Overcoming Incomplete User Models in Recommendation Systems Via an Ontology

Vincent Schickel-Zuber and Boi Faltings

School of Computer and Communication Sciences – IC
Swiss Federal Institute of Technology (EPFL)
Lausanne, Switzerland
{vincent.schickel-zuber, boi.faltings}@epfl.ch

Abstract. To make accurate recommendations, recommendation systems currently require more data about a customer than is usually available. We conjecture that the weaknesses are due to a lack of inductive bias in the learning methods used to build the prediction models. We propose a new method that extends the utility model and assumes that the structure of user preferences follows an ontology of product attributes. Using the data of the MovieLens system, we show experimentally that real user preferences indeed closely follow an ontology based on movie attributes. Furthermore, a recommender based just on a single individual's preferences and this ontology performs better than collaborative filtering, with the greatest differences when little data about the user is available. This points the way to how proper inductive bias can be used for significantly more powerful recommender systems in the future.

1 Introduction

Consider a situation where you find yourself with an evening alone and would like to rent a DVD to watch. There are hundreds of movies to choose from. For several reasons, this is a difficult problem. First, most people have limited knowledge about the alternatives. Second, the set of alternatives changes frequently. Third, this is an example of a low user involvement decision process, where the user is not prepared to spend hours expressing his preferences. Recommender systems, RS, have been devised as tools to help people in such situations. Two kinds of techniques are widely used in e-commerce sites today.

The first technique is item-to-item collaborative filtering (CF, [11]), which recommends products to users based on other users' experience. Amazon.com[1], with over 29 million customers and several million catalog items [11], uses this technique which is more commonly known to end-user as "Customers who bought this item also bought these items:". CF generates recommendations based on the experience of like-minded groups of users, based on the assumption that similar users like similar objects. Therefore, CF's ability to recommend items depends on the ability to successfully identify

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

O. Nasraoui et al. (Eds.): WebKDD 2005, LNAI 4198, pp. 39 – 57, 2006.

the set of similar users, known as the target user's neighbourhood. CF does not build an explicit model of the users preferences. Instead, preferences remain implicit in the ratings that the user gives to some subset of products, either explicitly or by buying them. In practice, CF is the most popular recommendation technique and this is due to three main reasons. First, studies have shown it to have satisfactory performance when sufficient data is available. Second, it can compare items without modeling them, as long as they have been previously rated. Finally, the cognitive requirements on the user are low. However, as argued by many authors [12][13][18] [19][21], CF suffers from profound problems such as:

- *Sparsity*. This is CF's major problem and occurs when the number of items far exceeds what an individual can rate.
- *Cold start*: When a new user enters the system, he is usually not willing to make the effort to rate a sufficient number of items.
- *Latency problem:* Product catalogs evolve over time; however, the collaborative approach cannot deal with new products as they have not been previously rated.
- *Scalability.* The computation of the neighborhood requires looking at all the items and users in the systems. Thus, the complexity grows with the number of users.
- *Privacy*. In most systems, the similarity matrix is located on a server and is accessible to a third party, thus raising privacy concerns.
- *Shilling attacks*: malicious users can alter user ratings in order to influence the recommendations.

The other widely used technique is preference-based recommendation. Here, a user is asked to express explicit preferences for certain attributes of the product. If preferences are accurately stated, multi-attribute utility theory (MAUT, 10) provides methods to find the most preferred product even when the set of alternatives is extremely large and/or volatile, and thus has no problems of sparsity, cold starts, latency or scalability. Furthermore, since recommendations are based only on an individual user's data, there are no problems with privacy or shilling. However, the big drawback of preference-based methods is that the user needs to express a potentially quite complex preference model. This may require a large number of interactions, and places a higher cognitive load on the user since he has to reason about the attributes that model the product.

However, attribute-based preference models can also be learned from user's choices or ratings, just as in collaborative filtering. In our experiments, this by itself can already result in recommendations that are almost as good as those of collaborative filtering. The main novelty of this paper, however, is to use an ontology of product attributes to provide an inductive bias that allows learning of this individual preference model to succeed even with very few ratings. This leads us to a technique for recommender systems that outperform the best known collaborative filtering techniques on the MovieLens data that we have been using for our experiments. Furthermore, very few ratings, just 5 movies, suffice to get recommendations that are almost as good as what can be reached with many, 30 or more, ratings. At the same time, the user effort required by this technique is not significantly different from collaborative filtering system. Thus, we can effectively get the best of both techniques.

This paper is organized as follows: Section 2 provides fundamental background in Collaborative Filtering, Multi—Attribute Utility Theory, and Ontology Reasoning, while our novel approach with its algorithm is explained in Section 3. Section 4 contains experimental results with comparison to existing techniques. Finally, Section 5 provides the conclusions.

2 Existing Techniques

Our approach uses the cognitive simplicity of Collaborative Filtering whilst maintaining the advantages of the Multi-Attribute Utility Theory. This is achieved by employing the knowledge of an ontology and reasoning over its concepts instead of the item's content itself. Before defining our model, we start by introducing fundamental background.

2.1 Collaborative Filtering

In pure collaborative filtering systems, users state their preferences by rating a set of items, which are then stored in a user-item matrix called *S*. This matrix contains all the users' profiles, where the rows represents the users $\mathbf{U} = \{u_1,\ldots,u_m\}$, the columns the set of items $\mathbf{I} = \{i_1,\ldots,i_q\}$, and $S_{k,l}$ the normalized rating assigned to item l by the user k. Given the matrix S and a target user u_i, a user-based CF predicts the rating of one or more target items by looking at the rated items of the *k* nearest neighbors to the target user. On the other hand, an item-based CF algorithm looks at the *k* most similar items that have been co-rated by different users. Due to the complexity of the user-based CF, we will not consider it and focus on item-based CF.

