Abstract

SREENATH, RAGHURAM MASTI. A Community-Based Rating System for Selecting Among Web Services. (Under the direction of Munindar P. Singh.)

The current infrastructure for Web services has a static approach to discover a service. It is based on a common repository that has a simple search interface that lets the user query and find a provider for the desired service. More often than not, the repository produces a long list of service providers along with typical interfaces to talk to them. There is no support to evaluate service providers. Such evaluations are important to make a selection among competing service providers. This thesis develops a community-based approach for evaluating service providers. In this approach, agents cooperate with each other to evaluate different providers. Importantly, the agents rate each other, to decide how to weigh each other's recommendations. The reasoning of each agent is enabled by a concept lattice, which supports a mechanism to rate the agents.

A COMMUNITY-BASED RATING SYSTEM FOR SELECTING AMONG WEB SERVICES

BY
RAGHURAM SREENATH
DEPARTMENT OF COMPUTER SCIENCE
NORTH CAROLINA STATE UNIVERSITY
RALEIGH, NC 27695

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH
DECEMBER 2002

APPROVED BY:

PETER R. WURMAN

MICHAEL R. YOUNG

MUNINDAR P. SINGH
CHAIR OF ADVISORY COMMITTEE

To my mother

Biography

Raghuram M. Sreenath was born on May 5th, 1978 in Bangalore, India. He lived in Bangalore until August, 1999. He got his Bachelor of Engineering degree in Computer Science and Engineering from University Visvesvaraya College of Engineering, Banglore in 1999. He was a masters student in North Carolina State University in the Department of Computer Science from August 2000 to December 2002.

Acknowledgments

I thank my advisor, Dr. Munindar Singh, for his constant support and inspiration. I thank my committee members, Dr. Peter Wurman and Dr. Michael Young. I thank Ashok, Vinay, Raj, Dharshan, and Srikanth for proof reading my work. A special thanks to Pınar for convincing me to switch to the Linux operating system. I would also like to thank her for all the interesting discussions we had. They bear a huge influence on the quality of this work. I thank Yathi and Amit for letting me have priority over the lab resources. I thank Subhayu for lending me his laptop for my presentation.

This research was supported by the National Science Foundation under grant ITR-0081742.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer applications executing on distributed systems have become popular over the past decade. Distributed system continue to grow in popularity due to the increasing need to develop reusable components that can coordinate with each other . There is a definite trend among software vendors to expose these reusable components as independent modules, referred to as *services*. The public interface that these services expose is usually published in a registry for the world to discover them. The more challenging aspects of choosing among competing services is either left as a problem for the user to solve or is handled in artificial ways such as issuing of certificates by *trusted* third parties, and advertising service attributes on popular portals.

1.1 Applying Web Services

Web services are usually defined as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. For example, a Web service may provide stock quotes or process credit card transactions. Once a Web service is deployed, applications (and other Web services) can discover and invoke the deployed service.

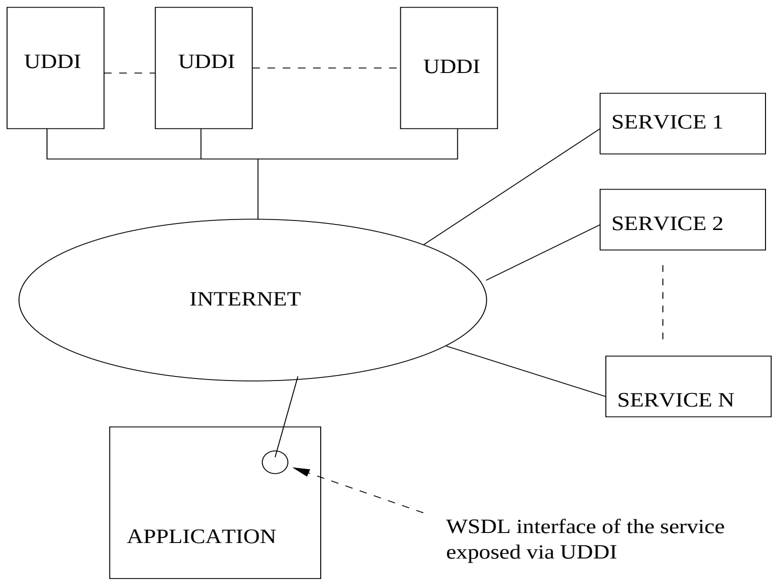


Figure 1.1: Current web services architecture

1.1.1 Description of Services

The *Web Service Description Language* is an industry standard used to describe services [WSDL, 2002]. It includes such information as:

1. Interface description, i.e., the methods exposed.

2. Port description, i.e., its url.

3. Fault types, i.e., the messages returned when the request is not understood or when an illegal request is made.

1.1.2 Discovery of Services

Universal Description, Discovery and Integration is an industry standard for discovering services [UDDI, 2002]. A database, called *Business Registry*, contains information about services that are registered with it. There are many such registries, which are distributed globally and synchronized periodically for consistency of data. These registries expose a search interface for the world to discover the registered services. The working of UDDI can be summarized as follows:

1. Software companies, standards bodies, and programmers volunteer to populate a business registry with descriptions of different types of services.

2. Businesses populate the registry with descriptions of the services they support.

3. Business registries assign unique ids to each service and business registration.

4. Marketplaces, search engines, and business applications query a registry to discover services at other companies.

5. Businesses use these data to facilitate easier integration with each other over the Web.

1.2 Challenges

The current Web Services architecture, described in the Section 1.1, has an implicit assumption that the choice of a service provider is already made. Further, this architecture assumes that an appropriate service provider can easily be chosen. Choosing a particular service provider among a list of competing service providers is a task in which the amounts of *trust* the user places in various service providers becomes an important factor.

1.2.1 Current Approaches

A few definitions are in order before we discuss popular approaches.

Definition 1 A set of people who have similar interests in services form a *community*. ▮

Note that it is not necessary that everyone in the community know each other a-priori, as long as there is a mechanism for them to discover each other.

Definition 2 The set of all possible communities forms a *society*. ▮

Definition 3 *Consumer* is a role taken on by a member of a community when he requests a service. ▮

Definition 4 *Service Provider* is a role taken on by a member of a community when he provides or is willing to provide a service requested by a *consumer* ▮

We now discuss some of the popular approaches that have been used to find service providers:

Reputation

Online communities such as *Epinions.com* [2002] include a setup where the users rate each other and the services they use. An accumulation of such ratings given by many users to an individual of the community results in his *reputation* in the community. This reputation could be used by other members to decide the extent to which they would like to mimic his actions and select the same service as he does. Given the reputations of the users and their ratings of various services, the process of finding a good service could be automated. Reputation systems, e.g., Sporas [Zacharia et al., 1999], come up with one rating for a service provider for everybody in the community to use. This rating may not be optimal

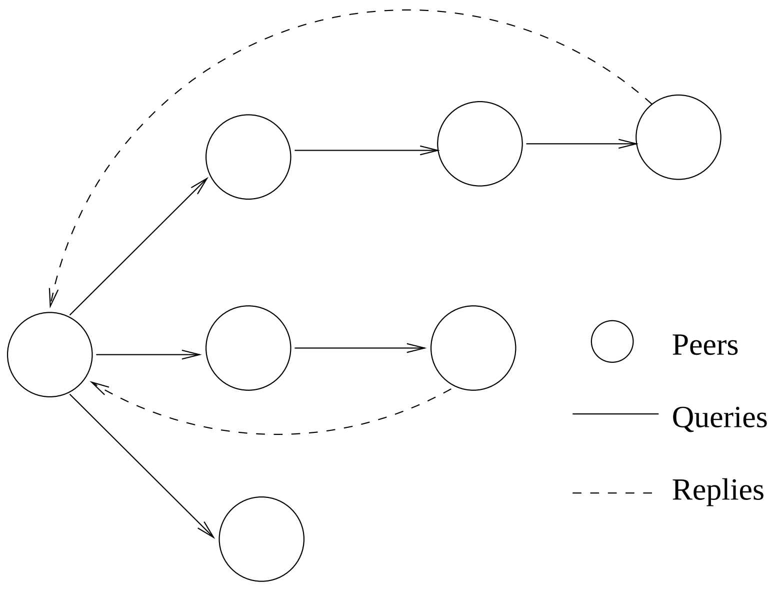


Figure 1.2: Propagation of queries in Gnutella

with respect to every user in the system. The extent to which an individual's interest is represented in these ratings is proportional to the *reputation* of the rater himself. Although reputation systems are better in locating service providers when compared to systems with no reputation such as Kasbah [Chavez and Maes, 1996], their drawback is that they assume that one rating of a service provider fits all. This may not always be justified, especially when the service providers have multiple attributes, each of which is preferred differently by different users.

