Multi-Agent Information Classification Using Dynamic Acquaintance Lists

Snehasis Mukhopadhyay, Shengquan Peng, Rajeev Raje, and Mathew Palakal
Department of Computer and Information Science, Indiana University-Purdue University Indianapolis, 723 West Michigan Street SL 280, Indianapolis, IN 46202. E-mail: {smukhopa, speng, rraje, mpalakal}@cs.iupui.edu

Javed Mostafa
Laboratory of Applied Informatics Research, Indiana University, Bloomington, IN 47405-3907. E-mail: jm@indiana.edu

There has been considerable interest in recent years in providing automated information services, such as information classification, by means of a society of collaborative agents. These agents augment each other's knowledge structures (e.g., the vocabularies) and assist each other in providing efficient information services to a human user. However, when the number of agents present in the society increases, exhaustive communication and collaboration among agents result in a large communication overhead and increased delays in response time. This paper introduces a method to achieve selective interaction with a relatively small number of potentially useful agents, based on simple agent modeling and acquaintance lists. The key idea presented here is that the acquaintance list of an agent, representing a small number of other agents to be collaborated with, is dynamically adjusted. The best acquaintances are automatically discovered using a learning algorithm, based on the past history of collaboration. Experimental results are presented to demonstrate that such dynamically learned acquaintance lists can lead to high quality of classification, while significantly reducing the delay in response time.

1. Introduction

In recent years, cooperative information agents have been introduced as an important new methodology for designing complex information systems (Communications of the ACM Special Issue, 1994). Such systems provide information services such as searching, retrieving, classification, and filtering to users or user communities. The multiple agents cooperate with each other in order to speed up computation, complement each other's capabilities, share each other's knowledge structures, or improve the efficiency/quality of information services.

In Raje et al. (1997a; 1997b) a multi-agent information classifier system was introduced, where the agents were equipped with disparate vocabularies/thesauri enabling them to represent and classify documents related to their own thesaurus. (In this paper, we use the term 'thesaurus' to mean a linear list of keywords describing a domain of interest with an imposed synonym structure). Such information classification has been used as a crucial intermediate stage for both single-agent (Mostafa et al., 1997) as well as multi-agent (Raje et al., 2000) information filtering systems. In the case of multi-agent information classification and filtering, the disparate nature of the thesauri of the agents may be caused by the different domains of information, or independent term discovery or adaptation processes in the agents. For example, the agents may be personalized to different users' interests covering different domains. When an agent is unable to represent and classify an incoming document using its own thesaurus, it seeks help from other agents (possibly with different thesauri). In experimental studies on classification tasks involving up to 3 agents with disparate thesauri in the area of computer science documents, a performance improvement of up to 61% was observed as compared to the performance of a single agent operating in isolation (Raje et al., 1997a).

In all such multi-agent information systems (including the multi-agent classifier system), very little is known about the scalability of these systems with increasing numbers of agents. In particular, if an agent interacts with all other agents, the communication overhead as well as the response time for the given agent, may be expected to grow at least linearly with the total number of agents in the agent society. Hence, exhaustive interaction with all other agents becomes

Received July 29, 2002; revised December 17, 2002; accepted March 20, 2003

practically infeasible for relatively large-scale open information agent societies.

To circumvent this problem, the notion of acquaintance lists has been introduced in the multi-agent research literature (Salampasis, 1995). An agent explicitly maintains a list of other agents that it wishes to interact with. In the event that a desired performance level can be achieved with relatively few acquaintances, such a control mechanism for interaction can significantly reduce the response delay and collaboration overhead. The acquaintance list mechanism represents the underlying agent society by means of an acquaintance graph structure which, unlike exhaustive interaction, is not completely connected.

A problem that arises in such selective agent interactions using acquaintance lists, is how the acquaintances of a particular agent are determined. In an off-line method, such acquaintances may be hard-coded by the system designer. However, this results in a considerable amount of effort on the part of the designer and may not result in the best set of acquaintances. Further, in an open, relatively large-scale agent society that is possibly distributed over a wide area, agents may cease to operate or new agents may originate in an unpredictable manner. In such an environment static acquaintance lists may not be adequate for dealing with dynamic changes in the agents society. Therefore, there is a need for methods to automatically learn about the "best" acquaintances based on suitable agent models created and updated using on-line interaction experiences.

In this paper, we present the notion of dynamic acquaintance lists where the acquaintances of an agent are learned and updated by means of a learning algorithm, based on the history of past interactions. The algorithm starts with a random list of acquaintances and gradually evolves to selecting the best acquaintances, according to some criteria. We study such adaptive acquaintance lists in the context of a multi-agent information classification system. We measure the classification performance both in terms of number of successful classifications, as well as by means of an algorithm-specific classification quality measure. We show that near optimal classification performance can be achieved with adaptive acquaintance lists in both cases. These experimental studies indicate that learning acquaintances is a promising alternative to assigning them in a pre-determined manner. Hence, such dynamic acquaintance lists constitute an essential approach to designing relatively large-scale multi-agent information systems.