The first step of item-based collaborative filtering is to compute the similarity between two co-rated items. Many similarity measures exist, but the most common one is the *cosine metric*, which measures the angle between two vectors of size m. When 2 items are similar, then the angle between them will be small and consequently give a cosine value close to 1; conversely, when items are very different then the value is close to 0. Over the years, the cosine metric has been updated in order to take into account the variance in the ratings of each user. For example, the *adjusted cosine similarity* subtracts from each user rating $S_{j,a}$, the user's average rating. Once all the similarities have been computed, CF estimates the rating of an item *a* by selecting the *k* most similar items to *a*, and computes the weighted average of the neighbor's rating based on the similarity the neighbors have with a.

Unfortunately, as the number of items and/or users grows so will the data sparsity in the matrix. This is CF's fundamental problem and explains why numerous authors [12][13][21] have focused their work to try to overcome it. Data-mining [21] or the Two-way Aspect Model [19] are now used to extract the item similarity knowledge by using association between the user's profile [21] and the object's content [19] in order to augment a standard similarity matrix. To overcome the latency problem, the content of the object has also been used to try to predict its rating. This is achieved by filling up the similarity matrix [4] or simply by using a weighted combination [12] of the content and collaborative prediction.

2.2 Multi-attribute Utility Theory

Formally, each item is defined by a set of n attributes $\mathbf{X} = \{X_1,\ldots,X_n\}$. Each X_i can take any value d_i from a domain $D_i = \{d_1,\ldots,d_k\}$. The Cartesian product $\mathbf{D} = D_1 x\ D_2 x\ldots\ x\ D_n$ forms the space of all the possible items **I**. The user expresses his preferences by defining the utility function and weight of each attribute. The simplified form of the Von Neumann and Morgenstern [23] theorem states that, if item x is considered better or equivalent to item y, then there exists a utility function *u* such that u(x) is bigger or equal than u(y). In this paper, we do not consider uncertainty, therefore the utility function becomes equivalent to a value function, but later we consider expected values of similarity.

Furthermore, if we assume Mutual Preferential Independence (MPI, [10]), then the theorem of Additive Value Function[10] can be used , and we can define the utility V of an item o_k as the sum of the sub-utility functions v_i of item o_k on each attribute multiplied by its weight w_i. This is commonly called the Weighted Additive Strategy (WADD, [10]), and the item with then highest overall evaluation value is chosen as the optimal solution.

$$V(o_k) = \sum_{i=1}^{n} w_i v_i(o_k) \tag{1}$$

Theoretically, when the MPI assumption holds, this strategy can achieve 100% accuracy if all the parameters can be precisely elicited. Unfortunately, the elicitation of the parameters is expensive and various authors have tried to simplify this elicitation process in order to make it usable in real systems. Stolze et al. [20] for example, exploit the idea of a scoring tree where a user expresses his preferences by modifying an existing tree via the use of rules. Once the preferences have been elicited, the system translates the scoring tree into a MAUT additive value function and then searches in the catalog for the most suitable products. Incremental Utility Elicitation [7], IUE, is another approach that eases the elicitation by an incremental process that interleaves utility elicitation and filtering of the items based on the elicited information. A major contribution in that domain is the work done by Ha et al [7][8], where polyhedral cones and pair wise comparison are used to estimate the user's true weights. Also, [8] makes the assumption that the utility function has a multi-linear form, and that all the sub-utility functions are known. Regrettably, computing the cone is a hard problem that makes it unsuitable for real life scenarios. More recently, Blythe in [2] has simplified the process by assuming MAUT additive value functions and used a linear programming formulation and pair wise comparison of alternatives to estimate the user's true utility. Nevertheless, all of the mentioned approaches work only for a small number of attributes. This makes them difficult to apply to real life scenarios where alternatives are modeled by many features.

2.3 Ontology Reasoning

With the emergence of the Semantic Web, it has been widely accepted that ontologies can be used to model the world we live in. In its general form, an ontology is a lattice,

where a node represents a concept (i.e.: an object of the world we want to model) whilst the edges between concepts correspond to their semantic relations.

One of the simplest forms of an ontology is the concept tree (CT); A graph where the topology is a tree with only *is-a* relations. The tree structure makes the reasoning computationally efficient and the modeling of the domain easy. Despite its simplicity, a CT can greatly enhance modeling the domain and the filtering process. In [3], Bradley et al. have successfully used a concept tree to model the job domain and shown through experimentation that it is very useful for personalization.

Concept similarity is the predominant form of ontology reasoning. Most techniques use the distance between the concepts in a concept tree or similar graphical ontology to estimate their similarities; the smaller the distance between two concepts the more similar they are. In [3], for example, distance was simply the number of edges between the concepts, while Yang and al. in [24] used the depth of the lowest common ancestor. In [16], Resnik defined similarity based on the information content shared by the concepts rather than its distance. Resnik's metric postulates higher similarity among rare concepts. While this makes sense for concepts themselves, there is no reason why this would also hold for preferences.

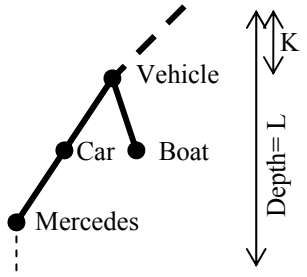

Fig. 1. A CT on transports

The metrics in [3][16][24] assume that the similarity between two concepts is symmetric, i.e.: the similarity between concepts A and B is identical to the similarity between concepts B and A. However, in real life situations, the distance should be considered asymmetric. Consider for example the concept tree in Fig. 1. If a user liked any kind of vehicle, then he will probably like Mercedes cars to a similar degree. On the other hand, liking Mercedes cars does not necessarily mean liking any vehicle as some people become sick on boats.

Formally, and assuming that P(X) corresponds to the probability of X occurring [16], and P(V ∩ M) is equal to α, then the probability P(V | M) is equal to $\alpha \times (L/(K+2))$ while P(M | V) is equal to $\alpha \times (L / K)$. This implies that P(V | M) < P(M | V), which means that the similarity function is asymmetric.

Andreason et al. in [1] defined an asymmetric metric based on the principle of upward reachable nodes. Their metric can be applied to any graph structure, and differentiates between the cost of traveling upward or downward the ontology. Three important postulates that reflect common sense intuitions are defined in [1]:

1. *Generalization cost property*. The cost of generalization should be significantly higher than the cost of specialization.
2. *Specificity cost property*. The cost of traversing edges should be lower when nodes are more specific.
3. *Specialization cost property*. Further specialization reduces similarity.