P2P

The problem of a centralized database is overcome by Peer-to-Peer (P2P) systems. P2P systems are distributed systems with no central authority or management. Searching for a service provider reduces to querying *peers* for the kind of service required. Gnutella is one of the most successful implementation of a P2P system [2001]. It is used to share files over the Internet. The system assigns a set of neighbors to each user. When the user requests a file, a query is sent out to all of his neighbors, who in turn may forward the query to their neighbors, and so on until the requested file is found. The search process is depicted in Figure 1.2.1. The drawback of this approach is that there is no *trust* associated with the suggestions given by the peers. That is, there are no guarantees made on the quality of the file that was found.

Referral

A referral system is a form of P2P system. Referral systems address the problem of *trust* by associating a permanent identity with the peers. The peers in this case not only provide services directly, but may also refer to other peers. In this manner, they can find the right peer to provide the required service. Every peer is represented by a software agent in the referral system. These agents maintain a changing list of *trusted* peers of the system, which are referred to as *neighbors*. Whenever a service is requested, the agent *asks* its neighbors to help it locate the right service provider. The agent changes the ratings of its peers based on the quality of answers (replies or referrals) that they give.

1.2.2 Our Approach

Competing service providers potentially have an infinite number of attributes to advertise. It is highly likely that different users would be interested in different subsets of the possible attributes. Also, the weights that different users attach to each of the attributes would most likely be different. Let $\{a_1, a_2, ..., a_n\}$ be the true representation of scores given to individual attributes of a service provider j and let $\{w_{i1}, w_{i2}, ..., w_{in}\}$ be the weight attached by user i to the respective attributes. The local rating (R_{ij}) of the service provider by the user can be given as:

$$R_{ij} = \sum_{k=0}^{n} (w_{ik} \times a_{jk}) \tag{1.1}$$

Equation 1.1 suggests that the rating of a service provider must be personalized for individual users. Consider Example 1.

Example 1 Suppose there are three travel agents: A, B, and C. Also, suppose that the complete list of attributes that any travel agent can advertise is: cost, reliability, speed, customer-relationship, and efficiency. Now, let there be two users, U1 and U2, that are seeking a travel agent. It is highly likely that the preference of these two users are different for the different attributes listed. User U1 might feel cost is the most important factor and not bother about customer-relationship, while user U2 might value efficiency more and not bother about cost. If there were only one rating given for any travel agent, or if a rating of only a sub-set of the complete list of attributes were available for all travel agents, it would be difficult for the users to select among these travel agents. ▮

Personalizing the rating of a service provider for individual users would require the user to clearly indicate his preferences in terms of the weights he attaches to the attributes defining a service provider. Coming up with weights (w_i's) for individual attributes is not possible for most users. This could be because they lack enough experience to give a correct estimate. Moreover, it is the overall evaluation of the service provider that is more important than the evaluation of individual attributes. A community based rating system could be used to address this problem. We could attach weights (denoted by W in Equation 1.2) to the raters, so that a user can find his own true rating of the service provider, using the ratings given by other known users. If there are N agents in the community, this rating can be represented by the following equation:

$$R_{ij} \approx \sum_{k=0}^{N} (W_k \times R_{kj}) \tag{1.2}$$

We define a community in which the *consumers* do not offer any service but may have (either at present or in the near future) a need for some service, and the *providers* do not request any service but have services to offer presumably in exchange for some material benefits which, however, are not directly modeled. Consumers *query* each other to find out the Quality of Service (QoS) offered by a service provider. The query is in the form of a list of service providers that a user needs to choose among, and the reply is the same list with a *rating* attached to each of them. This rating could be *null*, indicating that the person replying doesn't know about the service provider. A *consumer*, when in need of a service, sends out *queries* to other consumers in the community. Upon receiving replies, he makes a local decision as to which service provider to select finally. In order to make this decision, the consumer should have a local rating of other consumers in the community.

1.2.3 Implementation

As discussed in Section 1.2.1, agent-based P2P systems could be used to rate agents in a community. Yolum and Singh show that by maintaining expertise (a vector) and sociability (a scalar) for each agent and by having a set of changing neighbors, we can have a schematic configuration of the society with desirable properties such as good consumer-provider proximity [2002]. The system that they define is described next.

Architecture

When an agent wants to ask a question, it sends the question to its neighbors. These neighbors are chosen from among its *acquaintances* based on a neighbor selection policy. At the other end, when an agent receives a query, it does one of the three things:

1. Reply with an answer.

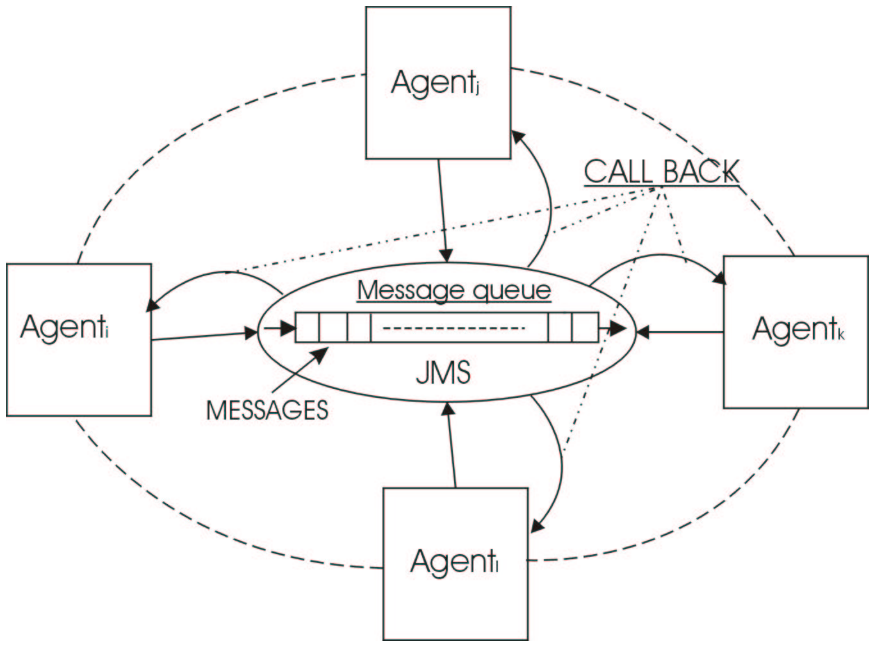


Figure 1.3: Agent based referral system

2. Not reply at all.

3. Reply with a referral (based on a local referral policy).

The agent which asks the query maintains a directed acyclic graph (DAG) capturing the communication between itself, and all the agents that the query was sent to. This avoids duplication of query messages and formation of referral loops.

Finally, when the agent receives a reply, it evaluates the reply and updates the expertise and sociability of the agent giving the reply based on this evaluation.

Extending the architecture for service appraisal

We make a few modifications to the approach described above and apply it to our society, where the peers are all consumers who use and rate service providers. The specific modification made to the architecture are listed below:

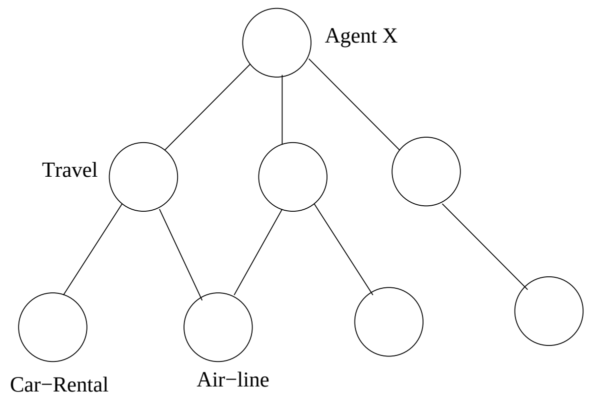


Figure 1.4: A sample hierarchy of categories

1. *Represent expertise as a graph rather than a vector*

   Modeling expertise of an agent as a vector has the inherent danger of the number of entries in this vector growing too long. Moreover, a particular query may not always find a perfectly matching area of expertise. Conversely, an area of expertise might in itself not be definable independently i.e., there could be a hierarchical relationship between areas of expertise. For example, an expert in vehicles has to be an expert in automobiles and bikes among others. In order to solve this problem, we use a hierarchical structure (a graph) like the one shown in Figure 1.4 to maintain expertise of an agent. An area of expertise, which translates to a node in this graph, will henceforth be called *category*. Also, we refer to expertise as *rating*.

2. *Use confidence to represent trust in rating*

   A scalar value, *confidence*, is associated with every rating, similar to the one discussed in [Chen and Singh, 2001]. This confidence changes with each question answered by the agent. A new agent that has declared itself with an advertisement, sends its agent model (its rating of itself) to the other agents in the community. Each agent associates a confidence level with ratings in the model, which changes with the number of questions answered by the agent. Thus, confidence acts a measure of the level of *trust* an agent associates with the other agent. By taking *confidence* into account in calculating an agent's rating, we have addressed the issue of *trust* to a certain extent.

3. *Have a non-referral system*

   Since we are using the referral system only to appraise service provider rating and not to find service providers themselves, having neighbors give referrals would not be very meaningful. So, we modify the system so that the agents give only answers and no referrals. We now have a non-referral system with the trust issue taken care of (from the previous extension). So, we do not maintain sociability values for agents.

