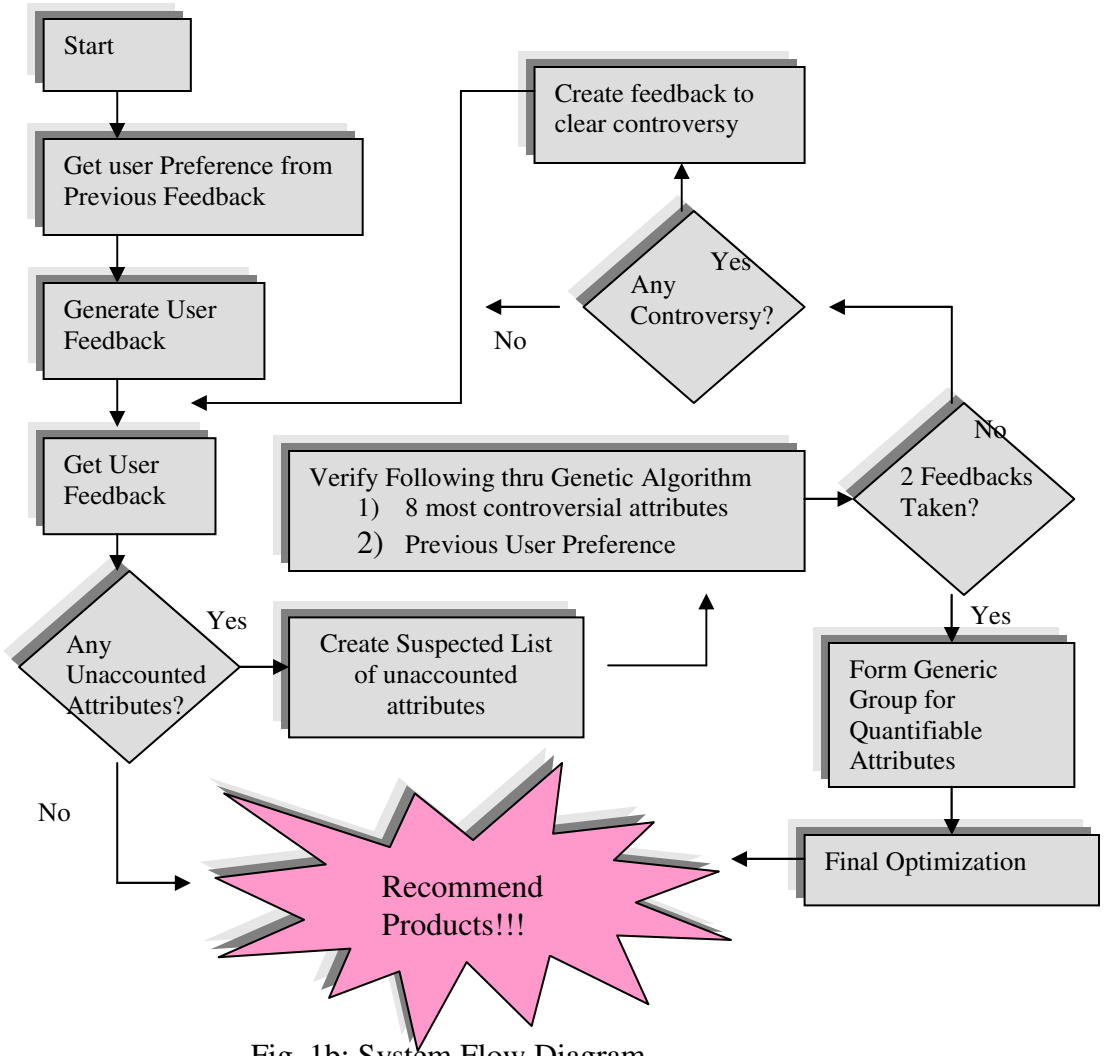

Fig. 1b: System Flow Diagram

Figure 1b shows the System Flow Diagram which illustrates how the system works. As the system is able to incrementally detect the attributes that affect user preference, the system firstly retrieves any information captured regarding the user from some previous feedback. The system generates a feedback in the form of a list of products for the user to rank. Next, the system attempts to investigate the presence of any unaccounted attribute affecting the user's preference. The system shall compile a list of possible attributes that are unaccounted for by analyzing the user feedback and rank them according to their confidence levels. The most controversial attributes and any information captured from

previous feedback are then verified through a genetic algorithm. Redundant attributes that do not affect the user's preference are dropped. After which, the system attempts to clear any controversy arising regarding the attributes captured by a new cycle of feedback. If two cycles of feedback are completed, the system attempts to detect any quantifiable attributes that are able to form a generic group of attributes (e.g. speed represented in MHz). The system finally optimizes all information accumulated by a genetic algorithm and recommends a list of products for the user according to the preference captured.

2.2 Multiple-Agent System

The system is generally made up of a family of 60 product-brokering agents with the fittest agents recommending suitable products for the user. Before any recommendation can be made, each agent needs to grade the products according to a common score system. The score given to each product is based on a Tangible Score based on some attributes accounted by the system and Modification Score based on detected attributes. Detected attributes are unaccounted attributes outside the ontology of the system.

2.3 Tangible Score

In our application, we shall consider two quantifiable attributes, Price and Quality as the basis in deriving the Tangible Score. The effect of these two attributes is always accounted for. The equation to derive this score is as shown in equation 1-3.

$$Score_{PriceCompetitive} = PrefWeight * (MaxPrice - Price) / MaxPrice \quad (1)$$

$$Score_{Quality} = (1.0\text{-}PrefWeight) * Quality \quad (2)$$

Equation 1 measures the price competitiveness of the product. PrefWeight is the weight or importance the user places on price competitiveness as compared to quality with values ranging from 0 to 1.0. A value of 1.0 confirms that 100% of the user's preference is based

on price competitiveness. A product with a price close to the most expensive product (MaxPrice) will have a low score in terms of price competitiveness and vice versa.

Equation 2 measures the score given to quality. The quality attribute measures the quality of the product and takes a value ranging from 0.0 to 1.0. The value of "1.0 – PrefWeight" measures the importance of Quality to the user.

The final score given to tangible attributes are computed by adding equation 1 and 2 as shown in equation 3 below.

$$\text{TangibleScore} = \text{Score}_{\text{PriceCompetitive}} + \text{Score}_{\text{Quality}} \quad (3)$$

2.4 Modification Score for Detected Attributes

The modification score is the score assigned to all detected attributes by the system. These detected attributes are previously unaccounted attributes but had been detected by the system to be a vital attribute in the user's preference. These include all other attributes besides price and quality. As these attributes may not have a quantifiable value, the score is taken as a factor of the TangibleScore derived earlier. The Modification Score is as shown in equation 4 whereby the modification factor K is introduced.

$$\text{Modification Score} = \sum_{i=1}^{NoOfAttributes} (K_i - 1) * TangibleScore \quad (4)$$

The values of each modification factor K; ranges between 0.0 and 2.0. A value of K shall be assigned for each newly detected attribute (e.g. each brand name will have a distinct value of K). The modification factor K takes a default value of 1.0 that gives a Modification Score of 0. Such a situation arise when the detected attribute do not affect the user's choice. When K<1.0, we will have a negative or penalty score for the particular attribute. This will take place when the user dislikes products from a certain brand name or other detected attributes. When K >1.0 we will have bonus score to the attributes and it

takes place when the user has special positive preference towards certain attributes. By using a summation sign as shown in equation 4, we are considering the combined effects of all the attributes that are previously unaccounted by the system.

The difference in design when deriving the Modification Score and Tangible Score lies in the uncertainty nature of the unaccounted attributes detected. The design in modification score allows detected attributes to be dropped without affecting other attributes if they are found to be unimportant.

With all new attributes captured, the final score for the product is as shown below in equation (5) as the summation of Tangible and Modification scores.