The generalization cost property models the asymmetry of the similarity function, which implies that the similarity function is not a metric. The specificity cost property represents the fact that sub-concepts are more meaningful to the user than super-concepts, whilst the specialization property reflects the fact that the further away two concepts are, then the more dissimilar they become. As a consequence, the specificity property reduces the cost of traversing edges as we go deeper in the ontology and the specialization property increases the cost between two concepts if other concepts are found on the path between these concepts.

3 Heterogeneous Attribute Preference Propagation Model

The utility model defined in section 2 is a powerful tool as long as we have a complete user model, and all the items are defined by the same set of attributes. However, in volatile environments such as the web, products continuously change, yet the utility model can only compare outcomes if they have the same attributes. In this section, we introduce the *heterogeneous attribute preference propagation model,* HAPPL, which is capable of building an accurate user model from implicit preferences. HAPPL relaxes the previous two conditions and uses two key components to evaluate the user's parameters:

- *Ontologies* to model and propagate utility values.
- *Multiple regression* to estimate the weights.

The ontology is the key ingredient of our model and we assume that the one required for modeling our attribute is already existing and available. [14][15] were first to propose the idea of using a concept hierarchy directly inferred from the website structure to enhance web usage mining. Major e-commerce sites such as Amazon and Yahoo also uses concept tree to model the item they sale, and to the best of our knowledge, these ontology were hand crafted. This work does not consider how such an ontology was constructed as this is way beyond the scope of this paper. Currently, most ontologies are hand crafted by experts as no fully automated tools are yet available. However, the clustering community has promising result with non-supervised incremental hierarchical conceptual algorithm such as COBWEB [6].

For our experiments, we built a movie recommender system and the ontology modeling a movie. The concept tree was built from scratch based on our common sense as there is none modeling the movie domain. We did not learn it on the data set, but instead we used term definitions found in various dictionaries. To give an idea of the resources require to build the ontology, it took the author about one working day to conceive it.

3.1 Basic Idea

To illustrate the principle that ontology can be used to model and estimate user preferences, take the following problem, where items are defined by a set of 3 attributes X_1, X_2, and X_3. For our DVD example, X_1 could be the theme, X_2 the duration of the

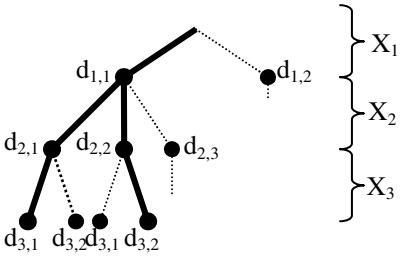

Fig. 2. Representation of the Attributes

movie, and X_3 its MPPA rating. Fig. 2 illustrates a possible representation of the domain as a concept tree, where each level represents an attribute and the different concepts at depth i the possible domain values of the attribute i. Let item o be defined by the domain values $\{d_{1,1}; d_{2,1}; d_{3,1}\}$ and item p by $\{d_{1,1}; d_{2,2}; d_{3,2}\}$. Again, Fig. 2 shows clearly that the 2 items have some similarity as they both share the node $d_{1,1}$, which means that they both have this domain value in common.

We define a simple similarity function (isim) between 2 items o and p as the number of attributes they share over the total number of possible attributes. This normalization guarantees that the item similarity value lies in the interval [0..1].

$$isim(o, p) = |\{x \mid x(o) = x(p)\}| / |\{X\}| \tag{2}$$

where x(o) is the set of attributes defining item o, and {X} is the set of all attributes. Furthermore, this equation assumes that each attribute is equally likely to contribute to whether one likes an item, and that each item is defined by the same number of attributes. In our example and using equation (2), we can deduce that items p and q have a similarity equal to 1/3. Informally, this means that if the user liked item p, then there is one chance out of three that he will like outcome q. Notice that the additive utility model makes the assumption that if a user has liked features A and B, then he/she will like items containing A+B. This assumption is not made in probabilistic models, where the features and their combination are considered to be completely independent. This introduces an inductive bias that we think is realistic in most real life situations; and this is supported by our experimental results.

In volatile environments, items do not always have the same number of attributes. For example, suppose that item o is defined by the values $\{d_{1,1}; d_{2,1}\}$, q by $\{d_{1,1}; d_{2,2}\}$ and p by $\{d_{1,1}; d_{2,2}; d_{3,2}\}$. Following equation (2), the similarity between o and q is equal to 1/3, which is also equal to the similarity between o and p. Furthermore, the similarity between o and q is equal to the similarity q and o. However, both situations are incorrect as the former violates the specialization property, while the latter violates the generalization cost property.

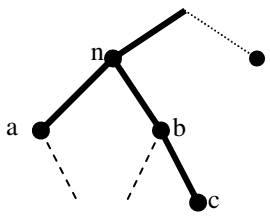

Fig. 3. A simple tree

According to ontology theory, if two concepts are closely related in terms of distance in the graph, then we can assume that they are very similar in terms of meaning. Furthermore, the similarity between two concepts is equal to one minus the distance between those concepts, where the distance is defined as a metric. Consider for example the simple generic graph of Fig. 3. The distance between nodes a and b can be given as the distance between node a and its ancestor n plus the distance between node n and b. However, due to the generalization property, the distance from a to its ancestor should be higher

than the distance from the ancestor to b. This property implies that the distance is in fact asymmetric; taking into account that the transitivity property of the distance, we can define the asymmetric distance between two nodes as:

$$adist(a,b) = \rho \times dist(a, lca_{a,b}) + (1-\rho) \times dist(lca_{a,b}, b) \tag{3}$$

where ρ is a generalization coefficient, and $lca_{a,b}$ is the lowest common ancestor to node a and b. Notice that this coefficient implements the generalization property stated in section 0, which means that it must be included in the interval (0.5, 1]. In practice, ρ is learnt by trial and error on a subset of data. In our experiment, we have found that the optimal value lies close to ¾. However, this coefficient is task dependent and will be greatly influenced by the ontology, which is also task dependent.