4. *Use a concept-lattice representation to help rate raters*

   Correlation measurements adopted in collaborative filtering techniques help compare two users based on the common set of items that they have rated. Collaborative filtering does not allow indirect evaluations of raters. Such indirect evaluations are important since a friend of a friend could also be a friend. In order to facilitate such measurements, we have a concept-lattice representation of the users and the scores they give to service providers. This approach will be described in detail in Chapter 3.

1.3 Organization

The rest of the thesis is organized as follows: Chapter 2 discusses the methodology we develop for selecting service providers. Chapter 3 provides a detailed description of the most important step in our approach – rating the raters. Chapter 4 describes the experimental setup that we adopt to evaluate our approach. Chapter 5 explores relevant literature and outlines future directions.

Chapter 2

Evaluation Model

In our experiments, we iterate through a list of tasks. Each task requires an agent to find a service provider in a particular category. We refer to the agent that is trying to find a service provider as the *active-agent*. We adopt a three-step approach to solve the problem of selecting a service provider. The active-agent goes through these three steps in every iteration. At the end of each iteration, the agent gets to evaluate a new service provider and based on this experience, correct the weights that it associates with other agents in the community. Unless otherwise specified, I represents the set of agents (users) in the community, and J represents the set of service providers. The word *user* is sometimes used to refer to an agent when the emphasis on the behavior of the human, which the agent represents, becomes important. Similarly, the word *rater* is used when the act of rating of the agent is to be emphasized. Also, the ratings given by the agents to individual service providers are referred to as *scores* to distinguish them from the *ratings* the agents attach to each other.

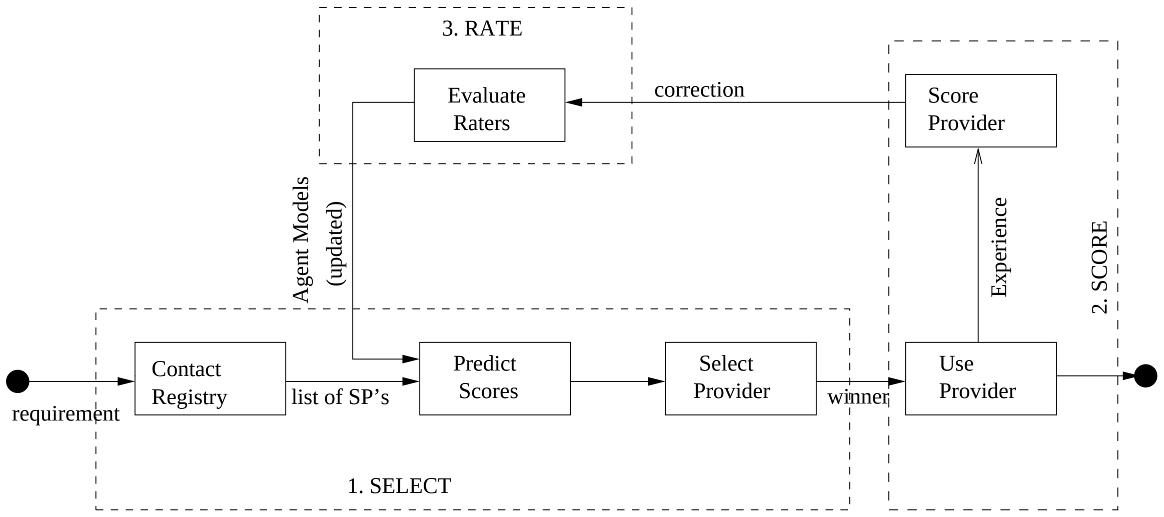


Figure 2.1: Flow diagram

2.1 Evaluation Cycle

An agent goes through the following three steps every time it wishes to use a service offered by competing service providers.

1. Select a service provider
    (a) Get a list of providers for the kind of service desired.
    (b) Contact raters to obtain appraisals of providers in the list.
    (c) Obtain scores from raters.
    (d) Evaluate scores (using the current ratings given to the raters) and choose a service provider.
2. Score the selected service
    (a) Use the selected service.
    (b) Rate the service based on the experience.

3. Rate raters and adjust their ratings.

2.2 Selecting a Service

2.2.1 Getting a List of Competing Service Providers

Two approaches could be taken here:

1. A registry (such as UDDI) is queried for the kind of service desired and a list of competing service providers is built. This list is then sent out to all the peers, in a query message, for appraisal. The agent then collects replies and builds a representation as discussed in Section 2.2.2.

2. The type of service required is sent out to the peers and requesting them to return a list of service providers with scores attached to each of them. This approach would be useful when the requester doesn't care as to who provides the services, as in the case of buying a book.

In our experiments we use the first approach.

2.2.2 Representation

	a	b	c	d
N_1	0.3	0.5	0.1	–
N_2	0.2	0.4	0.1	0.5
N_3	-	0.3	0.2	0.1

Table 2.1: Agents and the services they have rated

The raters and their scores are stored in a two-dimensional matrix, with the columns representing the different service providers, and the rows representing the raters. Let N_1, N_2, and N_3 be three agents who have rated four services providers, a, b, c, and d. Not all services are rated by all agents, i.e., each agent would rate only a subset of these services. An example is shown in Table 2.1.

2.2.3 Choosing the Winner

One way of choosing the winner is by using a weighted average of the scores (weighted by the local rating of the agents giving the scores). So, the final score of a service j, which belongs to a category c, is given by

$$S_{aj} = \frac{\sum_{i=1}^{n} (R_{ai} \times S_{ij})}{\sum_{i=1}^{n} R_{ai}} \tag{2.1}$$

Where R_{ai} is the local rating of agent a_i in category c, S_{ij} is the score given by this agent to service j, n is the number of agents evaluating the service and $\{a_1, a_2 ... a_n\}$ are the agent that have evaluated the service j.

The problem with Equation 2.1 is that different users have different score-ranges that they prefer while scoring, although the possible range of scores they can give remains the same. For example, a user could give a 4.0 out of 5.0 for the best service he has seen till now, while another might give a score of 5.0. Thus, a more accurate measure of the value that a user attaches to a service is the deviation of his score of the service from his average score. Hence, Equation 2.2 was used in our simulations in preference over Equation 2.1.

$$S_{aj} = \overline{S_a} + \frac{\sum_i \left(\left(S_{ij} - \overline{S_i} \right) \times R_{ai} \right)}{\sum_i R_{ai}} \tag{2.2}$$

2.3 Rating the Service

After using the service, the user can give the true score to the service provider as follows:

1. Individual attributes of the service are scored. These are the attributes that are important to the user, e.g., speed, accuracy, cost, and reliability.

2. The service is then given a score calculated as a weighted average of the individual attributes above. The score is then normalized to be in the interval [0,1] and represents the true score of the service for the user.

In our simulation, we fake the actual service evaluation, by maintaining a simple table of services and their attributes. Any agent that wants to evaluate a service is returned the record corresponding to the service. How each agent weighs each of the attributes, is up to the agent (perhaps driven by a local policy).

We now move on to evaluate the agents based on our current experience.

2.4 Evaluating the Raters

The local rating (R in Equation 2.1) itself would be a function of the rating (r) and the confidence (q) in the rating of the agent in a category c. The formula given in [Chen and Singh, 2001] is:

$$R = (r + 1)^q - 1 \tag{2.3}$$

Equation 2.3 has the following desirable properties:

1. It increases with r and q when the other variable is fixed.

2. It is 0 when either r or q is 0.

3. It is 1 only when both r and q are 1.

4. It is exactly same as r when q is equal to 1.

A hierarchy of categories is maintained like the one shown in Figure 1.4. Every agent models its peers based on this hierarchy. This is similar to the hierarchy described in [Chen and Singh, 2001]. Each node entry is a pair $[r, q]$ where r is the rating and q is the confidence in this rating, in the corresponding category in the category hierarchy.

Our approach to calculate r (rating) and q (confidence) in Equation 2.3 is discussed in detail in Chapter 3.

Chapter 3

Rating the Raters

Before we go on to discuss our approach to evaluate the agents, a few definitions and explanations are in order.

3.1 Formal Concept Analysis

Definition 5 A triple (G, M, I) is called a *Formal Context* if G and M are sets and $I \subseteq (G \times M)$ is a binary relation between G and M. We call the element of G *objects*, those of M *attributes*, and I the *incidence* of the context (G, M, I). ▮

Definition 6 For $A \subseteq G$ and $B \subseteq M$, if we define:

$$A' := \{m \in M \mid gIm \,\forall\, g \in A\}$$

$$B' := \{g \in G \mid gIm \,\forall\, m \in B\}$$

then, (A, B) is a *Formal Concept* of (G, M, I) iff: $A \subseteq G, B \subseteq M, A' = B$ and $B' = A$ ▮

Here, A is called the *extent* and B is called the *intent* of the concept.