Final Score = TangibleScore + Modification Score (5)

2.5 A Ranking System for User Feedback

As shown in equation 1, 2 earlier, there is a need to capture user's preference in terms of the PrefWeight in equation 1 and the various Modification Factor K in equation 4. The system shall request the user to rank a list of products as shown in Fig. 2. The user is able

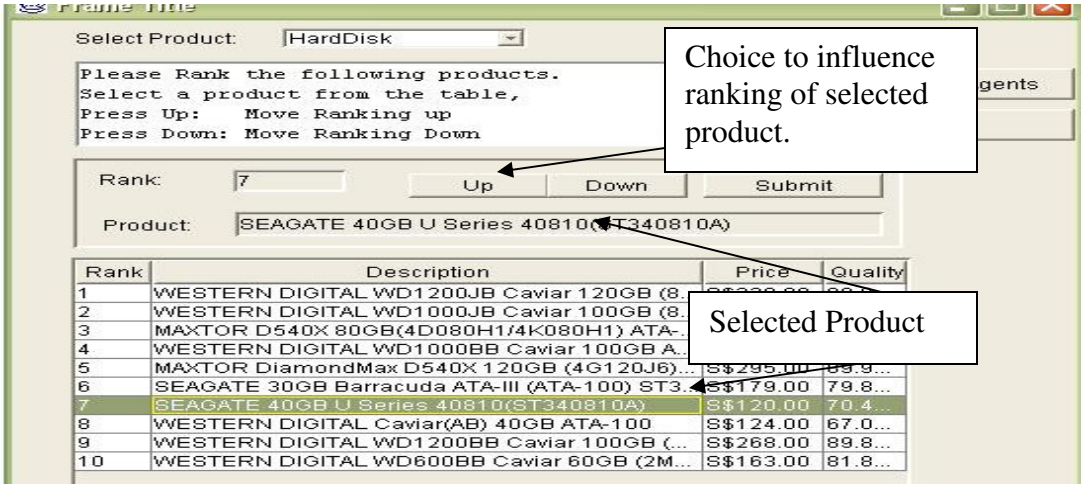

Fig. 2: Requesting the User to Rank a List of Products

to rank the products according to his preference with the up and down button and submit when done. The system shall make use of this ranked list to assess a best value for

PrefWeight in equation 1 and 2. In a case whereby no unaccounted attributes affect the user's feedback, the agents will be evolved along the PrefWeight gradient to optimize a value for the PrefWeight.

2.6 Fitness of Agents

The fitness of each agent shall depend on the similarity between the agent's ranking of the product and the ranking made by the user. It reflects the fitness of agents in capturing the user's preference. Consider a case whereby the user is required to rank a list of 10 products as shown in Fig. 3.

If a product is ranked 1st, it should have a higher score than the No. 2 to No. 10 products. Thus 1 fitness point is awarded to an agent if the agent grades the No. 1 product to have a higher score than the No. 2 product. Similarly, 1 fitness point is awarded to the agent if the No. 1 product has a higher product score than the No. 3 product. Extending to all cases of comparison involving the No. 1 product, a maximum of 9 fitness points can be awarded to an agent. Similarly a maximum of 8 fitness points can be awarded to an agent with regard to the No. 2 product, as it should have a higher score than the other 8 products under it. Considering all permutations, a total of 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45 fitness points can be awarded to an agent.

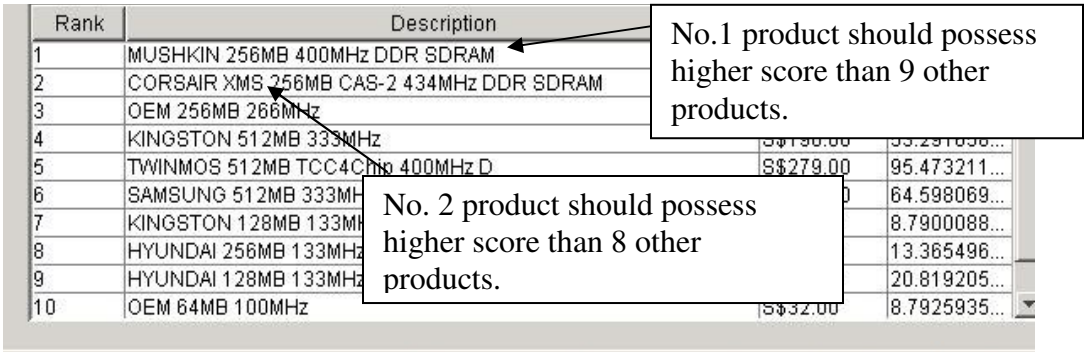

Fig. 3: Relationships between Product Rankings

3 DETECTION OF UNACCOUNTED ATTRIBUTES

This research unlike the related work is able to detect and handle attributes beyond the attributes classified under the ontology of the system. To demonstrate this ability, the system's ontology shall contain only Price and Quality while all other attributes are unaccounted and remain to be detected, if they are vital to the user. These unaccounted attributes include non-quantifiable attributes that are subjective in nature (e.g. brand name). The unaccounted attributes can be retrieved by analyzing the description field of a product database thus allowing new attributes to be included without the need of change in ontology or system design. Each token or word in the description field is considered as an independent possible attribute and the system serves to detect and confirm their presence.

The system goes through 2 stages to handle unaccounted attributes. The system firstly goes through a detection stage where it comes up with a list of attributes that affect the user's preference. These attributes are considered as unaccounted attributes as the system has not accounted for them during this stage. A *"Confidence Score"* is assigned to each attribute according to the possibility of it being the governing attribute influencing the user's preference. From this list, the system makes a set of hypothesis that 8 attributes with the highest Confidence Score affect the user's preference. In the 2nd stage, the system attempts to verify the hypothesis that each of these attributes is indeed a governing attribute. Attributes affecting the user's preference in the previous set of feedback are also assumed to be a vital attribute and will be verified. The attributes that are not vital in determining the user's preference (i.e. hypothesis fails) will be deleted, leaving a pool of vital product attributes affecting the user's preference. These previously unaccounted

attributes are thus detected by the system to be vital and their status of being "unaccounted" shall become "detected".

3.1 Detection of Products Containing Unaccounted Attributes

The presence of unaccounted attributes is detected when an illogical ranking occurs. Each agent shall consider Price, Quality and a list of detected attributes (previously unaccounted) to be governing the user's preference. The system shall request the user to rank a list of products and analyze the feedback according to the process as shown in Fig. 4. The agents shall attempt to explain the rankings by optimizing the PrefWeight value and various K values. The fittest agent shall give each product a score.

The system shall loop through the 10 products that are ranked by the user and compare the score given to products. If the user ranks a product higher than another, this product should have a higher score than a lower ranked product. However if the agent awards a higher score to a product ranked lower than another, (e.g. product ranked 2nd has higher score than 3rd), the product is deemed to contain an unaccounted attribute causing

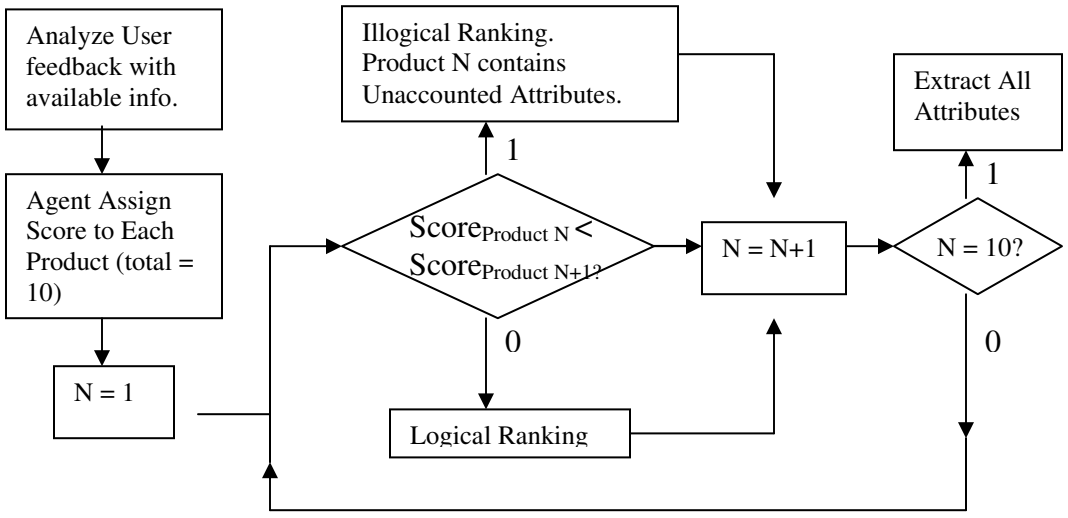

Fig. 4: Process Identifying Products with Illogical Rankings

an illogical ranking. This process shall be able to identify all products containing positive unaccounted attributes that the user has preference for.