The paper is organized as follows. In section 1.1, we present some related work in the areas of multi-agent systems and multi-agent information systems, and place this paper in the context of such work. In section 2, we present the background and preliminary discussions on information classification and multi-agent information classification. This is essential in understanding the main contributions of this paper which are described in sections 3, 4, and 5. Section 3 presents experimental studies motivating the need to use acquaintance lists. Section 4 describes the algorithm used to learn and reconfigure acquaintances in multi-agent information classification. Section 5 describes experimental results with dynamic acquaintance lists for multi-agent information classification without and with agent failures.

1.1. Related Work

Multi-agent Systems have grown out of the Distributed Artificial Intelligence community. Durfee (Durfee & Montgomery, 1989) defined a multi-agent system (MAS) as "a loosely coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities." These problem solvers, which are essentially autonomous, distributed, and maybe heterogeneous in nature are called "agents" and usually have a single focus of control and/or intention. Multi-agent systems offer a way to relax the constraints of centralized system and provide systems a decentralized, emergent and concurrent environment. The advantages offered by multi-agent systems are (Baker, 1996): 1. Fault-tolerance: agents are an inherently distributed mechanism and thus a system made of autonomous agents will not collapse when one or more of its components fail as there will not be any single point of failure. 2. Modular software, scalable architecture: The reason agents are such powerful entities is because of the factorization of the problem they provide. Each agent can be identified as an entity and thus facilitate incremental growth and flexible expansion. The advantage of scalability is provided as each agent can join a system, start working with other agents, or just leave a system once it has finished a plan it was engaged in without affecting the operation of the system. 3. Flexible systems: agents with different abilities can dynamically team up to solve current problems.

In Sycara and Zeng (1996) and Sycara et al. (1996) the authors investigated techniques for developing a distributed and adaptive collection of information agents that coordinate to retrieve, filter and fuse information relevant to the user, task, and situation, as well as anticipate user's information needs. They presented a distributed system architecture, called RETSINA (Reusable Task Structure-based Intelligent Network Agents), which has three types of agents: interface agents, task agents, and information agents. In the system of agents, information gathering is seamlessly integrated with decision support. The MACRON (Multi-agent Architecture for Cooperative Retrieval Online) multi-agent system (Decker & Lesser, 1995) uses a centralized planner to generate sub-goals that are pursued by a group of cooperating agents, using KQML, a standardized language for inter-agent communication and negotiation. Amalthaea (Moukas, 1996) is an evolving, multi-agent ecosystem for personalize filtering, discovery and monitoring of information sites. Two different categories of agents are introduced in the system: filtering agents that model and monitor the interests of the user and discovery agents that model the information sources. These two agents compete and cooperate to serve the user. SIGMA (System of Information Gathering Market-based Agents) (Karakoulas & Ferguson, 1995; Ferguson & Karakoulas, 1996) involves decentralized

decision making for the task of information filtering in multi-dimensional document spaces such as the Usenet news. Different learning and adaptation techniques are integrated with SIGMA for creating a robust network-based application that adapts to both changes in the characteristics of the information available on the network as well as to changes in individual user's information interests. RAAP (Research Assistant Agent Project) (Delgado et al., 1998) was devoted to support collaborative research by classifying domain-specific information retrieved from the Web, and recommending these "bookmarks" to other researcher with similar research interests.

To our knowledge, the relatively narrow but well-defined problem of multi-agent information classification using multiple agents with different thesauri (vocabularies), has not been investigated by others. This is the problem considered in the current paper. Further, a solution approach is presented for the problem of learning acquaintances (the most promising remote classification agents) and reconfiguring them in the presence of agent failures. This is an important contribution of this paper, which, to our knowledge, has not been presented in the literature.

2. Multi-agent Information Classification—Background and Preliminaries

As stated in the introduction, the overall problem being addressed by the society of agents is one of information classification. This is accomplished in each agent by first representing a text document as a vector of real numbers and then classifying the vector using an unsupervised clustering algorithm. The particular representation algorithm used is the *tf.idf* (term frequency-inverse document frequency) method, based on a vector-space model (Salton, 1988). In this, a domain-specific thesaurus (vocabulary) is used that contains a set of terms from the domain of interest. The elements of the vector qualitatively correspond to the strength or importance of the corresponding terms in the thesaurus. The classification algorithm used by each agent is based on the Maximin Clustering (Tou & Gonzalez, 1974) where a set of centroid vectors are determined from a representative collection of documents as an a priori operation, and classification is conducted on-line based on similarity of new documents to centroids. The representation and classification algorithms are briefly described in Section 2.1 and 2.2 respectively.