Example 2 Consider the context of "Living Beings" where $G = \{Leach, Frog, Dog\}$, $M = \{a, b, c\}$ and I is as represented in Table 3.1.

	a	b	c
Leach	x	x	
Frog	x	x	x
Dog	x		x

Table 3.1: Context of "Living Beings". The attributes are a: needs water to live, b: lives in water, c: lives on land.

The concepts for this context are:

1. $(\{Leach, Frog, Dog\}, \{a\})$

2. $(\{Leach, Frog\}, \{a, b\})$

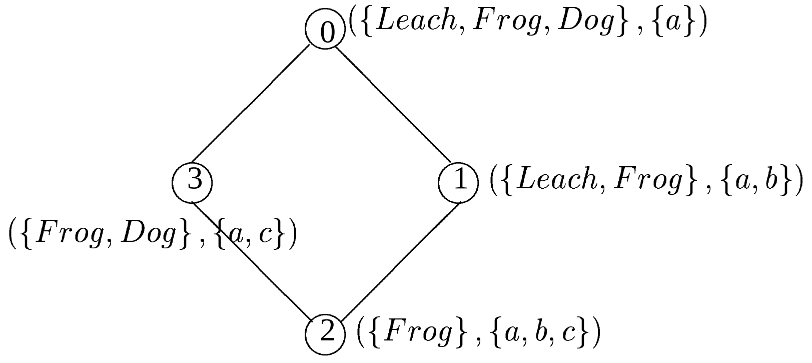
3. $(\{Frog\}, \{a, b, c\})$

4. $(\{Frog, Dog\}, \{a, c\})$

These concepts, when represented as a lattice, with "subset of objects" as the ordering relation, like in Figure 3.1, is called the "Concept Lattice" of the context.

∎

We could visualize the agents and the services they have rated as a context, and apply the principles of concept lattice to it. The agents would represent the objects and the service providers, the attributes. Such a context would represent a *has rated* relationship between the agents and the service providers. The concept lattice constructed by treating Table 2.1 as the context is shown in Figure 3.2. In this figure, the nodes are labeled with only the agents (*extent*). The service providers (*intent*) they have rated are dropped, since, in any concept, the *intent* and the *extent* can be derived from each other, given the context.



0 $(\{Leach, Frog, Dog\}, \{a\})$
3
$(\{Frog, Dog\}, \{a, c\})$
1 $(\{Leach, Frog\}, \{a, b\})$
2 $(\{Frog\}, \{a, b, c\})$

Figure 3.1: Concept lattice constructed from the context in Table 3.1

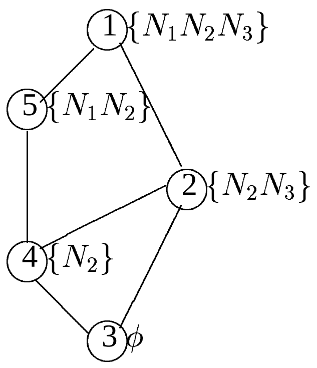


Figure 3.2: Concept lattice constructed from the context in Table 2.1

The following could be said about the scores given to a service provider in a concept:

1. The greater the height of the concept from which the score for a service provider was got, more the number of agents rating it, so more the confidence in the absolute score.

2. The lower the concept from which the score for a service provider is obtained, more the number of service providers being compared together (since the lattice formed by treating services as objects and agents as attributes would form a *Galois connection* with the lattice in Figure 3.2) and hence better the relative standing of the score.

3. The score from concepts whose height and depth are both low, i.e., concepts formed out of very few agents and very few service providers are undesirable, as nothing significant can be inferred from them. For example, if there were a service provider, e, which was evaluated by just one agent, say N_3, It would form a concept in itself and appear in the concept lattice of Figure 3.2 between concepts 1 and 3. Hence it would have a low depth and height (both equal to 1). In such a scenario, it would be difficult to rely on such a score. Hence such concepts could be considered for being pruned from the final evaluation lattice.

3.2 Motivation

Our motivation to do a concept-wise evaluation of the raters are as follows:

1. People seeing similar facets of the world should be evaluated together. For example, there is no point in comparing two agents, A and B, where A has evaluated services $\{a, b\}$ and B has evaluated services $\{c, d\}$.

2. Assume all agents are “learners”. If they have seen only a small number of services, they should not be penalized or rewarded unduly.

3. What we are evaluating is the agents learning capability, so evaluate agents treating their answer space (services evaluated), as the sample space their learners are exposed to.

3.3 Considerations While Evaluating

1. Direction of traversal through the concept lattice.

2. Is the evaluated service present in the concept? If so, the distance of each agents evaluation from the “correct” evaluation.

3. Are previously evaluated agents present in the *extent* of the concept being currently examined?

4. What should be done if the evaluated service is not present in the *intent* of the concept and no previously evaluated agents are present in the *extent*.

The remaining of this chapter deals with the issues raised here.

3.4 Traversing the Concept Lattice for Agent Evaluation

Since concepts are ordered such that a parent is always the super set of a child, we could quickly increase the number of raters that have been evaluated, by starting from a concept with the lowest depth, which contains the winning service (service that was rated). Next, we could handle the non-trivial (having a non-empty *intent* set) root concept of the sub-tree containing this concept. If, after this, all the raters have not been evaluated, we could consider all the children of the trivial root node, one at a time, till all the raters have been evaluated. The method is illustrated in Algorithm 1.

Algorithm 1 Re-Evaluate()

```
1: Do a breadth first search (BFS) to find a concept, say $C$, that contains the winning service.
2: Handle concept $C$ (i.e., re-evaluate all the agents in concept $C$).
3: Handle the non-trivial root concept of a sub-tree containing concept $C$.
4: If all the raters have not yet been re-evaluated, handle the children of the trivial root node, one at a time, till every agent has been re-evaluated.
```

Since all the agents would have been re-evaluated when the above algorithm executes, considering any other concept, that has not been handled, would be unnecessary and is ignored in our simulations.

Example 3 Consider the concept lattice of Figure 3.2. Suppose that the winning service was found only in concept $[N2]$. Note that this case is impossible considering the context from which this concept is derived: Table 2.1. But, we still consider this case, just to explain the algorithm above. If the above algorithm was run on the lattice:

Step 1: The concept $[N2]$ is found by doing a *breadth first search* to find a concept that contains the winning service.

Step 2: The Agent N2 is evaluated during the course of handling concept $[N2]$.

Step 3: Concept $[N1N2]$ is a non-trivial root node of a sub-tree containing $[N1]$, so it is handled next. This would result in agent N1 being included in the set of agents that have been evaluated.

Step 4: Since all the agents have not yet been evaluated (Agent N3 is still left out), the children of the trivial root concept ($[N1N2N3]$) will be considered one at a time. This would result in concept $[N2N3]$ being handled, or agent N3 being evaluated.

At this time, all the agents have been evaluated and hence the algorithm stops. ▌

3.5 Handling Concepts

The important step during evaluating raters (agents) is the way in which concepts are handled. The end-result is that the rating (r) and confidence (q) of the agents that are present in the concept (as its *extent*) are somehow altered to better represent the real world.

	1	2	3	r_{t-1}	q_{t-1}	r_t	q_t
N_1	x_1	y_1	z_1	r_1	q_1	?	?
N_2	x_2	y_2	z_2	r_2	q_2	?	?

Table 3.2: Handling Concepts

The scenario, for a concept with two raters (N_1, N_2) and three rated services ($1, 2$, and 3), is depicted in Table 3.2. Furthermore, in order to facilitate the calculation of r_t and q_t, we could divide this into three sub scenarios as follows:

3.5.1 Scenario 1: Early Stages

During the first few iterations, the active user would not have rated enough service providers to have a row wise comparison with other users. The scenario is depicted below:

	1	2	3	r_{t-1}	q_{t-1}	r_t	q_t
N_1	x_1	y_1	z_1	r_1	q_1	?	?
N_2	x_2	y_2	z_2	r_2	q_2	?	?
active-user	x_3	-	-	-	-	-	-

In this scenario, we have a concept, which contains a service provider (1) that has been evaluated by the active user (this evaluated service provider will be called e.s.p henceforth).

Updating the Ratings

The new rating (r_t) depends on following:

1. The current rating (r_{t-1}).
2. The absolute difference between the scores given to the e.s.p by the rater and the active user.

$$d = |s_{i'j} - s_{aj}| \tag{3.1}$$

We use the following equation to rate agents in this scenario:

$$r_t = r_{t-1} \pm \left(\frac{\beta}{1+d} \right) \tag{3.2}$$

The decision whether to increase or decrease the rating ($+$ or $-$) in Equation 3.2 is made by agreeing on a *threshold* value for the difference in rating (d).

Normalizing

The difference d in Equation 3.2 is normalized to be in the interval $(0, 1)$. In order to prevent agent ratings from growing very large, we also normalize the rating such that $\sum r_{t-1} = \sum r_t$. This also serves as a minor incentive for the agents to reply to queries, since the only way they can increase their rating is by competing with others (they can't just wait for other's ratings to go down).