The logic behind the process is demonstrated by the following example. The user ranks product A higher than product B. However, the agent awards product B a higher score creating an illogical ranking. Either A or B may be the product with illogical ranking. The system 1st assumes that A contains some unaccounted positive attribute causing the illogical ranking and ranking of B is perfectly logical.

However this assumption that product A contains an illogical ranking may be wrong. If indeed product B and not A contains an illogical ranking, it will cause all other products to be classified as having an illogical ranking. This is because the ranking of other products will be illogical compared to product B (e.g. the user ranks product B lower than C or D. However the agent had given B a higher score than C and D.). On this basis, in the case whereby more than half of the products are deemed to cause illogical rankings, the status of these products is reversed. Products that are deemed logical in ranking shall be considered illogical and vice versa. The products in such case consist of some negative unaccounted attributes that the user dislikes.

3.2 Detection of Unaccounted Attributes

As shown in section 3.1, we can detect the presence of unaccounted attributes when a few products show illogical rankings. The next step is to identify the unaccounted attributes inside these products that give rise to such illogical rankings. As shown in Fig. 5 the products with illogical rankings are tokenized. Each word in the product descriptions field is considered as a possible unaccounted attribute affecting the user's preference. Attributes that have already been accounted for shall not be considered

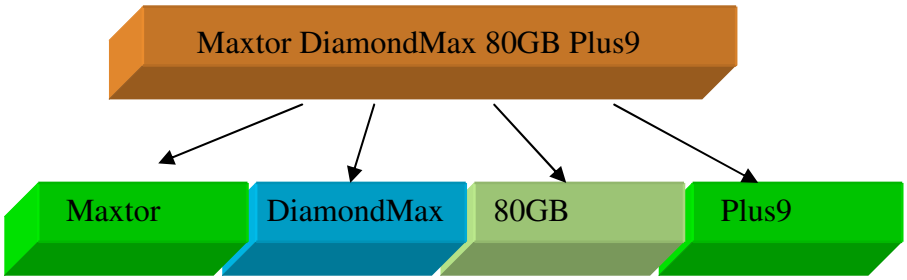

Fig. 5: Getting Intangible Attributes from Product Descriptions

Each of the tokens is considered as a possible attribute affecting the user's taste. If a particular unaccounted attribute contains several tokens and is common (e.g. famous brand name), it will be more efficient to consider the full attribute as a whole rather than several tokens (e.g. Consider Creative Technologies as a single token). Thus the system can contain a small database of common attributes (e.g. database of brand names) and every token shall be checked for the possibility of it belonging to an attribute consisting of several tokens. The system can also compile a list of common attributes and give it a higher confidence score. Each token shall initially be awarded the same Confidence Score. The system shall next analyze the situation and modify the Confidence Score according to the cases as shown.

1) The token appears in other products and shows no illogical ranking

We can conclude that there is a low possibility of the unaccounted attribute being a crucial influencing attribute that causes the illogical rankings. This is because their presence in other products does not cause an illogical ranking situation. For every case in which other products contain the token we shall deduct 15 points from the Confidence Score.

2) The token appears in other products and shows illogical ranking

We can conclude that the token in question here has a higher chance of containing an unaccounted attribute. This is because other products containing these tokens also have an illogical ranking. For every case that the product contains the token, we shall add 10 points to the Confidence Score.

The design above only provides an estimate on the Confidence Points according the 2 cases described and may not be 100% reliable. It is discovered that the 1st case (penalty approach) is slightly more reliable than the 2nd and thus the larger magnitude in affecting Confidence Points (15 penalty points compared to the 10 bonus points).

3.3 Confirmation of Attributes

The system has come up with a list of unaccounted attributes through the Confidence Score. It was found that most of the attributes affecting the user's preference are within 6 products containing the highest Confidence Score. Attributes captured in previous feedbacks may be relevant in the current feedback as the user may choose to provide more than 1 set of feedback. The system thus makes a hypothesis that the user's preference is influenced by attributes affecting him in previous feedbacks if available and 8 other new attributes with the highest Confidence Score. The effect of these attributes on the user's preference is verified next.

Each agent in the system shall make estimation on the user's preference by randomly assigning a modification factor (or "K" value) for each of the 8 attributes with high Confidence Score. Attributes identified to be positive (described in Sec. 3.2) are given K values greater than 1.0 while negative attributes have K values less than 1.0. The K values and PrefWeight are optimized by a genetic algorithm (as discussed later) to improve the fitness level of the agents.

The agents with the highest fitness level are the best agents that understand the user's preference based on the rankings made in the feedback. Thus the "K" values of these agents form good estimations to describe the user's preference. From the "K" values we can verify each of the hypotheses made earlier. Attributes that do not affect the user's decision will have a "K" value close to 1.0 and the hypothesis fails for such attributes. Positive attributes having "K" values greater than 1.0, and negative attributes with K smaller than 1.0, are vital attributes and thus the hypothesis that such attributes affect the user's preference is verified. The agents are put through a filtering process to filter off the unimportant attributes. Attributes with "K" values between 0.9 and 1.1 are removed as their effect is mild, affecting the Tangible Score by less than 10%.

The remaining attributes undergo another filtering process as shown in Fig. 6 whereby redundant attributes that do not affect the agent's fitness are filtered off. The

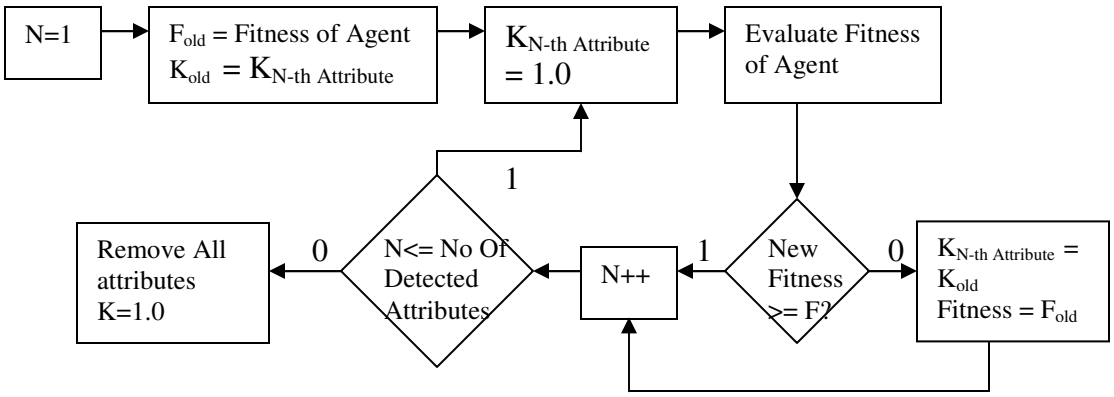

Fig. 6: Removing of Redundant Attributes

system loops through all the attributes detected and substitute K = 1.0 for all these attributes. If the fitness of the agent drops as a result of this substitution, the previous K value is restored and the process iterates for all detected vital attributes affecting the user's preference.

3.4 Optimization Using Genetic Algorithm

The status of detected attributes perceived by the agents and the 8 most controversial attributes should be verified here. The attributes unimportant to the users shall be deleted here, as they assume a modification value (K) of 1.0. The PrefWeight and various K values shall be optimized here to produce maximum agent fitness. As this represents a multi-dimensional problem with each new K value introducing a new dimension, we shall optimize using a Genetic Algorithm that converts the attributes into binary strings. The agents shall evolve under the genetic algorithm to optimize the fitness of each agent. The PrefWeight and various K values are optimized in this algorithm. The PrefWeight and K values may not be accurate in some unique cases when multiple feedbacks from the user are obtained as the user preference may change. To aid the system adapt to any changes in user preference in terms of PrefWeight, four PrefWeight values 0.2,0.4,0.6,0.8 shall be conserved. This will allow quick adaptation especially if the user shows a huge change in PrefWeight (e.g. from 0.2 to 0.55) preference as the agents will have higher chances of adopting a better estimate (agents that adopt PrefWeight = 0.6 in mating process has better fitness).

As mentioned earlier, 8 attributes with the highest Confidence Score are assumed to affect the user's preference. However, the user may be affected by only 1 of the 8 attributes (thus the 7 unimportant attributes should have K = 1.0). Thus agents with K values of 1.0 shall be preserved in the optimization process to allow other agents higher chances of adopting a K value closed to 1.0.