The distinguishing feature between the multitude of agents is assumed to be the disparate nature of their thesauri. As mentioned in the introduction, this may be due to several factors including, difference in the domains of information that the agents are designed for, personalization of each agent's thesaurus to different users, and independent automatic term discovery process carried out by agents.

When an agent's own thesaurus is inadequate to represent an incoming document, the representation algorithm generates a so-called *null vector*: a vector where all elements are zero. In such a case, the classification algorithm labels that document as unclassified. However, in the agent society, there may be a remote agent with a different thesaurus that may have the ability to classify a document that cannot be classified by the local agent. The objective of multi-agent classification is to minimize the occurrences of unclassified documents through collaboration among multiple classifier agents by exploiting the natural diversity of their thesauri. In the following sections key components and architecture of the system, associated algorithms, and highlights of experimental results from earlier studies are described.

2.1. Representation of Documents

The representation algorithm is responsible for converting a document into a numeric structure that can be used for classification. To accomplish this, we use the vector-space model (Salton, 1988) for representation and the popular tf.idf technique to establish a degree of importance for each term in that document. In this algorithm, the following equation is used to calculate the elements of the document vector:

$$W_{ik} = T_{ik} * \log(N/n_k)$$

where T_{ik} is the number of occurrences of term T_k in the current document *i,* $\log(N/n_k)$ is the inverse document frequency of term T_k in the entire document set, N is the total number of documents in the document set, and n_k is the number of documents in the set that contains the given term T_k. It is this vector that is used for classification. If none of the terms present in the thesaurus occur in the document, all elements of the vector will be zero. This is called a null vector which results in a classification failure.

2.2. Classification of Documents

The classification of documents is based on a similarity measure between the documents (or, more precisely, their document vectors). The measure used for computing the similarity between two document vectors is the cosine similarity measure (Salton, 1988). Given two non-null document vectors $X = [x_q, \ldots, x_t]^T$ and $Y = [y_q, \ldots, y_t]^T$, the similarity measure represents the cosine of the angle between them and is given by

$$\sum_{i=1}^{t} x_i y_i \Bigg/ \sqrt{\left(\sum_{i=1}^{t} x_i^2\right)\left(\sum_{i=1}^{t} y_i^2\right)}.$$

The classification consists mainly of two processing stages: an unsupervised cluster learning stage and a vector classification stage. During the learning stage, initial cluster hypotheses $[C^1, \ldots, C^k]$ are generated from a representative sample of document vectors $[S^1, \ldots, S^N]$. Each cluster C^i is then represented by its centroid, Z^i. During the classi-

fication stage, an incoming document vector V^i is classified into a particular class C^k using the learned centroids from the first stage. The learning of cluster centroids is done in an off-line batch mode while classification is carried out continuously as documents arrive.

A simple heuristic unsupervised clustering algorithm, called the Maximin-Distance algorithm (Tou & Gonzalez, 1974), is used to determine the centroids over the document vector space. In this iterative algorithm, at each stage, a document vector from the document set is selected that has the least similarity with the existing centroids. The similarity of this document vector with the existing set of centroids, in turn, is the maximum of its similarities over all centroids. The selected point is then added as a new centroid if and only if its similarity with the the existing set of centroids is less than an implementation-specified threshold parameter. This process is continued until no new centroids can be identified.

During classification each new document vector is classified to one of the centroids that has the largest similarity with the document.

2.3. Multi-Agent Collaborative Classification

The goal of any information classification is to obtain the most successful and accurate classification. Multi-agent collaborative classification is built upon a single agent classifier by creating an inter-connected environment of agents to maximize the possibility of achieving this goal. In this environment, all agents are allowed to collaborate over the network with the intention of achieving a higher rate of successful document classification for their users than what is possible only by the local agent. As mentioned earlier, the agents differ in terms of their thesauri and their ability to classify different domains of documents. This section provides a brief description of the collaborative methodologies behind multi-agent information classification as well as some results of earlier experiments.

(i) The collaboration environment. The experimental environment consisted of a variable number of classifier agents executing on Sun Sparc machines via Ethernet with the Solaris 2.3 operating system. The agents were implemented using JAVA and agent collaboration was achieved using the JAVA-RMI (remote method invocation) utilities. A set of 81 terms from the domain of computer science constituted the *master thesaurus* which was broken up into 9 disjoint smaller thesauri, and distributed among the agents. The list of terms in the thesaurus is not presented here to conserve space, but can be found in Mostafa et al. (1997, p. 383) Hence, when the number of agents were greater than nine, some agents could have same or overlapping thesauri. In our earlier studies, the communication between the agents were indirect in nature, i.e., based on a common server. However, in the experimental studies presented in the paper, the agents communicated directly with each other. Since multicasting to multiple agents was not possible using the JAVA remote method invocation (RMI) method, it was simulated by repeated sequential communication of requests to remote agents (e.g., the acquaintances). Despite the difference in the underlying thesauri, all classifiers functioned in an identical manner, that is they implemented the same representation and clustering algorithms discussed earlier.