After substituting the distance by one minus the similarity and simplifying equation (3), the asymmetric similarity can be decomposed between two nodes a and b as:

$$asim(a,b) = \rho \times sim(a, lca_{a,b}) + (1-\rho) \times sim(lca_{a,b}, b) \tag{4}$$

Furthermore, and by considering equation (2), we can define the similarity between a node a and the closest common ancestor of node a and b as the number of nodes on the path from the root to the common ancestor over the number of nodes on the path from the root to node a.

$$sim(a, lca(a,b)) = |URN(lca_{a,b})| / |URN(a)| \tag{5}$$

In this equation, URN(a) is the set of upward reachable nodes from node a to the root of the tree, which corresponds to the path of a to the tree root. Subsequently, URN($lca_{a,b}$) is the set of nodes from the closest common ancestor of both nodes a and b to the root. Note that in our model, the root node is not contained in the path.

By considering equations (4) and (5), we improve equation (2) to measure the similarity between heterogeneous items o and p, which are respectively modeled by concept c and d as:

$$asim(c,d) = \rho \times \frac{|URN(lca_{c,d})|}{|URN(c)|} + (1-\rho) \times \frac{|URN(lca_{c,d})|}{|URN(d)|} \tag{6}$$

Following this, if we know that a user has liked concept c with a score equal to x, then we can estimate that he or she will also like concept d by a score equal to x×asim(c, d). Consequently, if we know that a user has liked the domain value $d_{2,1}$ by a value y, then we can estimate that they will like $d_{2,2}$ by a value equal to y×asim($d_{2,1}$, $d_{2,2}$). Hence, we can make use of concept tree as user model to store the user's utility values and also estimate missing ones by looking at the similarity between the concept representing the missing value and the nearest concept on which the user has expressed a preference.

Once all the utility values are known, we propose a novel way of estimating the weights. Rather than eliciting the weights from the user, we suggest to estimate them by multiple regression. This is made possible by the fact that each user has rated a given number of items, and thus we can obtain a system of n equations with n unknowns.

3.2 Definitions

Section 0 introduced the idea that attributes could be modeled by a concept tree, and that a concept tree could be used to estimate missing values. We can extend this idea on domain values, as they usually also have some kind of relationship between them. For example, if you like *action* movies then you will probably also like *adventure* movies, as action and adventure are closely related. We represent this information by an *attribute concept tree*, where the domain values are modeled by concepts, the value of the concept represents the user's utility value, and the relationships between them are is-a relations.

Definition 1. *An attribute concept tree, ACT, is defined by the 8-tuple: <X, D, T, ≤, DCT, f, sim, comb >, where*

- X is a concept representing the attribute X, and linked to *DCT* with the binary relations *hasDom*.
- D is the domain of the attribute X.
- T defines the type of all the elements in D.
- ≤ is an ordering relation on all the element of D.
- DCT is a concept tree modeling D, with the *null* concept as the root.
- f is function f:d →c, where d ∈ D, c ∈ DCT.
- sim is a function that computes the similarity between any two concepts in DCT.
- comb is a function that estimates the value of a concept in DCT.

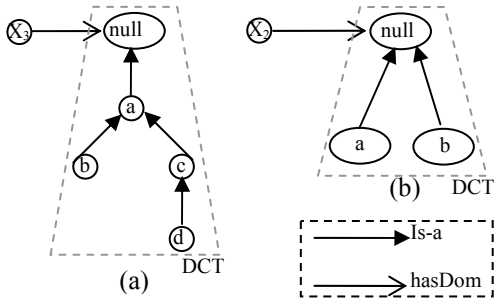

Fig. 4. (a) ACT with is-a relationship between domain values, and (b) without

Depending on the nature of the domain D, the DCT is built differently. When the domain D is discrete, the HAPPL technique exploits the *is-a* relationships in the domain D by building a hierarchy (Fig. 4.a). On the other hand, if a domain value is without any relationship, then it will be directly attached to the null concept (Fig 4.b). The null concept represents the null domain value, and is used if an object does not contain that attribute. However, if D is continuous, we tend to use the classic ordering (i.e. ≤), and each element of D is directly connected to the root concept as before.

Hence, if we know that two concepts are linked to the root concept, then we can deduce that they have nothing in common, and the similarity function will return zero.

In many situations, different attributes represent the same concept or have similar meaning. In the computer domain, for example, the attributes processor speed and RAM are related because large memory usually implies a fast processor. Our model exploits this pattern by grouping similar attributes together in a set called a *compound attribute*.

Definition 2. *A compound attribute concept tree CACT is defined by the 7-tuple: <CA, Z, ≤, AsCT, f, sim, comb >, where*

- CA is a compound attribute concept representing the compound attribute CA, and linked to AsCT with the binary relations *hasDom*.
- Z is the set of attributes in CA.
- $\leq$ is an ordering relation on all the element of Z
- AsCT is a concept tree ontology modeling all the elements in Z and with the *root* concept as the root.
- f is function f:d→c, where d ∈ Z, c ∈ AsCT.
- sim is a function that computes the similarity between any two concepts in AsCT
- comb is a function that estimates the value of a concept in AsCT.

Note that the concepts of a compound attribute concept tree are in fact attribute concept trees. Informally, a CACT can be seen as a bi-dimensional ontology, where one dimension represents the attributes, while the other represents their corresponding domain values. Similarly to an ACT, if 2 attributes have no relation between them, then they will be attached to the root concept.