Damping

We introduce the damping factor, β, in Equation 3.2 in order to damp the increase and decrease of the rating. This damping factor should reduce the incremental value ($1/(1+d)$ in Equation 3.2) at the extremities, i.e., the reduction of an already low rating, and the increase of an already high rating should be low. We use Equation 3.3, in which λ is a positive constant and r'_t is the minimum of r_t and $(1.0 - r_t)$.

$$\beta = \lambda r'_t \tag{3.3}$$

3.5.2 Scenario 2: Later Stages

This scenario is same as in Section 3.5.1 except that the active user has now rated enough service providers to be able to make a row-wise comparison with other agents. The considerations for this scenario is same as for the previous one, except that the vector similarity between the active user and the other user replaces the additive factor in Equation 3.2. There are many algorithms that calculate the correlation between vectors, the simplest among them being the *mean square difference* between the vectors: $\overline{(U_1 - U_2)^2}$. Breese *et al.* compare many such algorithms [1998]. The Pearson's r-correlation with a few extensions seems to have performed best. Accordingly, user i's rating of another user i' is given by:

$$r_{ii'} = \frac{\sum_j \left(s_{ij} - \overline{s_i}\right)\left(s_{i'j} - \overline{s_{i'}}\right)}{\sqrt{\sum_j \left(s_{ij} - \overline{s_i}\right)^2 \sum_j \left(s_{i'j} - \overline{s_{i'}}\right)^2}} \tag{3.4}$$

Where $\{j \in J\}$ is the set of all service providers, s_{ij} refers to the score that user i has given to the provider j, and $\overline{s_i}$ refers to the average rating of user i. Note that we don't have to make a decision whether to add or subtract the correlation factor, since $r_{ii'}$ is positive or negative depending on the degree of correlation. So, this value (positive or negative) is added or subtracted from the old rating depending on the sign of $r_{ii'}$. The ratings are normalized as in Scenario 1.

3.5.3 Scenario 3: Propagation of Evaluation Through Concept Chains

An agent (rater) that is considered for evaluation satisfies at least one of the following conditions:

1. Agent is in a concept that contains the winning service in its $intent$.
2. Agent is in a concept that has agents that have been evaluated by virtue of satisfying condition 1.
3. Agent is in a concept that has agents that have been evaluated by virtue of satisfying conditions 1 or 2 above. (Note that this is the generic case of condition 2.)
4. Agent is in a lonely concept that cannot be reached through any of the above ways. Such concepts are not handled (i.e., the agents belonging to such concepts are not evaluated).

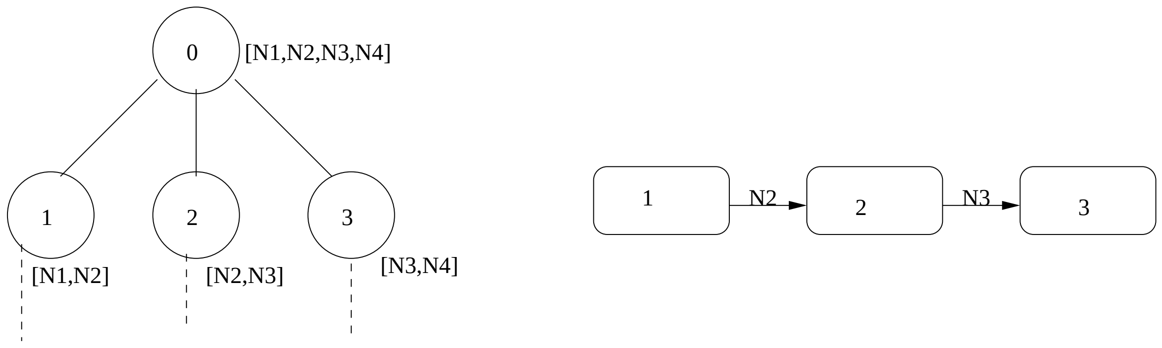


Figure 3.3: Propagation of rating through concepts

In order to facilitate indirect evaluation of raters (to find friends of friends), we have to find a way of propagating rating and confidence through concepts. Consider the concept lattice in Figure 3.3. In this lattice, if concept 1 contained the active agent (letting a direct evaluation of the agents in this concept), then the ratings (and confidence) would have to propagate to concept 2 through N2, and to concept 3 through N3, via concept 2. The propagation graph is shown in the same figure.

	1	2	3	r_{t-1}	q_{t-1}	r_t	q_t
N_1	x_1	y_1	z_1	r_1	q_1	?	?
N_2	x_2	y_2	z_2	r_2	q_2	?	?
N_3	x_3	y_3	z_2	r_3	q_3	$r_3^{'}$	$q_3^{'}$

In this scenario, we have a concept in which one of the raters has already been evaluated, perhaps in a previously handled concept. We have to somehow find a way of propagating this new rating and confidence to the other agents in this concept. One such scenario, with the propagating agent being $N3$, is shown in the table above.

Updating the Ratings

The new rating (r_t) depends on the following:

1. The current rating (r_{t-1}).
2. The vector distance of the rater's rating and the linking rater's rating ($d2$).
3. The vector distance between the active agent and linking agent ($d1$).
4. The distance between the previous and current rating of the linking rater (l).

A graphical representation of the situation is shown in Figure 3.5.3.

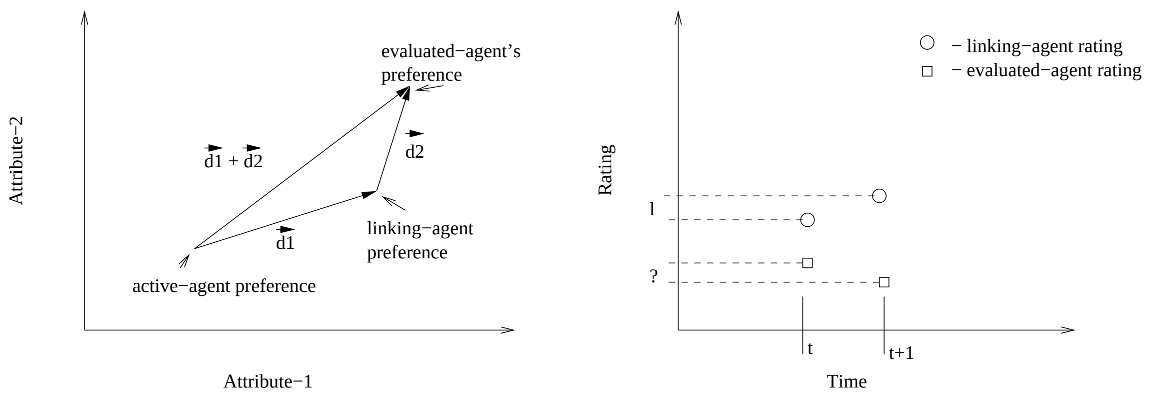


Figure 3.4: The problem of propagating ratings through concepts

The formula used to calculate the new rating is given in Equation 3.5. Here, β is a constant.

$$r_t = r_{t-1} + \beta \left(\frac{d1 + d2}{d1} \right) l \tag{3.5}$$

In Equation 3.5, β is the damping factor calculated as in Equation 3.3.

3.5.4 Updating the Confidence

We use a generic formula to update the confidence in rating (q in Equation 2.3) in all the scenarios discussed in Section 3.5. In general, we believe that the confidence in rating must go up after every evaluation of the agent, irrespective of whether or not the rating of the agent itself was increased. The case when confidence goes down (e.g., when an agent behaves erratically) could be a possibility in a real life scenario. But, we do not explore this research direction in our work.

The requirement for the value of confidence is summarized below:

1. The value of confidence is kept in the interval (0,1).

2. Confidence is initialized to a low value and is incremented every time there is a chance to evaluate the agent.

3. The amount by which the confidence is increased during every iteration depends on the degree of indirection while evaluating the agent's rating, i.e., the incremental value decreases with the propagation length of the ratings. For example, in Figure 3.3, the amount by which the confidence in the rating of N4 is increased should be less than the amount by which the confidence in the rating of N3 is increased, which in turn should be less than the amount by which the confidence in the rating of N2 is increased.

The equation used was:

$$q_t = \alpha \times \left(q_{t-1} + \frac{\delta}{(l+1)} \right) \tag{3.6}$$

In Equation 3.6, δ is the damping factor, which is set to be equal to the reciprocal of the sum of the confidences associated with every category of the agent being evaluated — $1/\sum_c q_{tc}$. α is a positive constant.

Chapter 4

Experiments

This chapter describes an experimental setup to test the key aspects of our proposal. Although implementing the entire architecture described in this thesis is not the goal, we have identified the key differentiators of our approach with respect to the literature. We describe a methodology to see these key differentiators in action and define a set of metrics with which to measure them.

4.1 Methodology

4.1.1 Data-sets Used

We evaluate our work with respect to two data-sets.