(ii) Collaborative models. Two kinds of collaborative models were developed: single-opinion model and multiple-opinion model. Both models used a simple overall algorithm—*Attempt to classify my own documents first, then if there is time, try and help other remote agents* (Raje et al., 1997a; Raje et al., 1997b). The major difference between these models is the number of remote agents that are given the opportunity for classification. The single-opinion model allows only one remote agent an attempt at the classification. This is not desirable for two reasons. First, the agent that attempts the remote classification is randomly chosen and secondly, due to this randomness, the selected agent may not be able to classify that document successfully. These two drawbacks are eliminated in the multiple-opinion model. The multiple-opinion model allows all remote agents an attempt at classifying a remote document. It represents the real-world by allowing different agents (with varied expertise) to participate in the collaboration. Such a collaboration has been found to result in better classifications (Raje et al., 1997a; Raje et al., 1997b). The most difficult problem of this model involves managing multiple remote agents. Critical issues include waiting for all agents to complete their classification and the overhead of determining the best classification (also known as selection criteria). Both single- and multi-opinion models make use of a "time-out" period. This wait period has implication for the ultimate utility of classification, i.e., once an agent asks for classification assistance, how long should it wait before the use of the document or the document itself becomes obsolete?

3. An Experimental Study Motivating the Need for Using Acquaintance Lists

In this section in order to motivate the need for an intelligent means of acquaintance selection, we present additional experimental results on the impact of increasing the number of agents in a multi-agent environment on classification performance.

3.1. Response Time and Classification Performance with an Increasing Number of Agents

In the multiple-opinion model described in Section 2.3, it was assumed that an agent, when confronted with a locally unclassified document, seeks collaboration from all other agents. In previous studies, this method of exhaustive col-

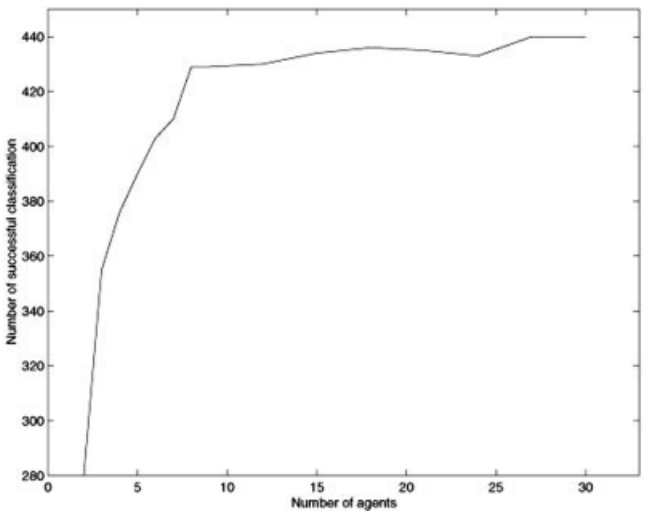

FIG. 1. Classification success rate.

laboration was possible due to the small number of agents (e.g., only three agents). As the number of agents is increased, the overheads and delays with exhaustive communication is expected to increase. In order to study the effects of increasing the number of agents, we conducted classification experiments with a relatively larger number of agents (up to 30 agents) employing the multiple-opinion model for collaboration. Our objective was to experimentally study the classification performance in terms of the number of successful classifications as well as the average response time (i.e., the average time needed for local and collaborative classification) with increased number of agents engaged in exhaustive collaboration. A classification was defined as successful if the result was a non-zero similarity value or a non-null vector assignment (treated as the same).

In the experimental scenario, one agent was engaged in classifying 500 documents, while all other agents were idle except for responding to remote requests. Figure 1 shows the percentage of successful classifications for the given busy agent (including local and remote classifications), while Figure 2 shows the corresponding response time, as the number of agents is increased, beginning with 2 and with a maximum of 30 agents. It is seen that, in the given situation, the classification performance quickly saturates with increased number of agents. For example, even with as few as eight agents, we get 429 successful classifications, while with 30 agents the performance improves further only by 11 more successful classifications. On the other hand, the average response time (averaged over all documents) is seen to grow almost linearly with the number of agents. For example, with eight agents, the response time is 552 milliseconds, while the response time is 897 milliseconds with 30 agents. This clearly demonstrates that even when there are as many as 30 agents, it might be possible to achieve a satisfactory performance by interacting with only eight agents. In fact, as we will demonstrate later, by selecting the agent properly, it might be possible to achieve 92% of the maximum achievable performance by interacting with only four out of 30 agents. On the other hand, interacting with only four instead of all 30 agents will reduce the response time significantly. We believe that such a behavior is generic in nature irrespective of the particular document domain used in these studies.

In the case of such selective interactions, we call the selected agents as acquaintances of the given agent. Thus, each agent is assumed to have an acquaintance list which is a subset of the set of all agents. An agent will seek collaboration only with agents in its acquaintance list, and not with every other agent.