Throughout the optimization process, four agents having PrefWeight values of 0.2, 0.4, 0.6, 0.8 and "K" values of 1.0 shall not be evolved. The genetic algorithm adopts a 4-step process as shown in Fig. 7.

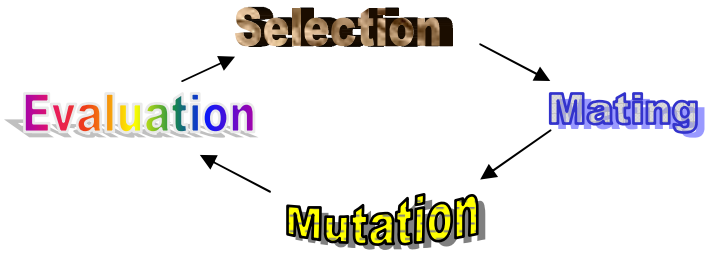

Fig. 7: A Genetic Algorithm Process

The system starts with a selection process whereby the parent agents are selected for mating. The system mates the selected agents to create offspring. Some of the offspring are mutated with a random mutation probability and the fitness of the entire population of parents and off spring is evaluated with respect to the rankings in the user feedback. The pseudo code of the algorithm can be found in Section 5, Figure 21.

Genetic algorithm involves the theory of survival of the fittest. The strongest agents have a chance to mate to increase the chances of producing stronger offspring. Upon mating and random mutation, the resultant population is evaluated and the strongest of the population are selected in the next iteration for mating.

3.4.1 Selection

There are various selection models available and this research uses the tournament selection method as it is proven to be very efficient [4] (Pg 56 Advanced Evolutionary Algorithm). In tournament selection, the agents are divided into subgroups and the winner of each subgroup will have the chance to undergo mating in the mating pool. A binary subgroup consisting of 2 agents per subgroup is chosen to increase the chances of lower fitness agents being selected in the mating process. The process continues until

convergence. Convergence occurs when no fitter generations or offspring can be produced in further iterations. This happens when the maximum possible fitness has been reached or the entire population is derived from a small number or ancestral parent agents.

3.4.2 Mating and Mutation

The attributes of each agent are converted into a binary string as shown in Fig. 8 and each binary bit representing a chromosome.

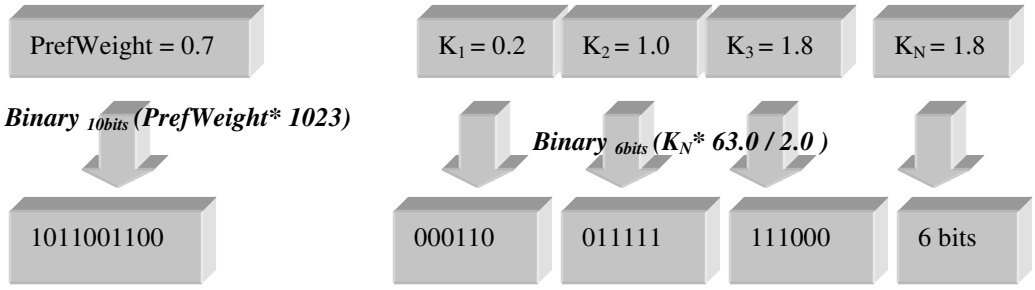

Fig. 8: Chromosome Encoding

In the design, 10 bits of data is used to represent the PrefWeight while 5 bits of data is used to represent the various K values. This allows us to estimate to a maximum error of 0.001 for PrefWeight and 0.03 for K which represents high accuracy. The various attributes are converted into an integer before converting into a binary string. As K ranges from 0.0 to 2.0, various values of K are divided by 2 in the conversion process.

During the mating process, a random crossover point as illustrated by Fig. 9 is selected among the chromosomes between the 2 parents and the new generation is produced by swapping the chromosomes of the parent agents across the random crossover point. The offspring produced will have a chance to undergo a mutation process according to a specified probability of 0.15. During mutation, a random number of bits or chromosomes are mutated and the binary symbol of the chromosome inverted (e.g. 1 becomes 0 and vice versa).

The fitness for each agent in the population is analyzed before the agents are selected into the mating pool for mating. The process continues until no improvement is noticed in the fitness level after 25 iterations.

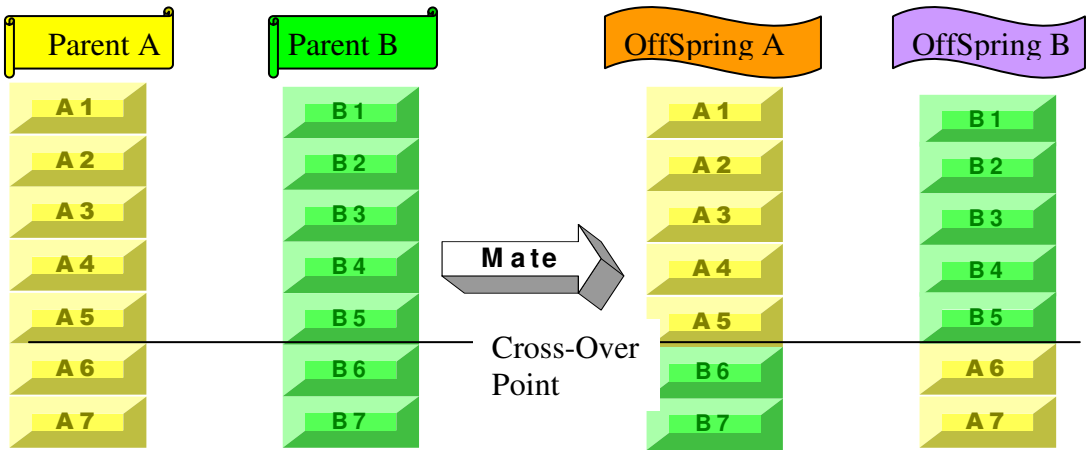

Fig. 9: Mating Process with Random Crossover

3.5 Controversial Situations

The genetic algorithm performed will return a set of optimized modification factor values for the detected attributes. This will identify the attributes affecting the user's preference. However some of the attributes identified may cause controversy as substituting them with other attributes may get the same level of fitness. This is shown in the screenshot in Fig. 10 whereby the 1st product is suspected to contain an unaccounted attribute that gives it an illogical ranking.

Rank	Description	Price	Quality
1	FUJITSU 17GB ATA-100	S$99.00	30.4097077...
2	SEAGATE 30GB Barracuda ATA-100 ST330620A	S$179.00	56.8441462...
3	SEAGATE 20GB Barracuda ATA-100 ST320414A	S$119.00	38.1761669...
4	SEAGATE 80GB Barracuda ATA-100	S$186.00	56.9206851...
5	MAXTOR 536DX 100GB 4W100H6 ATA-100	S$90.00	28.8568464...
6	IBM 40GB 40GV ATA-100	S$169.00	54.0798861...
7	IBM 120GXP 40GB ATA-100 2MB	S$127.00	37.0961761...
8	IBM 60GXP 20GB ATA-100	S$119.00	34.3893526...
9	MAXTOR D540DX 20GB4D020H1/4K020H1 ATA-100	S$105.00	28.0827214...
10	SAMSUNG SP4002H 40GB 7200rpm ATA-100	S$140.00	38.1642923...

Fig. 10: Unclear Situation - Multiple Equally Possible Attributes

Attributes distinct in this product and does not appear in other products include "FUJITSU" and "17GB". The Agents may capture "FUJITSU" as a detected attribute but "17GB" may have caused the illogical ranking instead. To solve this problem, the system compiles a list of attributes that are substitutes to the new list of detected attributes as shown in Fig. 11 below where we loop through each detected attribute and derive the potential substitutes from the products previously ranked. Consider the example in Fig.10 whereby the agents detect "FUJITSU" as a vital attribute. The system finds that the products containing "FUJITSU" also contains "17GB" and "ATA-100". The system loops through all other products (represented by PdtNo in Fig.11) and finds that "17GB" appears only when "FUJITSU" appears. The system also finds that "ATA-100" is not unique with the presence of "FUJITSU". Thus "17GB" is a potential substitute while "ATA-100" is not a possible substitute for "FUJITSU".

Upon compiling the detected attributes and their substitutes, the system generates a list of products according to the rules below to clarify the controversial cases.