1. MovieLens database

   In order to be able to compare our work with other similar works, we use the well known MovieLens database [MovieLens, 2002]. This is a huge database with 100,000 ratings by 943 users. But the lack of a definition of user-preferences in terms of the attributes of the providers (movies), makes it difficult to protect the evaluation process from ambiguities inherent in human perceptions. Although the user's ratings are not completely random, we require a better prediction of the truth values (the actual value of the provider to a user) in order to evaluate our approach. However, evaluating our approach (and comparing it with other approaches) against this database, helps us visualize the performance of our approach under presumed varying interests of the users.

2. Artificial database

   Due to reasons cited above we also evaluate our work against an artificially created database. In this database we have three categories of services: *travel, car-rental,* and *airline*. The *travel* service forms the parent category for the other two services. Each service has five service providers. Each service provider is assigned different values for five arbitrarily chosen attributes: *cost, speed, accuracy, public-relations,* and *availability*. We define ten user profiles, wherein each profile contains weights associated with each of the five attributes.

4.1.2 Problem Formulation

Given

During every round of simulation, we maintain the following:

1. The *truth* matrix, M, which contains the real ratings of the Users vs. Items.

2. The *prediction* matrix, $\hat{M}$, which is a matrix of predictions made by our algorithm.

3. The *observation* matrix, O. This matrix in general satisfies $O_{ij} = M_{ij}$ for every pair (i, j) that has been observed, and $O_{ij} = null$ otherwise. Initially, O is partially filled and used to *bootstrap* the agent.

4. A list of *tasks*, T, that the active agent has to perform. Each entry in this list can be thought of as a six tuple, $<$*active-agent, requested-category, done, winning-provider, predicted-score, actual-score*$>$. The first two fields are filled in by the simulator and the last three fields are filled in by the agent, depending on whether or not it is able to complete the task (thus setting the *done* field to either *true* or *false*. The truth matrix, M, is used to help the agent fill the *actual-score* field.

During every iteration, a *task* is handed over to the agent. If the *task* is completed successfully (the *done* field is *true* in this case), the *predicted-score* is entered into the prediction matrix, $\hat{M}$, and the *actual-score* is entered into the observation matrix, O. The experiment terminates when all the *tasks* are completed or a predefined time-out is reached.

Observation

During the simulation, we observe the following:

1. Variation of errors with time
   Error value of the prediction, $\left(M_{ij} - \hat{M}_{ij}\right)_t$, at the end of or each iteration, t, is noted for $t = 1$ to n. Where n is the number of *tasks* given to the system.

2. Variation of agent ratings with time
   For every user, i, of the system, the active user's rating of this user, w_{ai}^t, at the end of each iteration, t, is noted for $t = 1$ to n. Where n is the number of *tasks* given to the system.

4.2 Comparison With Other Approaches

Our approach is compared with two other popular approaches. These approaches can predict a winning service provider and evaluate the raters, thus providing alternate means of carrying out the functionality depicted by the first and the last block of the flow diagram shown in Figure 2.1. These approaches are described briefly in the following sections:

4.2.1 Correlation-Based Approach

In this approach, a user's rating of any other user of the system is based on the correlation between the ratings of the two users. Pearson's correlation factor is used to measure the correlation between two users. The actual formula is given in Equation 3.4. The difference between the correlation based approach and the way Pearson's correlation is used in our approach is that in the former approach, the time of recording of the score is immaterial, i.e., a correlation is calculated based on all the scores given by the two users till now. In our approach, we use the correlation factor only for the current iteration. It is only used to find the similarity in scores given by any two users in the current iteration.

While making a prediction, a weighted average of the scores given by the agents (weighted by their correlation with the active user) is computed for each competing service provider. The provider with the highest score is declared as the winner.

4.2.2 Generalized-Learn-Relationship (G-Learn)-Based Approach

Nakamura and Abe have described the G-Learn approach in detail [Nakamura and Abe, 1998]. It is a generalized version of the *weighted majority algorithm* [Littlestone and Warmuth, 1994]. The basic idea here is that, instead of a binary vote (as in the case of the *weighted majority algorithm*), the users vote for all values within a permitted tolerance from their true value. Let A denote the range of values in M (the *truth* matrix). Also for any $a \in A$, let $V(a)$ denote the set of prediction values that are permissible when the correct value is a. A prediction is made as follows:

$$\hat{M}_{ij} = \begin{cases} \arg\max_{a \in A} \sum_{i' : O_{i'j} \in V(a)} w_{ii'} & \text{if } (\{i' : O_{i'j} \neq *\} \neq \phi) \\ C_0 (\text{a constant}) & \text{otherwise} \end{cases} \tag{4.1}$$

After knowing the true value (the second step in the three step approach described in Section 2.1), the weights are updated as follows:

$$w_{ii'} = \begin{cases} (2 - \gamma) w_{ii'} & \text{if } O_{i'j} \in V(M_{ij}) \\ \gamma w_{ii'} & \text{if } O_{i'j} \notin V(M_{ij}) \end{cases} \tag{4.2}$$

4.3 Results

4.3.1 Change of Agent Ratings With Time

The three approaches compared - Correlation, GLearn, and Concept - adopt different strategies to update agent ratings. We test each approach against the artificial database in order see the way the ratings vary with time. The artificial database was preferred to the MovieLens database, since we have an accurate representation of the true value of a service to the user (programmatically computed from the user preferences) as opposed to the approximate true value in the case of MovieLens (since the ratings are given by real users). The *active-agent* field in all the tasks is kept the same, since we are only interested in one active user's rating of other raters in the community. The observation matrix for this user is set to *null* indicating that the user begins with no idea of other users and the rating he has to associate with them. Thus, he starts with a default rating of all users in the community (which depends on the approach chosen). The way in which the ratings change in each of

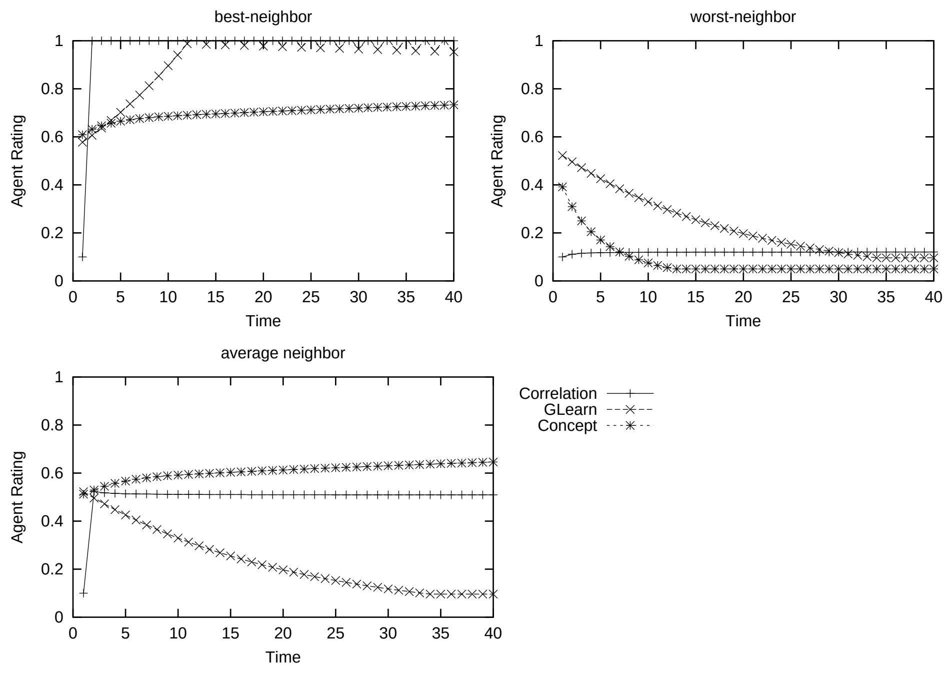


Figure 4.1: A comparison of the variation of agent ratings with time

the three approaches is compared in Figure 4.1. The figure compares the change in ratings of three neighbors of the active user - *best-neighbor*, *worst-neighbor*, *average-neighbor* - in each of the three approaches. The preferences of the *best-neighbor* was set exactly the same as the active user. All the neighbors were arranged in an increasing order of the root-mean square difference ($\sqrt{\sum (U_a - U_y)^2}/N$) of their preferences with that of the active user. The *average-neighbor* and the *worst-neighbor* were chosen to be the middle and the last users respectively, in this list. One of the interesting observations from these graphs is that the *GLearn* approach does not distinguish between an average and a bad neighbor (both are treated as bad).