4. A New Approach to Dynamic Learning of Acquaintances

In this section, we describe the main problem we concentrated on in this study and details of the learning algorithm known as the pursuit learning algorithm. We also describe a higher level method of reconfiguring the lower level acquaintance learning algorithm to deal with dynamic changes in the agent environment.

4.1. Problem Statement: Dynamic Learning of Acquaintances

Denoting the set of agents by $A = \{A_1, \ldots, A_N\}$, the acquaintance list of an agent A_i is defined to be L_i, which is a subset of set A. It is assumed that n_i, the cardinality of L_i (the number of elements in L_i), is known from prior analysis. For example, in the multi-agent classification problem, from experimental analysis, it was decided that having $n_i = 4$ acquaintances is adequate in an agent society with 30 agents.

The problem is to determine which n_i agents out of the set of A should be chosen as the members of L_i, so as to maximize the classification performance for agent A_i. The classification performance may in turn be defined by the rate

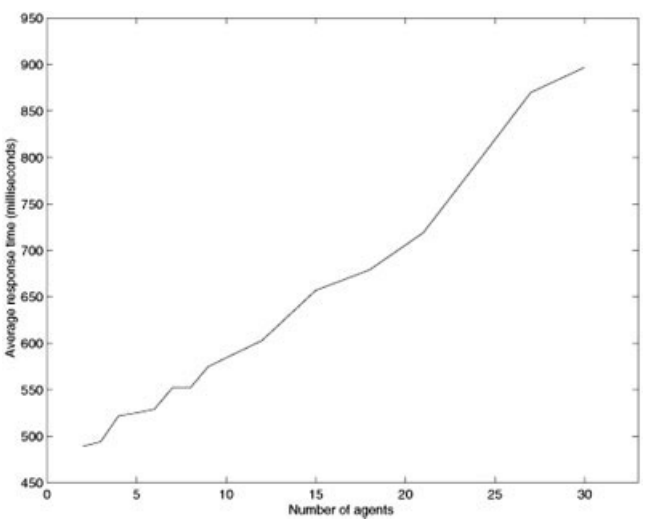

FIG. 2. Response time with increasing number of agents.

of successful classifications or by some algorithm-specific classification quality measure as discussed later in the paper.

In an off-line approach to determine the acquaintances, the system designer may hard-code the acquaintances based on knowledge of their respective thesauri and the document stream. However, in addition to the considerable effort required on the part of the designer, this approach will perform poorly in dynamic environments where new agents come up or existing agents terminate unpredictably and/or the nature of the document stream changes.

Our objective in this paper is to propose a method for an agent to automatically learn the "best" acquaintances on the basis of past experiences of interaction. This method is based on a simple reinforcement machine learning algorithm, called Pursuit Learning (Thathachar & Sastry, 1985; Mukhopadhyay & Thathachar, 1989). This algorithm is described in greater detail in section 4.2 in a general context. Its adoption to the problem of learning of acquaintances is in section 4.3.

It should be noted that in the experiments reported in this paper we adopt the multiple opinion model described in section 2 without any underlying economic framework. However, the implications of using agent modeling and acquaintance lists are strong even when an economic framework is used for collaboration. This is because such agent models can be used to determine bids, as well as to select the winning bid.

4.2. Pursuit Learning: A Simple Reinforcement Learning Algorithm

The paradigm of reinforcement learning assumes an agent, with a finite number of actions, operating in a feedback loop with an unknown teacher and environment. At each stage, the agent performs one of the set of actions and receives a stochastic, qualitative reinforcement (e.g., reward/penalty signal) indicating how well the agent is performing. The objective of the agent is to evolve an action strategy that maximizes its expected pay-off from the environment.

There are numerous algorithms proposed in the reinforcement learning literature that differ in respect to the environment (stationary/non-stationary, static/dynamic) and the methods of updating the agent's strategies (model-free/model-based, symmetric/asymmetric). The basic learning algorithm used in this paper is a model-based one, called Pursuit Learning (Thathachar & Sastry, 1985; Mukhopadhyay & Thathachar, 1989), that assumes stationary environments. This algorithm is briefly described below:

Suppose an agent with a finite number of actions performs one of the set of actions and receives a reward indicating how well it is performing. Let $p = [p_i]$ denote the action probability vector, such that p_i represents the probability with which the agent chooses action α_i, initially set to $1/n$. Let $d = [d_i]$ denote the underlying reward probabilities vector and $\hat{d}$ is the estimate of d vector. $\hat{d}_i$ is the estimated probability that action α_i results in a reward, which at any instant is the running average of the reward values for each action. The learning agent maintains and updates the p and $\hat{d}$ vectors of dimensions equal to the number of actions. α_m is the unknown optimal action (i.e., $d_m = max_{i=1}^{n}\{d_i\}$). S_i is the number of times action α_i has been tried so far. The desired behavior is that p_m should asymptotically approach 1, all other p_i's should approach zero. At instance k, choose an action according to p. Let $\alpha(k) = \alpha_j$ be the action chosen. Perform the action and receive the reinforcement β_k. Then update $\hat{d}_j$ by

$$S_j(k+1) = S_j(k) + 1$$

$$\hat{d}_j(k+1) = (S_j(k) * \hat{d}_j(k) + \beta_k)/(S_j(k) + 1).$$

Determine vector E such that $E_i(k) = 1$ if $\hat{d}_i(k) = max_{I=1}^{n}\{\hat{d}_l(k)\}$, otherwise $E_i(k) = 0$. After that, update p vector by

$$p_i(k+1) = p_i(k) + \lambda * (E_i(k) - p_i(k)).$$

The main idea behind the algorithm can be qualitatively described as follows. The objective of using the action probability vector is to provide a chance for the agent to try all actions probabilistically. If an action is attempted sufficiently often, the estimate of its reward (the corresponding element of $\hat{d}$ vector) will be sufficiently close to its true value. This implies that the E vector will be correct (i.e., its one element will result in the optimal action). This, in turn, will result in the convergence of the p vector to E, and hence, the optimal action. The convergence property of the Pursuit Algorithm has been proved in (Thathachar & Sastry, 1985). The convergence result states that the agent's probability of convergence to the correct optimal action can be made as close to one as desired by choosing the step-size of learning (η) sufficiently small.

4.3. Acquaintance Learning Using Pursuit Algorithm

In the context of learning of acquaintances in information classification, the actions of an agent correspond to selecting a specific agent (other than itself) as an acquaintance. Hence, the number of actions available to an agent is $(N - 1)$ when N is the number of agents in the society. The p vector and the $\hat{d}$ vector are distributed over the action set, i.e., the set of all other agents. A learning experiment for the agent takes place when a document arrives that requires remote collaboration (i.e., can not be classified locally). At that instant, the agent selects the first of its acquaintances by sampling the p vector. The remaining acquaintances are selected on the basis of ranking (from high to low) of the $\hat{d}$ elements of the rest of the agents. The document classification request is then sent to all the selected acquaintances. When the remote classification results come back, the "best" of them is selected on the basis of some criteria. In our

study, two different criteria for choosing the "best" classification were experimented with.

(i) *First Arrival:* first arriving result is the "best", i.e., the agent that can provide the fastest result will give the "best" classification.
(ii) *Classification Quality:* Since all selected remote acquaintances used the same Maximin Clustering Algorithm (albeit with different thesauri), as part of their classification they compute the similarity measures. It is assumed that in addition to the class labels, the agents also return the similarity of the document vector with the chosen class centroid. The remote agent that generates the largest similarity is assumed to provide the "best" classification.

Once the "best" returned classification is determined, the corresponding action (i.e., remote agent) of the requesting agent is given a positive reward of $+1$, and all other selected acquaintances are given penalties (a value of 0). The learning algorithm is subsequently used to update $\hat{d}$ and p vectors.

As the nature of documents change, different remote agents will result in best remote classification. Hence, the $\hat{d}$ vector, after convergence, achieved through the learning algorithm will quantitatively describe *the model of an agent's world* for a given agent. That is, it will indicate the suitability of a subset of remote agents as acquaintances. The p vector, in turn, will converge to (i.e., point to) the best acquaintance.

4.4. Reconfiguration of Acquaintances in Dynamic Agent Environment

When the agent society is dynamically changing, i.e., when existing agents fail or leave the society, and new agent may enter the society, there may be a need to reconfigure the acquaintances in response to such changes. In this section, we describe how this can be achieved for failure, leaving, and new entry situations.

(i) *Failure Situation:* When an existing acquaintance of an agent (learned by using the learning algorithm described in section 4.2 and 4.3) fails unexpectedly, the requesting agent has to discover this failure by itself and determine a new acquaintance to replace the old one. In our work, the failure was detected by a time-out mechanism, where the requesting agent assumes the failure of an acquaintance if it does not receive collaboration results from the remote agent within a given time for a number of successive remote requests. It is worth nothing that such failure, when correctly detected, may correspond to the agent failure or a network failure for the subnet containing that agent. Both these situations are treated in a similar manner. Also note that there is always a chance (however small) of false detection, such as when unusually long delays are caused by network congestion rather than network/agent failures. However, the requesting agent, after failure detection, quickly discovers a new acquaintance as a replacement. Hence, the performance may not be affected significantly.

The discovery of the new acquaintance as a replacement is carried out as follows. If a failure of an agent A_j is detected by a requesting agent in its k^{th} iteration of actions, the p and $\hat{d}$ vectors of the requesting agent are re-initialized. The p_j and $\hat{d}_j$ will be set to zero. For each of other agents, the p_i value will be updated as $p_i(k+1) = p_i(k)/(1 - p_j(k))$ while $\hat{d}_i$ keeps unchanged. After that, the agent will come up with new acquaintances in the same way by sampling the p and $\hat{d}$ vectors.