1. Products containing the detected attribute and substitutes
2. Products containing either detected attribute or substitutes

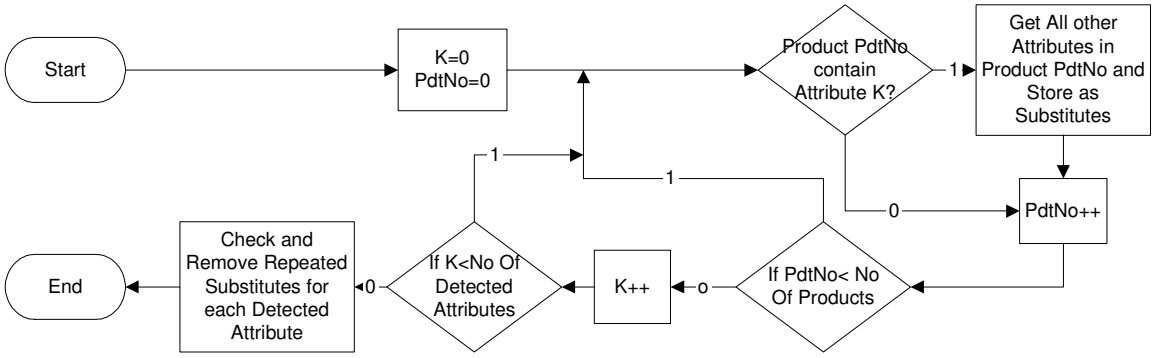

Fig. 11: Deriving Substitutes for Each Detected Attribute

The system will next substitute each detected attribute by the substitutes and check the fitness of the agent before and after the substitution. If the fitness level improves after substitution, the detected attribute shall be replaced by its substitute.

3.6 Generic Group of Quantifiable Attributes

Besides detecting unaccounted attributes, the system is also able to analyze the presence of generic groups of quantifiable attributes. The system checks for the possibility of classifying the detected attributes into generic groups and derive a formula to predict the user's response towards other attributes in the same group.

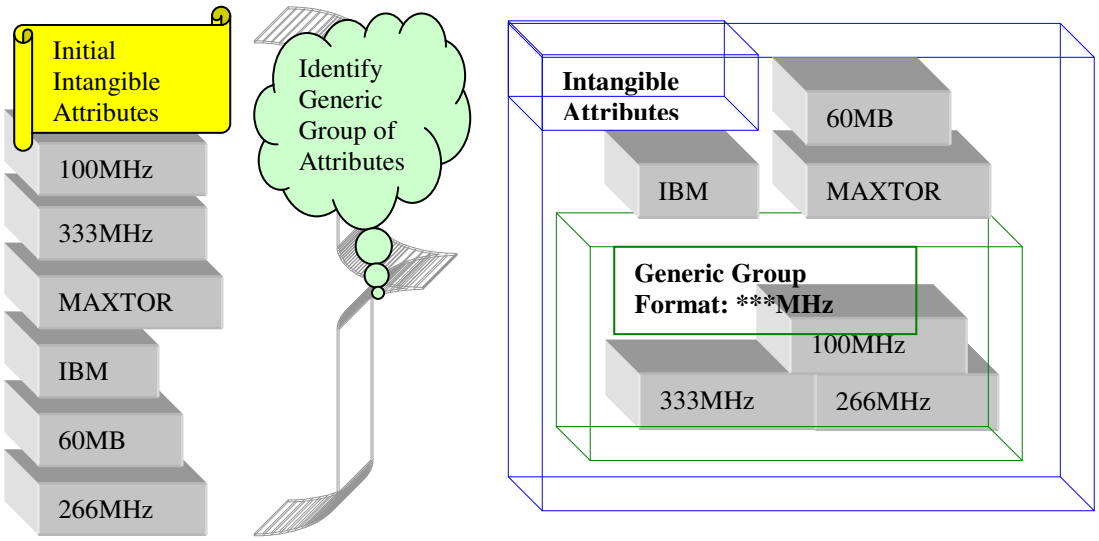

Fig. 12: Generic Attributes Identified from Intangible Attributes

When two or more detected attributes are of the same format except for its numerical values, they are classified into a generic group as demonstrated by Fig. 12 where attributes of the format ***MHz (*** represent some numeric values) are detected. An initial list of 6 attributes is detected. However, at least 2 attributes are detected to be of the form ***MHz and thus classified into a generic group of attributes. Although the attribute "60MB" is in the form "***MB", only 1 attribute of such property is found and thus

60MB is not sufficient to form a generic group. After detection, 1 generic group of attributes and 3 independent detected attributes are derived from the 6 original attributes.

An equation is derived from the attributes in the generic group to anticipate user's preference of other attributes in the generic group. This equation is derived as shown by the bold line in Fig. 13 using 2 detected attributes (Ovals) with highest and lowest K values (e.g. assume "100MHz" has K = 0.3 and "333MHz" has K = 1.6).

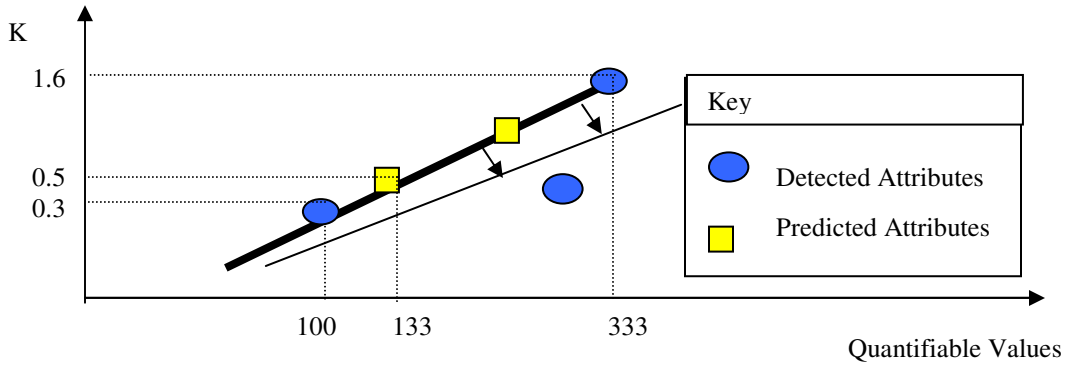

Fig. 13: Deriving and Optimizing Equation for Generic Group Attributes

From the bold line, the value of K for other attributes that are not detected but belongs to the same generic group is easily derived as shown by the squares. By utilizing the equation derived, "133MHz" although not detected, contains a Kvalue of 0.5. The straight line derived can further be optimized by shifting the gradient and Y-Intercept as shown by the arrows in Fig.13 to a best-fit line for the user.

The equation for the generic group is passed into the same genetic algorithm as discussed in section 3.4. The Y-Intercept (essentially K) takes a value between –8.0 and 2.0 for an equation with positive gradient and 8.0 to –2.0 for negative gradient equation. 7 bits are used to estimate the Y-intercept allowing high accuracy of resolution 0.08. The range of the gradient (possibly from 1.0 to 1000000!) may be too large to be optimized by

genetic algorithm. Thus the gradient is randomly mutated in the genetic algorithm iterations by multiplying it with a random number ranging from 0.2 to 5.0. This proves to be an effective solution as the initial estimation of the gradient shows good accuracy.

3.7 An Incremental Detection System and Overall Feedback Design

The system takes an incremental detection approach in understanding user preference and the results show success in analyzing complex user preference situation. The system acknowledges that not all vital attributes may be captured within one set of feedback and thus considers the results of previous sets. The attributes that affect a user's preference in 1 feedback become the prime candidates in the next set of feedback. In this way, the attributes that are detected are preserved and verified while new unaccounted attributes are being detected allowing the software agents to incrementally learn about the attributes that affect the user's preference. However some of the information captured by the system may be incorrect or no longer valid as the number of feedback cycles increase. This creates a problem in the incremental detection system, as the information may not be relevant. To solve this problem, the system checks the validity of past attributes influencing the user's preference and delete attributes that are no longer relevant in the current feedback. Every set of feedback contains 2 feedback cycles as shown in Fig. 14.