4.3.2 Learning Curve

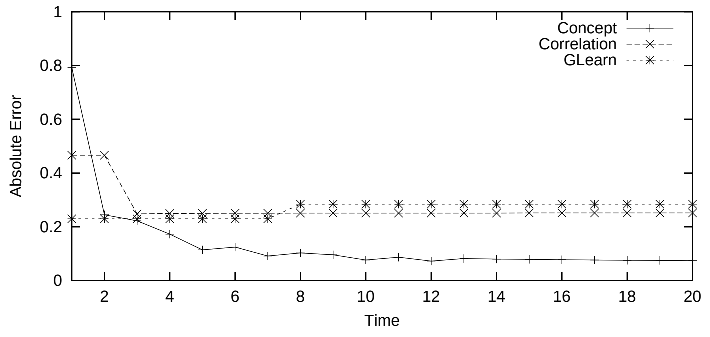


Figure 4.2: Time taken to reach steady state - comparison

In order to compare the three approaches based on the time they take to reach the steady state (after which their prediction accuracy remains almost constant), we initialize the active user with a *null* observation matrix and let him predict the scores of service providers in a chosen *category*. Thus the setup is similar to the previous one except that we now have the *category* field fixed in all the tasks. From the graph in Figure 4.2 we see that the concept based approach converges slightly faster than the other two approaches towards the steady state. Also, the absolute error made in the concept based approach is lower than the error in the other two approaches (for the artificial database).

4.3.3 Steady State Accuracy Test

We compare the accuracy of the three approaches in their steady state by recording the errors (both positive and negative) made during predictions (Figure 4.3). The tasks for this experiment is a long list of randomly chosen, but legal, tasks. Thus there are multiple *active-users* making predictions for service providers in multiple *categories*. In order to

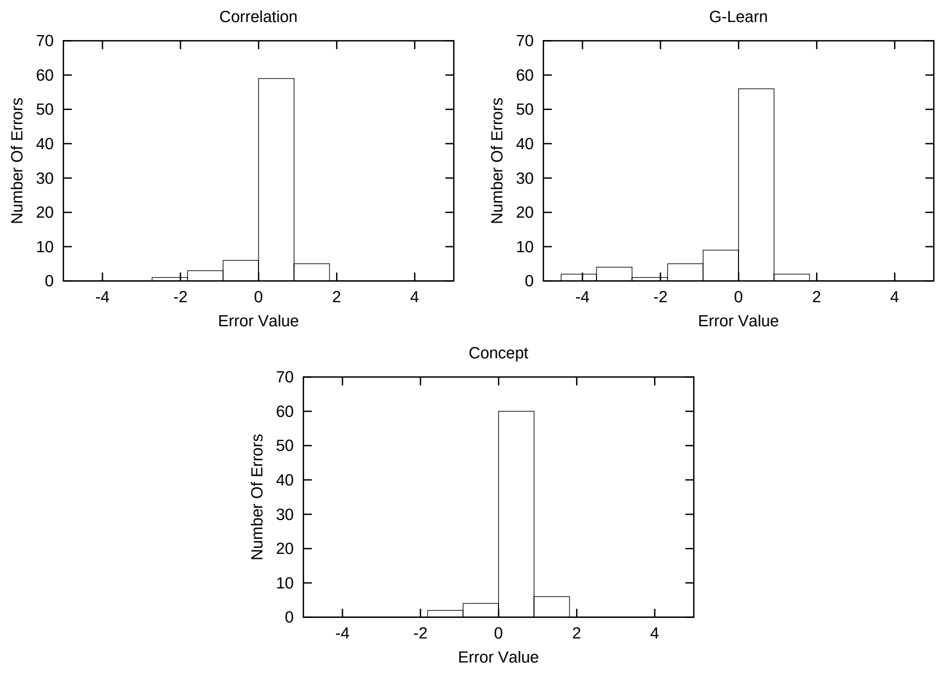


Figure 4.3: Steady state accuracy test - MovieLens

represent the real-world, the MovieLens database was chosen in this case. All the active agents were initialized with 70% of the *Truth* matrix (M).

4.3.4 Ordinal Error Test

In the steady state accuracy test, we record the error value of prediction irrespective of the quality of prediction. That is, we measure the error value whether or not the winning service provider was indeed the right choice. In order to measure the quality of the choice itself, we define a value called the *ordinal error*. This value is simply the rank of the winning service provider among the list of service providers considered in the current iteration, with respect to the active user. Note that we can measure this value since we have access to the truth matrix (M). The setup is exactly same as the one for the steady state accuracy test. Plots

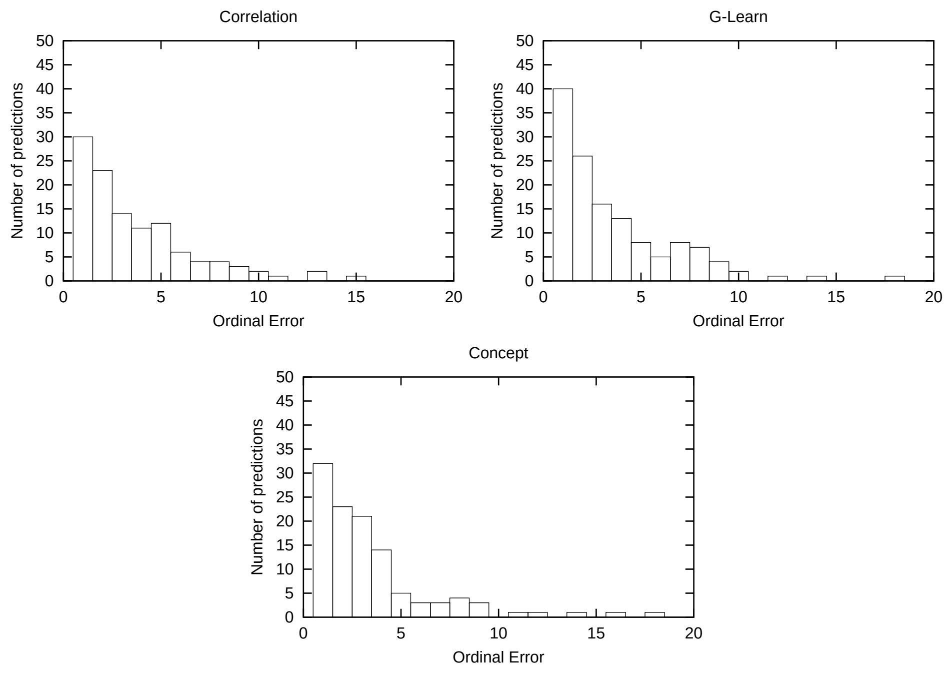


Figure 4.4: Ordinal error test - MovieLens

of the ordinal errors made in the three approaches is shown in Figure 4.4. The sum of the number of predictions with ordinal errors one, two, and three was computed for each approach. It was learnt that the performance of the *GLearn* predictor was better than the *Concept-based* predictor, which in turn was better than the *Correlation-based* predictor. However, the mean ordinal error in the *Concept-based* predictor was equal to 3.4, which was better than that for *GLearn* predictor which was equal to 3.6, which in turn was better than the mean ordinal error of 3.7 for the *Correlation-based* predictor.

Chapter 5

Discussion

In this chapter, we describe literature that is related to our work. We also describe two important concept lattice algorithms that we had to implement in our approach. We end the chapter with a discussion of the extensions that could be made to our work.

5.1 Related Work

The main focus of this work is service selection. One of the ways of selecting services is by defining Quality of Service (QoS) parameters, and specifying the value that the user attaches to each of these parameters. *Service level agreements* are made once a service willing to adhere to these demands is found. Our approach, on the other hand, is closer to the field of *information filtering*, although extended to apply to service selection. Classical information filtering techniques fall under two main categories: (1) Content based or cognitive filtering and (2) Social information filtering.

5.1.1 Content-Based Filtering

The most well-known techniques of content-based filtering is keyword-based search such as the one adopted by search engines like Google. Primitive QoS-based filtering could be introduced for keyword searches, by identifying the right keywords for search and combining them with boolean operations such as AND, OR, and NOT. Techniques such as term frequency-inverse document frequency (TF-IDF) are based on the view that the least frequently occurring words and the number of times such words appear in a document are the most likely factors that decide the relevance of a document. Latent Semantic Indexing (LSI) is a popular approach used to minimize the adverse effects of synonyms (many words used to convey the same meaning) and polysems (one word having many meanings) in a query based search [Dumais et al., 1988]. TF-IDF and LSI, along with techniques to measure correlation of the query with a document, such as the vector similarity measurements, aid the content-based filtering approach.

5.1.2 Social Information Filtering

Social information filtering refers to the generic technique of selecting articles based on relationships between people and on their subjective judgments. Such filtering, in general, could be for an individual or for a group. Placing an author in the “preferred list” is a crude example. Collaborative filtering [Goldberg et al., 1992] is a form of social information filtering in which people collaborate to help one another to perform filtering by recording their reactions to services they use (e.g., rating documents read).

The aspects in which the different collaborative filtering techniques differ is discussed in the following subsections:

Implicit vs. explicit voting

Implicit voting involves estimating the user's preferences without the user having to explicitly express his rating, by observing patterns like: purchase history, browsing pattern, and the amount of time spent reading an article. As opposed to this, in explicit voting systems, such as GroupLens [Resnick et al., 1994] and Ringo [Shardanand and Maes, 1995], the user rates the service he uses (in this case, the articles that he reads) on a scale of 1 to 5 and 1 to 7 respectively.