(ii) *Agent Leaving the Society:* This is a similar situation as (i), except that the agent is leaving in an orderly manner, i.e., after notifying all other agents. Hence, no failure detection needs to be carried out. However, the discovery of a new acquaintance requires to be carried out, as described in situation (i).
(iii) *New Entry in the Society:* As in the case of voluntary agent withdrawals, it is assumed that an agent, upon entering the society, will notify all other agents. A requesting agent consequently adds the agent as a potential acquaintance. Suppose there are N agents in the agent society, with the $(N+1)^{th}$ remote agent entering, the p and $\hat{d}$ vectors of the requesting agent are re-initialized in the following manner. The p value of the new agent will be set to $1/(N+1)$ while its $\hat{d}$ value is set to 0.5. For other agents, their p value will be their old p value times $N/(N+1)$ while $\hat{d}$ values are kept unchanged. Eventually, if the new agent offers good quality service, it has the potential to replace one of the existing acquaintances through re-learning.

5. Experimental Results

The experiments were carried out in an Ethernet local area network of Sun Sparc machines each running Solaris 2.3. Our test scenarios consisted of at most 30 agents, each executing on a different machine. The 500 test documents used for these experiments were taken from the domain of computer science, and the performance of classification was measured by the number of successful document classifications, average response time, and an algorithm-specific quality measure. We chose to implement the collaborative classification system in Java because of its portability, object-oriented features and RMI capability. The collaborative actions between different agents are implemented as remote calls over the network using RMI. Java-RMI provided a natural development environment due to its architecture neutrality as well as its simplicity and many other factors.

When an agent starts up, it registers with a name service. An agent requiring classification service can request from the name service the "handles" or addresses of all the remote agents in the agent society. We chose one specific agent as the "requesting" agent (i.e., agent requiring classification service) and for each experiment gave it 500 local documents to be classified.

The classification process is as following. If the requesting agent fails to classify any document locally, it then sends the document to four remote agents chosen from the acquaintance list. Based on the results it receives from the remote agents, the requesting agent updates the p and $\hat{d}$ vectors, which influences the rank order of agents in its acquaintance list.

Under different circumstances, users may be interested in different aspects of document classification. Sometimes users want to get their work done as soon as possible and do not care much about the quality of classification. On other occasions, users may have enough time and want to get the highest quality of service. To satisfy users' different interests, we have two methods to give rewards to the agent's actions (the first arrival and the highest classification quality), as discussed in section 4.3. In sections 5.1 and 5.2, we present experimental results involving these two criteria. Additionally, in section 5.3 we discuss some experiments examining agent failures.

5.1. Experiments with First Arrival as the Best Result

In these experiments with acquaintance learning using the pursuit algorithm, the first successful classification returned (again, a non-null classification is treated as a successful one) from a remote agent is considered as the "best" classification, and hence, the action associated with that remote agent will get a reward of 1. All the other actions associated with other agents in the acquaintance list will get a reward of 0 (a penalty). The goal of this method is to find out the acquaintances that can provide the most successful classifications.

Using the same document set and classification task we conducted the experiments by using four different methods to set up the acquaintance lists: (1) The best four remote agents (found through an off-line study) were hard-coded as static acquaintances; (2) Acquaintances were dynamically learned by using pursuit algorithm; (3) Acquaintances were selected randomly from all of other remote agents any time when a remote help was needed; and (4) The worst four remote agents (found through an off-line study as well) were hard-coded as static acquaintances. Out of these, the second method represents the method proposed in the paper, while the remaining three are for comparison purposes. The first method is theoretically the best possible and assumes complete knowledge on the part of each agent concerning the agent society. Table 1 shows the number of successful classifications achieved for the 500 documents across the four conditions. The rows represent successful classification by the local agent, successful classification by the remote agents, and the total number of documents successfully classified.

TABLE 1. Number of successful classifications with first arrival as the best result.

Acquaintances	Best four	Pursuit learning	Random four	Worst four
Local classifications	177	177	177	177
Remote classifications	229	213	153	46
Total classifications	406	390	330	223

TABLE 2. Average classification similarity of 500 documents achieved under four conditions.

Acquaintances	Best four	Pursuit learning	Random four	Worst four
Average similarity	0.491	0.461	0.360	0.216

The results in Table 1 provide a clear picture of the classification performance achieved by dynamically-learned acquaintances based on the pursuit-learning algorithm. With the dynamically-learned acquaintances, the agent can achieve almost optimal performance with the difference of only 16 documents to the acquaintances of the best four remote agents. Compared with the acquaintances of four random remote agents and the four worst agents, the number of remote classifications with dynamically-learned acquaintances increased about 39% and 363% respectively in terms of number of successful classifications. Therefore, the dynamically-learned acquaintances could provide very good performance in terms of number of successful classifications when it is designed to give responses to the user as fast as possible.