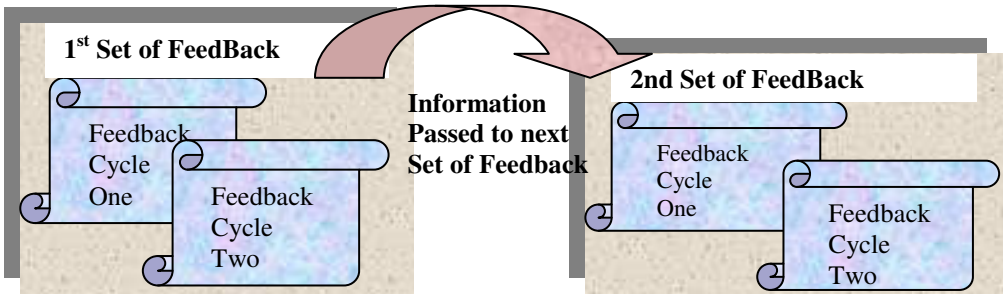

Fig. 14: One Set of Feedback Consisting of 2 Feedback Cycles

Both feedback cycles will attempt to detect the presence of any unaccounted attributes. In addition, the 1st cycle shall delete any attributes that are passed from previous feedbacks and no longer relevant. These attributes should have a K value of 1.0 after we apply the genetic algorithm discussed earlier. Any controversial attributes detected by the 1st cycle shall be clarified using the 2nd feedback cycle as described by section 3.5. The 2nd cycle shall attempt to detect unaccounted attributes from scratch to prevent getting trapped in a local minimum fitness that may be introduced by the 1st cycle. The 1st cycle may detect an attribute incorrectly and there may be a possibility that the 2nd cycle is not able to realize the mistake affecting detection of other attributes. All attributes detected by both cycles are finally optimized and checked for generic group of attributes described earlier.

Fig. 15: Recommended Products by the Agents

Upon completing one set of feedback, the user shall be able to view a list of recommended products from the agents by selecting the View Results button. The fittest agents shall interact to generate a list of the most suitable products for the user as shown in Fig. 15.

4 EVALUATION OF DESIGN

In this section, the prototype performance is tested and evaluated. An independent program is written and run in the background to simulate a user. This program is used to provide feedback to the system and ranks the list of products on behalf of a simulated user who is affected by Price and Quality as well as a list of unaccounted attributes. The system is also affected by some generic groups of quantifiable attributes.

The system performance shall be analyzed using 2 benchmarks, the Attribute Detection Rate and the Recommendation Fitness. The Attribute Detection Rate measures the percentage of the unaccounted attributes that are vital to the user being detected by the system. The Recommendation Fitness measures the fitness of the recommended list by the system. The system shall attempt to understand the user's preference and recommend a list of 10 most desirable products. This list is compared with the ideal top 10 products generated by the simulated user. Every product that appears in both lists is awarded 10 points and a total of 100 points for the 10 products is attainable.

4.1 Detection of Single Unaccounted Attribute

One unaccounted attribute along with 2 accounted attributes, price and quality is assumed to be affecting the user's preference and the user completes 3 sets of feedback with this preference. To test system adaptiveness, the user changes his preference during the 4^{th} set of feedback by adding another unaccounted attribute. The results are shown in Fig. 16 where the Attribute Detection Rate and Recommendation Fitness are analyzed using 50 samples.

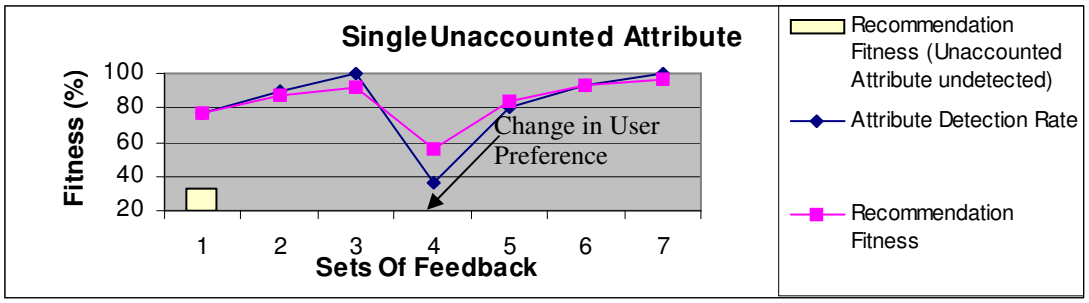

Fig. 16. Analysis of Attribute Detection Rate and Recommendation Fitness under the Influence of Single Unaccounted Attribute

The system obtains the Attribute Detection Rate at 76.7% after the user completes 1 set of feedback. About 80% of the products recommended by the system are indeed highly desired by the user. The Attribute Detection Rate and Recommendation Fitness increases to 100% and 92% respectively after 3 sets of feedback. In comparison, if the unaccounted attributes concerned are ignored, the Recommendation Fitness will only reach a maximum of 33%. During the 4th set of feedback the user changes his preference and the system is confused, resulting in a drop in performance. However this confusion is cleared at the next set of feedback and the Attribute Detection Rate and Recommendation Fitness restored.

4.2 Multiple Unaccounted Attributes

The capability of the system to detect multiple unaccounted attributes affecting the user's preference is analyzed and 2 unique cases are studied. The 1st case involves 2 mutually exclusive attributes that will not be present simultaneously in 1 single product. Any product is affected by only 1 unaccounted attribute. The 2nd case is more complex as 2 co-existing unaccounted attributes may be present in 1 single product.

A total of 50 samples are analyzed and results are as shown in Fig. 17 in which it is compared to the detection rate of having one unaccounted attribute. The user

experiences a shift of preference in the 6th feedback to test the adaptiveness of the system. From the results obtained, it can be seen that the detection capability of having two unaccounted attributes is lower than having one such attribute. However, the system is still able to have a detection rate of more than 50% in one set of feedback. The gap between the detection rate of the complex and simpler cases decreases with an increasing number of feedbacks. This demonstrates the ability of the system to incrementally detect the user's preference when a complex situation is present. The system also demonstrates its ability to adapt to changing user preference within two sets if feedback.

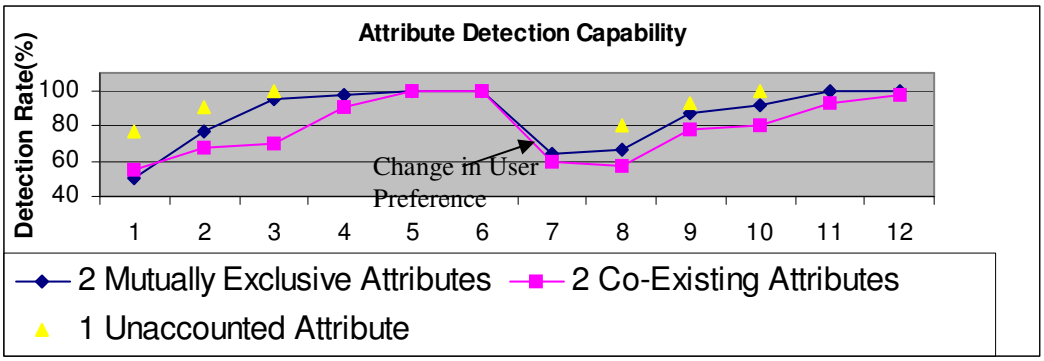

Fig. 17: Attribute Detection Capability for two Intangible Attributes

It is also observed that having co-existing attributes will cause the overall detection capability of the system to drop as demonstrated by comparing it to the mutually exclusive case. The result is expected as a product having two vital attributes to be detected creates a more complicated case as compared to only one attribute to be detected. However when the number of feedbacks increases, there is no significant difference in both situations. The system is able to incrementally detect almost all the attributes involved.

The Recommendation Fitness of both cases involving two unaccounted attributes is also analyzed and results as shown in Fig. 18. The result is as similar to the result

measuring the Detection Rate whereby performance drops when the test cases become more complex. However, the system is still able to produce high recommendation fitness using only 1 set of feedback. Out of 10 products recommended by the system, averages of 5 products are highly desired by the user. This increases to 8 by the 3rd set of feedback.