Given an ontology λ made of CACTs, we see the heterogeneous attribute preference propagation model (HAPPL) as a possible extension of the multi-attribute utility theory (MAUT), where all the outcomes are modeled by λ, and where the mutual preferential independence hypothesis holds on all the attributes. By generalizing the MAUT, we need to generalize equation (1). Thus given a problem where items are defined using the HAPPL model, the optimal solution is the item that maximizes the utility in equation (7).

$$V(o_k) = \sum_{i=1}^{m} w_i' v_i'(o_k) \quad , where \quad v_i'(o_k) = \sum_{j=0}^{n_i} w_j v_j(o_k) \tag{7}$$

In this equation, m is the number of compound attributes defining the class modeling the items, and n_i is the number of attributes in the compound attribute i. The sub-utility functions v_j and v'_i, and the weights w'_i of the compound attributes are defined on the interval [-1, 1] and $\sum|w_i'|=1$. However, the weights w_j are defined on the interval [0,1] and $\sum w_j=1$. Notice that this slightly differs from utility theory where utilities and weights are defined on the interval [0, 1]. This distinction was necessary to reflect the fact that in our model, a negative utility implies disliking by the user while a positive value implies liking.

In our model, we apply the Equal Weight Policy [24] to compute the value of each w_j, and use multiple regression to compute the weights w'_i. The EWP consists of

assigning the same value ($w_j=1/n_i$) to each weight, while making sure that $\sum w_j=1$. We have preferred this strategy as it has a very high relative accuracy compared to more complex one [24], while requiring no cognitive effort from the user. The computation of the w_i' is explained in more details in section 3.4.2.

Finally, HAPPL estimates the missing utility value of concept c by propagating the average weighted utility value of the closest concepts on which we have preferences, {CCP}. Formally, the estimated utility value of a concept c is computed as follows:

$$utility(c) = \frac{\sum_{i=0}^{|\{CCP\}|} \left(utility(CCP_i) \times asim(CCP_i, c)\right)}{\sum_{i=0}^{|\{CCP\}|} asim(CCP_i, c)} \quad (8)$$

where CCP_i is the i^{th} closest concept on which we have a utility value, and asim is the similarity metric of the concept tree from which the concept c is instanced . The size of the CCP set plays a crucial role, and must be build in order to represent the domain we are modeling. Obviously and in general, the bigger the set, the more accurate will be the utility estimation. By definition of our problem, only a few values can be elicited from the user, which means that a tradeoff as to be done.

3.3 HAPPL Process

The HAPPL algorithm is a four step iterative process as shown in Figure 5.

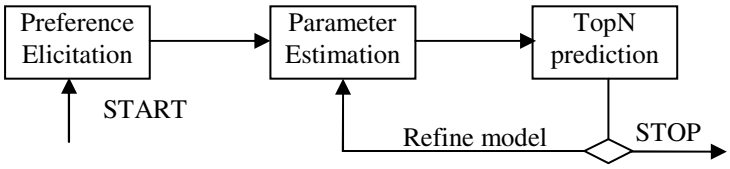

Fig. 5. Illustration of the HAPPL Process

3.3.1 Preference Elicitation

The preference elicitation process is designed to be very simple and similar to collaborative filtering. Each user is asked to define a learning set (LS) composed of at least m outcomes with their associated ratings. The number m corresponds to the number of compound attributes, whilst the rating is usually an integer ranging from one to five.

3.3.2 Parameter Estimation

Once the elicitation is completed, we apply a simple algorithm, *utilityEstimation*, on the learning set in order to estimate as many utility values as possible.

The algorithm works as follows. First we convert the rating of each movie in LS into a score, ranging in the interval [-1, 1].

```
Algorithm utilityEstimation(LS, λ)return CCP:
  CCP ← {};
  for x ∈ LS loop
    score ← rating2score(x);
    for d ∈ x loop
      λ.f(d).utility ← λ.f(d).utility + score;
      λ.f(d).isCCP ← true;
  for c ∈ λ loop
    if λ.f(d).isCCP then
        λ.c.utility ← λ.c.utility / |LS|;
        CCP ← CCP ∪ λ.c;
  return CCP;
```

Then, and for each features of the movie x, d ∈ x, we add the score of the movie to the utility of the representative concept in the ontology λ, λ.f(d).utility + score. Remember that the representative concept of a domain value d is extracted from λ by using the function f defined in section 3.2. Once all the movies in the learning set have been processed, and for all the concepts with a utility value, we normalize their utility values, λ.c.utility / |LS|, and add the concept to the set CCP. Notice that the algorithm assumes the independence of the attributes, and does not look at any combination what so ever. Note that our model is built on the additive utility model that makes such hypothesis.

Unfortunately, the algorithm utilityEstimation is very unlikely to have computed the utility of each possible domain value. Consequently, we use another algorithm called *utilityPropagtion* that will estimate the missing utilities by propagating the utility values already known. This propagation is achieved by calling the function *estimate MissingUtility* that will take the set of known values, CCP, and apply equation (8).

```
Algorithm utilityPropagation(λ, CCP):
  for c ∈ λ ∧ c ∉ CCP loop
    λ.c.utility ← estimateMissingUtility(c,CCP);
```

Once all the sub-utility functions have been computed, we have to estimate the weights of the compound attributes. After that, the utiliyPropagation algorithm has been applied, we are left with a system with at least m equations, m unknowns, and m grades. Thus, we use classical multiple regression (MR) to estimate the weights of the compound attributes. Finally, the weights are normalized to satisfy to the properties defined in section 3.2.

3.3.3 Top-N Selection

Once all the user parameters have been estimated, we can compute the utility of each outcome by applying equation (7). Then, we rank the outcomes in decreasing order of the computed utility and select the first N one; this is called the *top-N items recommendation* strategy where N is a parameter set by the system (usually ∈ [3, 10]).

3.3.4 Refinement Process

Finally, the user has the opportunity to add or remove outcomes from the learning set in order to refine his model. Furthermore, he can directly modify the utility value or weights assigned to each attribute. This allows building a dynamic user model using both implicit and explicit preferences.