Memory-based vs. model-based approaches

One of the main classifications of collaborative filtering techniques is into memory-based and model-based approaches. In the memory-based approach, a history of user preferences (ratings) is stored in a database. The system makes a definite prediction of the articles to be recommended based on this history using correlation between users as the deciding criterion. Some algorithms that calculate similarity between users are compared in [Breese et al., 1998]. The model-based approach is a probabilistic approach. It involves calculating the probability of a user liking a particular service. The crux is to identify a set of *features* that are most useful in making a prediction. Features are either present or absent in a service. These *features* could also be memory dependent, e.g., "Alice likes service *x*". Once such features are identified, they are used to predict the probability of the user liking a service based on his previous experiences. Feature selection, which is critical for the success of this approach, is discussed in [Basu et al., 1998]. The more popular model-based approaches are the Bayesian approaches. Breese *et al.* discuss a simple Bayesian network approach [1998]. Billsus and Pazzani discuss a Bayesian classification approach with emphasis on ways to reduce the number of *features* being considered [1998].

Offline vs. online filtering

The methods described above are *offline filtering* methods. These methods make use of a matrix representing a record of the history of the users of the system. This matrix is used to make predictions for the users. As opposed to this approach, *online filtering* methods treats the prediction process as one that is continuous and interactive. They make use of a *mistake-bound* model that has a *learner* that corrects itself after each iteration. Thus there is more personalizing of the recommender. The theoretical framework for one of the popular approaches, called the *Weighted Majority Algorithm* is discussed in [Littlestone and Warmuth, 1994]. The idea here is to attach weights to individual algorithms in a pool of algorithms called *expert predictors*. These algorithms make binary predictions (yes/no) given an instance of the problem. The master algorithm, calculates a weighted majority of these predictions and comes up with its own binary prediction. If the prediction is wrong, the master algorithm reduces the weights attached to all the algorithms that predicted wrong. This generic WMA can be used to predict user preferences treating the ratings of other users as the ratings by various algorithms, and the "learner" as the master algorithm that learns how much weight to attach to each of the users. This approach for a binary prediction is discussed in [Goldman and Warmuth]. Generalized versions of this learning algorithm, for a range of ratings (not just binary), is discussed in [Nakamura and Abe, 1998] and [Delgado and Naohiro, 1999].

5.2 Concept Lattice Algorithms

Section 3.1 gives a brief introduction to the theory of concept lattice [Ganter and Wille, 1999]. The following are the two main algorithms that were used in our simulations.

Identifying Concepts

The algorithm given in [Ganter and Wille, 1999] was used to get a list of all concept *extents* of a *context* (G, M, I) and is shown in Algorithm 2.

Algorithm 2 Get-Concepts()

```
1: The extent $G$ is entered into the list. Then we carry out the following for each attribute $m \in M$ (the attributes are processed in an arbitrary order.
2: For each set A, entered into the list in an earlier step, we form the set $A \bigcap m'$ and include it into the list, provided it is not yet contained within it.
```

Building the Concept Lattice

Once the concepts are identified, a lattice has to be built from them (capturing the parent-child relationship). A fast algorithm to build concept lattice described in [Nourine and Raynaud, 1999] was considered, but was given up in preference to a "dog-work" algorithm, for the sake of simplicity. The algorithm that was adopted is shown in Algorithm 3

Algorithm 3 Build-Lattice()

```
1: Place each concept in the list of concepts in a bucket representing the size of the extent.
2: For each concept in a bucket representing size $n$, add all the concepts in buckets representing size $n + 1$ or higher to its parent list.
3: For each concept, remove concepts that are found in the parent list of any other concept that is a parent of the concept under consideration.
```

5.3 Directions

We have identified the following directions in which our work could be improved or extended.

5.3.1 Bootstrapping Agents

One of the important problems in a real-life community is accomodating a new user. This would translate to bootstrapping the agent that represents this user in the community. In our experiments, we assume that an *observation* matrix is available for every agent in order to bootstrap the agent. This matrix is not very useful unless there is a row representing the active user, i.e., the observation matrix cannot be used to bootstrap the agent unless there is standard of comparison (which in our approach is the active user's row) is available to evaluate the raters. We could measure the deviation of a user rating of a service provider from the average rating. But, this would not be a personalized rating of the rater. A hybrid system — a mix of *reputation* and *referral* systems — could be considered in this case.

5.3.2 Decreasing Confidence

The confidence we attach to our rating of a rater goes up after every evaluation of the rater (agent), irrespective of whether or not the rating of the rater itself is going up. But, in real life there could be ambiguities that arise after every evaluation of the rater, this should translate to a loss in confidence (e.g., when an agent behaves erratically or changes preferences). However, we have not explored this direction since we do not consider the case of varying interests of raters. Barber and Kim discribe a probabilistic approach to decide whether a knowledge gained from a source is true or false [2001]. The probability of the source being reliable is also measured and taken into consideration while measuring the confidence in the knowledge. A similar approach could be adopted to associate confidence in ratings.

5.3.3 Neighbor Selection

In our experiments, we maintain a list of ten neighbors and evaluate twenty raters during every iteration, i.e., the score table contains ratings given by atmost twenty raters. At the end of the iteration, we update the list of neighbors to contain the top ten raters. This list is used to make sure that the scores given by any of the neighbors is retained while building the score table during the next iteration. The number of neighbors maintained and the number of raters chosen during every iteration could influence the quality of decision and should be typically driven by a local policy described by the user.

5.3.4 Using the Hierarchical Classification Scheme

A hierarchical classification scheme for the service categories was described in Section 1.2.3. We believe that this scheme would be very useful in classifying service categories in a world where all possible service categories are described. Such a classification would help locate the right kind of service provider. However, in our experiments, the usefulness of this scheme was not obvious since the classification of the categories was either very simple (as in the artificial database) or non-existent (as in the MovieLens database).

Bibliography

K. Suzanne Barber and Joonoo Kim. Belief revision process based on trust: Agents evaluating reputation of information sources. In R. Falcone, M. Singh, and Y.-H. Tan, editors, *Trust in Cyber-societies*, LNAI 2246, pages 73–82. Springer-Verlag, 2001.

Chumki Basu, Haym Hirsh, and William W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 714–720, 1998. URL `http://citeseer.nj.nec.com/basu98recommendation.html`.

Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA, 1998. URL `http://citeseer.nj.nec.com/billsus98learning.html`.

John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998. URL `http://citeseer.nj.nec.com/breese98empirical.html`.

Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, London, UK, 1996. Practical Application Company. URL `http://citeseer.nj.nec.com/chavez96kasbah.html`.

Mao Chen and Jaswinder Pal Singh. Computing and using reputations for Internet ratings. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 154–162. ACM Press, 2001. ISBN 1-58113-387-1. URL `http://doi.acm.org/10.1145/501158.501175`.

Joaquin Delgado and Ishii Naohiro. Memory-based weighted majority prediction for recommender systems, 1999. URL `http://citeseer.nj.nec.com/delgado99memorybased.html`.

Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'88*, 1988. URL `http://citeseer.nj.nec.com/dumais88using.html`.

Epinions. Home page, 2002. http://www.epinions.com.

Bernhard Ganter and Rudolf Wille, editors. *Formal Concept Analysis*. Springer, Berlin, 1999. ISBN 3-540-62771-5.

Gnutella. Home page, 2001. http://gnutella.wego.com.

David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

Sally A. Goldman and Manfred K. Warmuth. Learning binary relations using weighted majority voting. In *Proceedings of the 6th Annual Conference on Computational Learning Theory*, pages 453–462. ACM Press. ISBN 0-89791-611-5.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Information and Computation, 108(2)*, pages 212–261, 1994. URL `http://citeseer.nj.nec.com/littlestone92weighted.html`.

MovieLens. Home page, 2002. http://movielens.umn.edu.

Atsuyoshi Nakamura and Naoki Abe. Collaborative filtering using weighted majority prediction algorithms. In *Proceedings of the 15th International Conference on Machine Learning*, pages 395–403. Morgan Kaufman, 1998.

Lhouari Nourine and Olivier Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71(5-6):199–204, 1999. ISSN 0020-0190.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstorm, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM. URL `http://citeseer.nj.nec.com/resnick94grouplens.html`.

Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating “word of mouth”. In *Proceedings of ACM CHI Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, 1995. URL `http://citeseer.nj.nec.com/shardanand95social.html`.

UDDI. Universal Discription Discovery and Integration, 2002. http://www.uddi.org.

WSDL. Web Services Description Language, 2002. http://www.w3.org/TR/wsdl.

Pınar Yolum and Munindar P. Singh. An agent-based approach for trustworthy service location. In *Proceedings of the 1st Workshop on Agents and Peer-to-Peer Computing (AP2PC)*. Springer, 2002. To appear.

Giorgos Zacharia, Alexandros Moukas, and Pattie Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1999. URL `http://citeseer.nj.nec.com/zacharia99collaborative.html`.