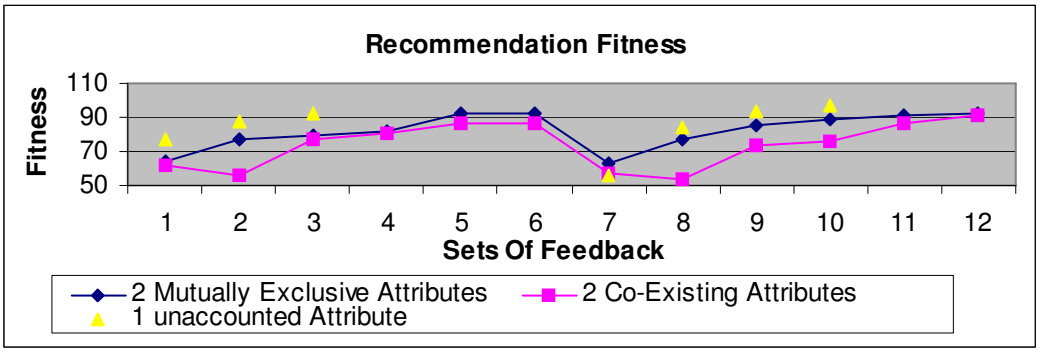

Fig. 18: Recommendation and Ranking Fitness for 2 Unaccounted Attributes

4.3 Generic Group Detection Capabilities and Adaptiveness

This measures the ability of the system to detect a generic group of quantifiable attributes and its ability to adapt to changes. In this test, the user considers speed of memory chips (represented by MHz) as a generic group of quantifiable attributes. Each speed is considered as a unique unaccounted attribute with a different Modification Factor, K, which increases with the speed of the memory chips. The Attribute Detection Rate for the generic group of attributes at different PrefWeight is analyzed using 15 test cases for each case and the results as shown in Fig. 19.

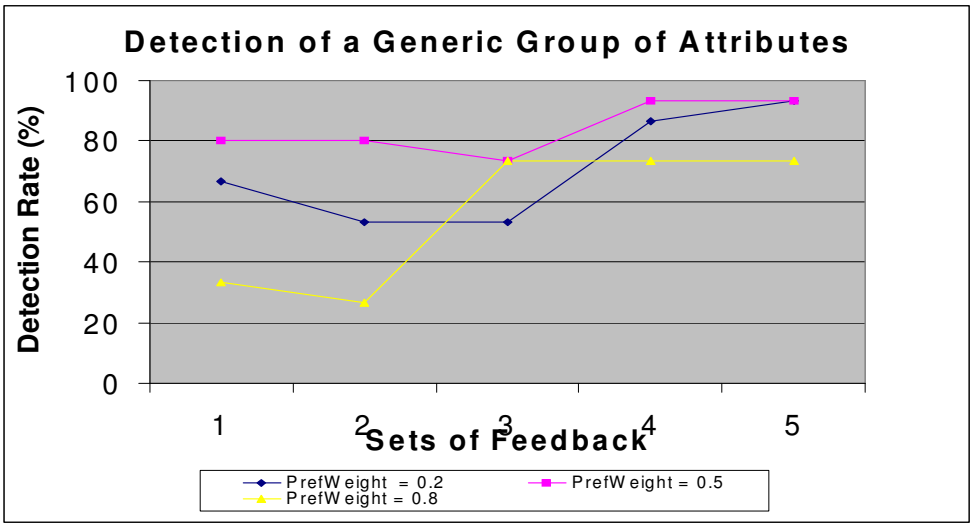

Fig. 19: Detection of Generic Attributes under Different PrefWeight

The Attribute Detection Rate is similar when PrefWeight takes the values of 0.2 and 0.5. However under the case when PrefWeight takes the value of 0.8, the system is not as capable in detecting the presence of the group of attributes. The user under such cases values the price competitiveness highly (represented by PrefWeight) and therefore desires a product with a lower price while placing less importance on quality. However, the user also prefers faster memory by indication of a positive "MHz" attribute creating confusion. Users who prefer fast memory in contrast should prefer a larger weight for quality and smaller weights for price competitiveness. Although the user's preference may seem contradicting and highly complex, the system is still able to have the Attribute Detection Rate as 70% after 3 sets of feedback are completed. This illustrates the ability of our incremental approach being able to understand the user's preference in a complex problem as illustrated when PrefWeight=0.8.

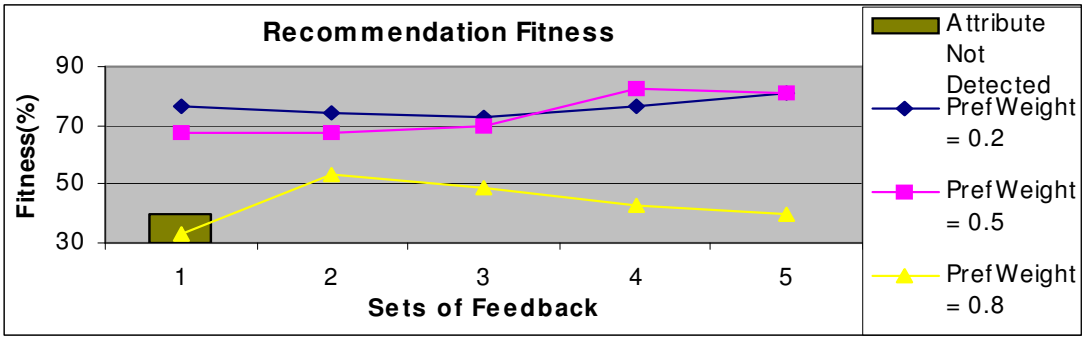

Fig. 20: Recommendation Fitness under Different PrefWeight for a Generic Group of Attributes

The Recommendation Fitness is also analyzed for the three PrefWeight using 15 test cases for each value of PrefWeight with results shown in Fig. 20. The system is able to produce a high Recommendation Fitness for PrefWeight values of 0.2 and 0.5. The fitness level increases with increased feedback due to better understanding of the user. From the Recommendation Fitness, almost 7 out of the 10 most desired products for the user is recommended by the system. This is a large improvement compared to the 40% fitness if the system is not able to detect the attribute concerned.

4.4 Design Choices

Some of the recent solutions towards an optimization process include Genetic Algorithm (GA), Tabu Search (TS) and Simulated Annealing (SA). To justify our choice of GA instead of TS or SA, there are some comparisons we have taken into account.

According to Jukka Kohonen [6], SA is very similar to GA, as SA can be thought as GA with a population size of one. Empirically, SA is able to reach good solution in a shorter time, but will not improve much as when given more time to progress. On the other hand, though GA is a slower starter, it is able to improve the solution consistently

over time, and can even reach more and better solutions. In fact, in our experiments, we are able to complete the optimization process using GA, in a relatively short time span.

Tabu Search (TS) has been proven effective in target-specific problems with the best solution in mind. However, the problem of optimizing multiple-attribute fitness is not really a target-specific problem. We cannot determine exactly what the user's preference is, and supposed "good" attribute may only have temporal wellness. Moreover, TS process concentrates on navigating towards a maxima or minima fitness and refrain from repeating a tested path. This will be a problem when user preference changes.

On the other hand, Genetic Algorithm (GA) has many advantages towards our problem. Traditionally, GA may be a very computationally intensive algorithm, taking easily 15min to compute. However, our processing time takes only around 10 to 15s, which is extremely fast and effective. In addition, GA is highly adaptive. We need a highly adaptive algorithm for two reasons: the user taste may change, and the algorithm may need to adjust its path of search. Using GA, we can effectively re-visit some of the solutions which we have discarded. GA has also the capabilities of handling a large number of variables. According to Chu and Fang [8], GA can produce several different near optimal solutions simultaneously because of the fact that GA holds the whole generation of chromosomes which may not originate from the same parents. GA also exhibits the capability of parallelism [7], by searching solutions from many points in the search space, rather than just one starting point.

In the following we compare our approach to the collaborative filtering approach [16]. At Amazon.com, though item-to-item collaborative filtering technique proved to be promising. The system relies mainly on the preferences of other customers in relation to

the current user. In traditional collaborative filtering recommendation, items are selected for a user when the items are also relevant to similar users. Based on the preferences of users who have similar taste as the current user, recommendation items are suggested. Building a recommendation list based on what other customers purchased or rated might not be useful, it can also be misleading and inefficient. Collaborative filtering functions primarily focus on the concept of suggesting what similar users have chosen, and not really focusing on what the current user really wants. Even with a well-thought algorithm, large product catalogs and customers' base will lead to high computational costs when trying to build a complete similar-item recommendation list.