3.3.5 Complexity Analysis

To simplify the complexity analysis, we will suppose that our model has m compound attributes, each consisting of r attributes and with n attributes in total. Among those r attributes, we have q attributes with known utility functions, and thus r-q without. The worst case complexity for the different component of our model is as follows:

1. *Elicitation Process*: ask m questions to the user. Thus, the complexity is O(m).
2. *Utility Estimation*: looks at the attributes of the m outcomes in LS. In the worst case scenario, all the outcomes have values on all the attributes which implies a complexity of O(mn).
3. *Utility Propagation*: estimate the r-q missing values of each m compound attributes by looking at the similarities with its neighbors. If we assume that each concept in the ontology knows its depth, and knowing that $n \geq r$, then our similarity function (6) will have a complexity of O(n). Thus, to estimate the missing values, equation (8) will look at the q neighbors. Thus, the complexity is O(qn).
4. *Weight Estimation*: Villard has shown in [22] that the complexity for solving a linear system of m equations with m unknown is $O(m^w)$, with w~2.38.
5. *Top-N Selection*: must compute the utility of all p items and then select the N best ones. The utility computation of an item can requires up to n operations. Thus, the top-N estimation requires $p \times n + p\log p$ operations, if the MergeSort algorithm is being used. In most problems $n > \log p$, which implies a complexity O(pn).

In today's system with millions of outcomes with tens of attributes, this implies that $p >> n^{2.38}$. Thus, we can estimate the overall complexity of the HAPPL as O(pn).

4 Experimental Results

In this section, we explain the experimental methodology and the metric used to validate the hypothesis of our model. We ran our model on the MovieLens[2] data set. We used this data set as it is widely being used through out the research community [13][18][19], and it contains the data requires to solve our example given in the introduction. MovieLens is a data set containing the rating of 943 users on at least 20 movies. There are 1682 movies in the database described by 19 themes (drama, action, and so forth). To increase the description of the movies, we wrote a wrapper that extracted the year, MPPA rating, duration, actors and directors (due to the high sparsity of the actors and directors, it was decided to ignore those attributes in our experiments) from the IMDB[3] website, given the movie's URL.

[2] http://www.cs.umm.edu/Research/GroupLens/data/ml-data.zip
[3] http://www.imdb.com

Unfortunately and to the best of our knowledge, there is no ontology modeling the movie domain. Thus, we created the concept trees from scratch based on our common sense, and from definitions found in various dictionaries.

4.1 Overall Performance Analysis

To benchmark our approach, we tested the accuracy of the HAPPL technique against existing recommendation techniques, and studied how many ratings were required by the parameters estimation algorithm in order to obtain an accurate model.

The experiment was as follows. First, users with less than 115 ratings were removed from the data set and for each remaining user, 15 ratings were inserted into a test set, while the rest was inserted into a temporary set. From the temporary set, 11 learning sets of varying size were created in order to test the accuracy of the model. The size of the sets varied incrementally from 4 to 100 ratings, and with learning_set$_{i+1}$ = learning_set$_i$ ∩ ratings_from_temporary_set. Various techniques were used to estimate the weights and the utility functions from the learning set: the model defined in the previous section with a ρ set to 0.75 – HAPPL, random utility values for all the missing utility value but compound attributes' weights estimated by multiple regression– *randFunc*, and random compound attributes' weights but with the utility values estimated by the ontology – randWeights. HAPPL was benchmarked on the users' test set against two popular strategies:

- random policy (*RAND*) which assigns a random rating to each movie in the test set.
- the adjusted cosine collaborative filtering with 90 neighbors (CF). We did not implement the CF algorithm; instead, we used the freely available MultiLens[4] package with the filter ZScore which will allow us to perform adjusted cosine similarity. We set the neighbors to 90 as authors [13][18] have shown that the optimal for the MovieLens data set is very close to this value.

Collaborative Filtering algorithm was chosen as benchmark over classical content based filtering as it is known that it is today's best performing filtering, and the most widely used recommendation system. Moreover, experimental results by Melvielle et al [12] have shown that CF performs better than pure content based filtering in the movie domain. Furthermore, content-based approach requires many ratings to train the user mode, which is not available in our situation.

The top-5 policy was used to select the best movies to show to the user. For each strategy, all the movies in the test sets are sorted based on their predicted ratings, and the first five movies are returned as the recommended items.. In order to compare HAPPL with existing work, the accuracy of the recommendations was measured using the Mean Absolute Error (MAE), which measures the mean average deviation between the predicted rating and the user's true ratings. Herlocker et al. in [9] have argued that MAE may be less appropriate measure when the granularity of the preference is small. However, the problem we are solving has very sparse data, which leads to significant differences between the predicted ratings and the real user's ratings. On the other hand, it was argued in [8] that the MAE has two main advantages:

[4] http://knuth.luther.edu/~bmiller/multilens.html

simple to understand and has well studied statistical properties when comparing two approaches.

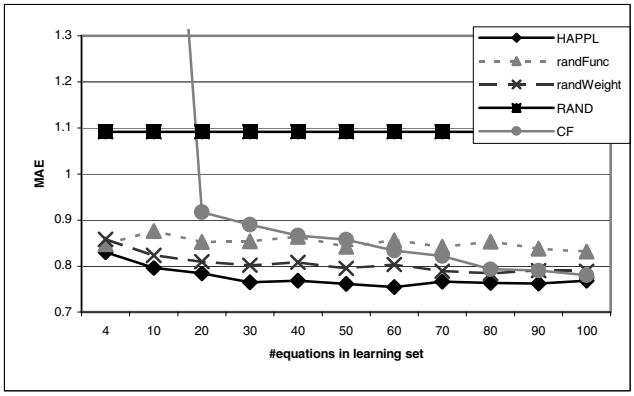

Fig. 6. Accuracy measure of various recommendation techniques bases on the number of known ratings in the learning set

The experiment was run 5 times, and the averaged results are illustrated in Fig. 6. The graph shows that HAPPL can be used to build robust recommendation systems as it has the lowest MAE, whatever the size of the learning set. Our approach is sensitive on the quality of the preference value of the CCP set, which is shown by an in increase in accuracy with increasing size of the learning set as However, after 60 ratings, the accuracy slightly decreases, which is due to the over fitting of the user's true preference value As expected, collaborative filtering performs poorly when the number of ratings in the learning set is very low. It rapidly improves until it reaches the 20 ratings threshold, and then the improvement slows down. This is a well known effect that is due to the high data sparsity (sparsity>0.99), it also reflects the behavior of the systems when new items are added and need to be evaluated. These results clearly show that CF is unsuitable for volatile environments and low involvement users. On the other hand, the HAPPL approach performs well even with high data sparsity, with a mean average error of up to 76.4% lower. The student test reinforces this statement with a t-obs=-39.99 and p -value<< 0.01. The random policy was also plotted to see if our HAPPL approach does actually improve something; and with an MAE of ~1.09, it is clear that it does.

An interesting aspect to consider is whether or not our utility propagation and estimation algorithm is better than just using random values? The graph clearly illustrates that HAPPL performs better than random values (randFunc) and statistical analysis showed that the improvement is significant when we had at least 10 ratings in the learning set (t-test: t-obs= 2.99, p-value=0.0016). This behavior makes sense and implies that a minimum of knowledge on some of the values must be known in order to estimate the rest of the utility values. Concerning the utility of the multiple regression to estimate the weights, we cannot assert anything as we obtained a p-value~0.1. However, the graph shows that HAPPL performs slightly better than randWeights,

whatever the size of the learning set. Finally, it is worth pointing out that with only 4 ratings in the learning set, the HAPPL model is able to perform reasonable well. The optimal accuracy is obtained with 60 ratings in the learning set, which is 10% better that CF. Moreover, this experiment shows that HAPPL is quite robust as the accuracy hardly changes with varying size of the learning set.

4.2 Ontology Influence

Finally, we tested the influence of the ontology on the recommendation with the HAPPL technique defined in the previous section. For this experiment, HAPPL used the ontology made up from our common sense and from definitions found in various dictionaries (*DictOntology*). We also created a second ontology with a topology similar to the previous one, but rearranged randomly the concepts (*RandOntology*).

We ran the experiment 5 times on users who had at least 65 ratings. From each user's ratings, 10 were used in the learning set, and 15 others were inserted in the test. From those 10 ratings, we extracted using our utility estimation algorithm the preference on a set of 5 representative attributes (RCset) and estimated the other utilities using HAPPL with: the optimized ontology (*DictOntology)*, the random ontology (*RandOntology*), and randomly generated values as utilities (*RandFunctions*).

Table 1. Accuracy measure of the HAPPL based on various apprach to estimate the utilities

	DictOntology	RandOntology	RandFunction
MAE	0.809	0.873	0.887

Table 1 shows the results obtained when applying the top-5 policy. As expected, DictOntology performed much better than RandOntology. As a matter of fact, the improvement is statistically very significant with an average t_{obs}=-2.6 and p-value<0.005. Furthermore, the HAPPL with the random ontology performed nearly as bad as when the utility functions where randomly generated.

These results tend to prove two things. First, the ontology plays an important role in predicting the user's preference. Second, the choice of the ontology is crucial as a bad one will perform badly. As mentioned in section 3, this paper does not focus on the construction of the ontology, but the results shows that our model could be used to evaluate the quality of the ontology, with respect to the user's true ratings.

4.3 Discussion

Many researchers have tried to boost the accuracy of Collaborative Filtering. Melvielle et al. [12] is probably the most famous work in this domain. They use a content-based approach to try to fill up the CF's matrix in order to avoid the sparsity problem and new-item problem. They performed experiments on the EachMovie data set, and obtained an improvement of 4% over classical CF when they have 25% of the ratings in the test set. With this configuration, HAPPL generates an improvement of over 10%. However, MovieLens is a subset of the EachMovie data so we cannot draw any clear conclusion. Mobasher et al. in [13] improved CF by adding semantic context to the problem. They extracted the attribute values of each movie and from it created a

semantic matrix. This matrix is exploited in order to compute the similarity between two movies. Finally, they predicted the rating using a weighted combination of the semantic matrix result and CF. Their method shows very good performance, with up to 22% improvement on the MAE metric when data sparsity is high. The best improvement is achieved when the ratio of the learning / test set is around 0.3. At very low ratio, their improvement is around 15%, which is still 60% worse than our HAPPL technique.

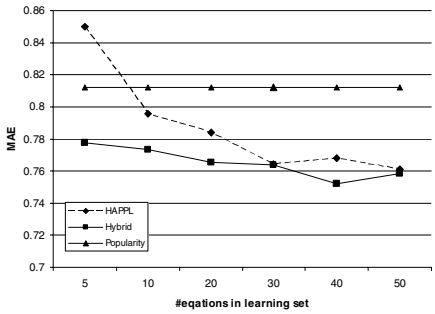

Fig. 7. Accuracy of HAPPL, Hybrid, and Popularity techniques

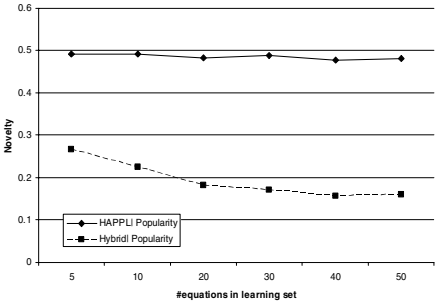

Fig. 8. Novelty of HAPPL and Hybrid compared to the Popularity technique

Those results tend to show that we could improve our HAPPL technique by combining it with CF. By doing that, we could improve the accuracy but we would then violate the user's privacy, make it vulnerable to schilling attacks, and decrease the novelty. We tested this observation on users who had at least 65 ratings and implemented three approaches:

- *HAPPL*: our model defined in section 4.1
- *Popularity*: select the most popular movies (learnt over users who had <65 ratings).
- *Hybrid*: that combines HAPPL and Popularity by averaging the predicted grades.

First, it is amazing to see (Fig. 7) that a simple popularity approach has a MAE of nearly 0.81, which performs even better than our model when we have just 5 ratings in the learning set. However, HAPPL quickly outperforms the popularity approach when the number of equations in the learning set increases. Notice that the popularity approach is a very basic collaborative strategy as it uses other people ratings to predict a movie's rating. One major difference with CF is the fact that it does not use the actual user's data, which explains why it performs so much better than CF when there is few ratings in the learning set. Our hybrid approach that combines the two previous techniques clearly improves the recommendation, especially when the number of learning equations is less than 30. This can be explained by the inaccuracy of our utility estimation algorithm when high data sparsity occurs.