Unquestionably, collaborative filtering is a popular way to model user preference. We might include collaborative filtering approach to extend our current research in future work. Within the focus of this research paper, we target only on identifying the current user preference. Our approach aimed to discover what the current user wants and we assumed the user has no access to or no interest in other users' preferences. Besides, focusing on the analysis of user preference allows closer monitoring the changes of user shopping preference. It allows the detection of atypical preference of the user with respect to similar users. We also have the flexibility to track preference of user with unusual taste. In particular, we are able to detect unaccounted attributes within the product that affect the user.

4.5 Summary of Results

It can be seen that the performance of the system is closely related to the complexity of the problem. More complex problems will give a lower overall performance as shown in

the various cases. However, this is greatly alleviated by providing multiple sets of feedback. The system incrementally detected attributes affecting the user's preference and in the cases shown, the gap in performance was negligible.

The system also demonstrated its ability to adapt to changes in consumer preference. This is extremely important when multiple sets of feedback are involved as the user's preference may vary between feedback cycles. It also demonstrated the system's ability to correct its own mistakes and search for a better solution.

5 IMPLEMENTATION DETAILS

The prototype system is implemented using Java programming language. In this section, we summarize what kind of program and data structure is used for the genetic algorithm and agents.

5.1 Program and Database Structure

A Microsoft Access database file is used to store all the tables of information. Below are the tables used and their descriptions.

Tables	Descriptions
`AgentInfo`	Contains all information of the agents.
`AttributeInfo`	Contains all information about the attributes.
`RankListHistory`	Contains ranked user feedback.
`IntangibleAtt`	Contains highly controversial product information, which may contribute up to 20% of the attribute score.
`Products`	Contains the database of all the products. Quality is being randomized with respect to price and the database is downloaded from a computer e-commerce website.
`Catinfo`	Contains the category information such as most and least expensive product price, and best and worst quality information.
`UserPref`	Contains the information of the simulated user.

Table 1: Database Tables

The agents form a group with no particular structure or organization. The group of agents works as a team, exchanging vital information throughout the evolution process. All agents are stored using array structures during evolution, as arrays have lower time consumption overheads. Single standard random point crossover is used during mutation to save processing since single crossover proved sufficient in our experiments.

5.2 Genetic Algorithm Design

```
// Agents contain all info about the agents estimation towards user's
// PrefWeight and K values
// Agent 0 to 4 each contains PrefWeight Value 0.2,0.4,0.6,0.8 and K = 1.0
// respectively
// This is to act as a strong check to allow the system to evolve quickly for
// PrefWeight by adding diversity and check for K=1.0 which is common in the
// evolution process

int NoOfCycleNoImprove = 0;   // No Of Iterations No Improvement
double BestFitness = 0;
int FittestAgent = 4;

Agents = GetAgentsFitness();  // Getting the fitness of the agents

// Loop till no improvement in fitness for 20 cycles or
// when maximum possible fitness for
// the 16th fittest agent is reached (Agents [20]>=1.0).
while (NoOfCycleNoImprovement < 20 && Agents[20].Fitness < 1.0)
{
        BestFitness = Agents[intFittestAgents].Fitness;
        Agents = SortAgents();// Sort Agents According to Agent's fitness
                              // Only Agent 4 onwards is sorted. This will
                              // allow the 1st 4 agents to be conserved.
                              // These agents help to track changes in user
                              // preference. Section 3.4
        MatingSelection();          // Tournament Selection Process.
                                    // Agents 0 to 3 is always selected
        Agents = Mate();            // Selected Agents Mate
        Agents = GetAgentsFitness();  // Getting the fitness of the agents
        if (Agents[intFittestAgent].Fitness <= BestFitness)
                intNoOfCycleNoImprovement++;
}
```

Fig. 21: Pseudo Code for Genetic Algorithm

As mentioned in Section 3, Genetic Algorithm is used to optimize the attributes. In particular, we coded the K values into binary chromosome as mentioned earlier in Section 3.4. The value of K ranges from 0.0 to 2.0 in steps of 0.05. Thus, in total there are 40 possible values of K. We convert this into a binary number and we have a set of binary chromosomes. For example, using 5 bits, we can encode K from 0.0 to 2.0 in 64 possible binary chromosomes as shown below.

00000 = 0/3*2.0	00001 = 1/63*2.0	00011 = 3/63*2.0	11111 = 63/63*2.0

The numbers on the left hand side are binary strings forming the chromosomes. Figure 21 shows the pseudo code of the genetic algorithm.

6 CONCULSION & FURTHER IMPROVMENT

In summary, this research work demonstrated a solution in the handling of previously unaccounted attributes without the need of change in the ontology or database design. An intelligent detection and verification scheme through the genetic algorithm has been implemented with success by using a prototype based on JAVA.

6.1 Results and Discussions

The results showed that the system designed is indeed capable of understanding the user's needs and preferences even when previously unknown or unaccounted attributes were present. The system is also able to handle the presence of multiple unaccounted attributes and classify quantifiable attributes into a generic group of unaccounted attributes.

In addition, the system demonstrated the power of incremental detection of unaccounted attributes by passing the detected attributes from within 1 feedback to the other. This is demonstrated by the usefulness of the system under extremely complex cases. The overall fitness of the system's recommendation for the user increases through the use of incremental feedback. The system is also adaptive to changes in user preferences.

6.2 Future Improvement

The current system generated user feedbacks to clarify its doubts on controversial attributes. However, more than half of the feedbacks were generated in random to increase

the chances of capturing new attributes. These random feedbacks were generated with products of different brand names having equal chances of being selected to add to the variety of the products used for feedbacks. This could be improved by generating feedbacks to test certain popular attributes to increase the detection capabilities.

References

[1] http://www.idc.com/

[2] http://www.media.mit.edu/getwari/MARI/

[3] Sheng-Uei Guan, Chon Seng Ngoo and Fangming Zhu, "HandyBroker - An Intelligent Product-Brokering Agent for M-Commerce Applications with User Preference Tracking", Electronic Commerce and Research Applications, 314-330, Vol. 1, No. 3-4, Autumn-Winter 2002

[4] Andrzej Osyczka. Evolutionary Algorithms for Single and Multicriteria Design Optimization. Physica-Verlag 2001

[5] Sybil Shearin, Henry Lieberman. Intelligent Profiling By Example. Available at http://sibyl.www.media.mit.edu/people/sibyl/projects/apt/final-shearin-iui.pdf

[6] Jukka Kohonen, A brief comparison of simulated annealing and genetic algorithm approaches, 1999, http://www.cs.helsinki.fi/u/kohonen/papers/gasa.html

[7] Genetic Algorithms vs Traditional Method, http://www.brunel.ac.uk/depts/AI/alife/ga-versu.htm

[8] Chu S. C., Fang H. L, Genetic Algorithms vs. Tabu Search in Timetable Scheduling, 1999 Third International, Conference on Knowledge-Based Intelligent Information Engineering Systems, Adelaide, Australia.

[9] Marios Koufaris, Ajot Kambil, Priscilla Ann Labarbera. Consumer Behaviour in Web-Based Commerce: An Empirical Study. International Journal of Electronic Commerce/, Vol. 6, No.2, pp. 115-138, Winter 2001-2002.

[10] Chanan Glazer, Surya B.Yadav. A Conceptual Model of an Intelligent Catalog Search System. Journal of Organizational and Electronic Commerce 11(1), pp.31-46, 2001.

[11] Fangming Zhu and Sheng-Uei Guan, "Towards Evolution of Software Agents in Electronic Commerce", Proceedings, the 2001 Congress on Evolutionary Computation (CEC2001), Korea, 1303-1308, May 28-30, 2001.

[12] Soltysiak, S., and Crabtree,B., Automatic learning of user profiles – towards the personalization of agent services, BT Technology Journal 16(3), 110-117,1998.

[13] Sheth, B., and Maes, P., Evolving agents for personalized information filtering, MIT Media Lab, 1993.

[14] Petra Schumann, Easy Shopping: A Value-Added Service for electronic Malls, International Journal of Electronic Commerce, pp. 99-119 Vol. 4, No. 2, Winter 1999-2000.

[15] Fangming Zhu and Sheng-Uei Guan, "Evolving Software Agents in Electronic Commerce", Proceedings, International Symposium on Distributed Intelligence in Technology, Economic and Social Applications (DI-TESA2001), Tucson, Arizona, USA, 3297-3302, October 7-10, 2001.

[16] Greg Linden, Brent Smith and Jeremy York, Amazon.com Recommendations: Item-to-Item Collaborative Filtering, IEEE Internet Computing, Jan/Feb 2003