Finally, we tested the novelty of the HAPPL technique and the hybrid approach against the popularity approaches, and display the result in Fig. 8. In this paper, we define the novelty between two approaches a and b as the number of correct predictions in the recommendations made by algorithm a that is not present in b over the

total number of correct prediction made by algorithm a. The results are very interesting and show that our method has about 50% novelty in the prediction over classical popularity strategy, which tends to prove that HAPPL recommends more personalized results. Furthermore, and as with the MAE, HAPPL's novelty remains constant whatever the size of the learning set. This tends to show that our method is robust, and that the propagation method works well even with very few data.

As expected, the hybrid approach has low novelty score, around 20%, but higher accuracy than all the other approaches. This confirms our hypothesis that by trading off the privacy and novelty, we can improve overall recommendation accuracy.

5 Conclusion

As shown in our experiment, most of the existing recommendation techniques include insufficient inductive bias to obtain an accurate user model from the amount of data that is typically available. We have shown how ontologies can provide the right inductive bias so that accurate recommendations are possible even with very little data about the user's preferences. On experiments using the MovieLens data, they consistently outperform collaborative filtering even when very little data is available.

Note that our HAPPL technique does not use any information about other users' preferences as in collaborative filtering. Instead, knowledge of what is common behavior is brought in through the ontology. The next step is now to apply ontology learning techniques to automatically construct reasonable ontologies for a given attribute model of the items. This should allow to better fine-tune the ontology to actual user's preferences, and achieve further significant performance gains. We also consider using a collaborative approach, as well as a learning technique, to better estimate the generalization coefficient rather than using a fixed value. Finally, we would like to perform live experiments to test whether or not the refinement process allows the user to build more dynamic user model.

References

1. Andreasen, T., Bulskov, H., Knappe, R.: From Ontology over Similarity to Query Evaluation. 2nd Int. Conf. ODBASE'03, Italy (2003)
2. Blythe, J.: Visual Exploration and Incremental Utility Elicitation. 18th national conference on Artificial Intelligence, Canada (2002) 526 - 532
3. Bradley, K., Rafter, R., Smyth, B.: Cased-Based User Profiling for Content Personalization. Int. Conf. on AH'00, (2000)
4. Claypool, M., Gokhale, A., Miranda, T.: Combining Content-Based and Collaborative Filters in an Online Newspaper. ACM SIGIR Workshop on Recommender Systems (1999)
5. Debreu, G.: Topological Methods in Cardinal Utility theory. Mathematical Methods in the Social Siences, Standford University Press, California (1960).
6. Fisher, D.H.: Knowledge Acquisition Via Incremental Conceptual Clustering. Machine Learning 2, (1987) 139 – 172.
7. Ha, V., Haddawys, P.: Problem-focused incremental elicitation of multi-attribute utility model. 13th Conf. UAI'97, (1997) 215 - 222

8. Ha, V., Haddawys, P.: A Hybrid Approach to Reasoning with Partially Elicited Preference Models. 15th Conf. Conf. UAI'99, (1999) 263 - 270
9. Herlocker, J.L., Konstan, J.A., Terven, L.G., Riedl, J.T.: Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems, Vol. 22, Issue 1, (2004) 5-53
10. Keeney, R., Raiffa, H.: Decisions with Multiple Objectives: Preference and Value Trade-offs. Cambridge University Press (1993)
11. Linden, G., Smith, B., York, J.: Amazon.com Item-to-Item Collaborative Filtering. IEEE Internet Computing (2003)
12. Melville, P., Mooney, R.J., Nagarajan, R.: Content-Boosted Collaborative Filtering. ACM SIGIR Workshop on Recommender Systems, (2001)
13. Mobasher, B., Jin, X., Zhou, Y.: Semantically Enhanced Collaborative Filtering on the Web. EWMF04, LNAI Vol. 3209, Springer, (2004)
14. Nasraoui O., Frigui H., Joshi A., and Krishnapuram R., Mining Web Access Logs Using Relational Competitive Fuzzy Clustering. Proc. of the Eighth Int. Fuzzy Systems Association Congress, Hsinchu, Taiwan (1999)
15. Nasraoui O., Frigui H., Krishnapuram R., Joshi A.: Extracting Web User Profiles Using Relational Competitive Fuzzy Clustering. International Journal on Artificial Intelligence Tools, Vol. 9, No. 4, (2000) 509 – 526
16. Resnik, P.: Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. J. of Artificial Intelligence Research (1999) 95 - 130
17. Salzberg, S. L.: On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. Data Mining and Knowledge Discovery, Vol. 1, Issue 3, (1997) 317 - 327
18. Sarwar, B., Karypis, G., Konstan, J. , Riedl, J.: Analysis of Recommendation Algorithms for E-Commerce. ACM Conf. EC'00, (2000)
19. Schein, A.L., Popesucl, A., Ungar, L.H., Pennock, D.M.: Methods and Metrics for Cold-Start Recommendations. 25th Int. ACM SIGIR'02 (2002)
20. Stolze, M: Soft navigation in electronic product catalogs. Int. J. on Digit. Libr., Vol. 3, Issue 1 (2000)
21. Sullivan, D.O., Smyth, B., Wilson, D.C., McDonald, K., Smeaton, A.: Improving the Quality of the Personalized Electronic Program Guide. User Modeling and User-Adapted Interaction, Vol. 14, Issue 1 (2004)
22. Villard, G.: Computation of the Inverse and Determinant of a Matrix. Algorithms Seminar INRIA (2003) 29 - 32.
23. Von Neumann, J., Morgenstern, O.: The Theory of Games and Economic Behavior. Princeton University Press, Princeton (1944)
24. Yang, J., Wenyin, L., Zhang, H., Zhuang, Y.: Thesaurus-Aided Approach For Image Browsing and Retrieval. IEEE Int. Conf. on Multimedia and Expo (2001)
25. Zhang, J., Pu, P.: Effort and Accuracy Analysis of Choice Strategies for Electronic Product Catalogs. ACM Sym. Applied Computing, (2005) 808- 814