5.2. Experiments with Classification Quality

In the experiments reported in this subsection, an algorithm-specific similarity value (see section 2.2) was taken as the classification quality measure. In this case, a user wants to achieve the highest quality of classifications, and not merely the first available classification. In such a case, the requesting agent would wait for returned results from all of the remote agents that the document had been sent to. After receiving all of the returned results, the requesting agent would select the classification with the largest value of similarity and present it to the user. Hence, only the action that led to the largest similarity of remote classification would get a reward, which is equal to the value of similarity (a real number between 0 and 1). Other remote agents in the same acquaintance list would get a penalty (a value of 0). The goal of this method is to find out the acquaintances that can provide the best quality of classifications.

Similar to the experiments in section 5.1, we did the experiments by using four different methods to set up the acquaintance lists. The only difference was that the requesting agent waited for all of the returned results and selected the classification with the largest similarity value as the best result instead of the first returned one. The classification performance is measured as the average similarity over the complete set of 500 documents. Table 2 shows the average similarity of over the complete document set achieved under four different conditions.

From Table 2 we can see the significance of acquaintance learning using pursuit learning in achieving high quality of classifications. With dynamically-learned acquaintances, the agent can achieve 94% of optimal performance. Compared with the acquaintances of random four agents and the worst four agents, the average similarity with dynamically learned acquaintances increased about 28% and 113% respectively.

5.3. Experiments with Agent Failures

We performed a preliminary study examining the performance without the reconfiguration of acquaintances in situations involving agent failures. If a remote agent failed, it would never return any results so that its corresponding p and $\hat{d}$ values would never be updated and will remain unchanged. The requesting agent may continue sending the documents to the failed agent and hence, the performance will be degraded. With reconfiguration of acquaintances in a dynamic agent environment, the requesting agent can detect the remote agent failure or be notified of agent leaving and entering the agent society. Hence, it can update its p and $\hat{d}$ vectors and discover new acquaintances, as described in Section 4.4.

We conducted a few experiments to test the stability and performance of reconfiguration of acquaintances by intentionally eliminating some agents from the acquaintance list during the execution period. We found that after eliminating one agent from the four acquaintances of a requesting agent, the requesting agent could detect the failure after sending it 3 or 4 more documents. Eventually the requesting agent updated the p and $\hat{d}$ vectors and reconfigured the acquaintance list. We could not discern much difference in terms of average classification similarity over the test document set after eliminating one or two agents from acquaintances. In fact after eliminating three agents from the acquaintance list and successfully re-learning three new acquaintances, we obtained an average similarity of 0.438 for total 500 documents, which is very close to the value of average similarity without any failure of agents (0.461).

6. Conclusions and Future Work

In this paper we have presented a method for automatically discovering preferred acquaintance agents in a society of information agents involved in collaborative information classification. Such a selective interaction with the agents in an acquaintance list provides a powerful mechanism for scaling up agent societies to include a large number of agents. It considerably reduces the response delays and communication overhead associated with an exhaustive interaction with all agents. At the same time, it provides near optimal performance which compares favorably with that obtained with exhaustive interaction. The learning algorithm, used here to discover the best acquaintances based on past interaction experiences, is simple and efficient, and provides a model of the entire agent population as a by-product. We have also presented a method of reconfiguring the learning of acquaintances in the presence of dynamic changes in the agent society (e.g., an existing agent leaving the society voluntarily or involuntarily, as well as new agents joining the society). Experiments were performed with classification of documents from the domain of Computer Science by means of a large-scale society of classifier agents, implemented using the JAVA-RMI environment. These experimental studies indicate that the method is both practical and efficient.

The model of the agent society, as explained earlier, is a simple one, where a real number (between 0 and 1) is learned for each remote agent, indicating the latter's quality of service for the entire document stream. A possible direction of future work is to study the advantages of using more complex agent models. In particular, agents may be assumed to offer a different quality of service for different kinds of information. This requires determining a high-level abstract representation of the nature of the documents (e.g., the domain they belong to), and modeling a remote agent's quality of service as a mapping from this representation. However, even with such complex agent models, the basic approach of discovering acquaintances on-line based on past performance remains valid.

Although the motivation of learning an acquaintance list is provided in the context of an agent society involved in information classification, it is clear that the problem (and, the solution method) is general, and can be applied to any large-scale agent society where exhaustive interaction needs to be avoided. All it requires is a suitable manner of characterizing the quality of service provided by remote agents in the context of the specific task they are solving. As such, we believe that learning good acquaintances is a necessary mechanism to develop large-scale, open, and distributed agent societies.

Acknowledgment

The authors were supported by the Digital Library Initiative Phase II. This is a joint program by multiple agencies: NSF, DARPA, NASA, NLM, LoC, FBI, NARA, SI, and IMLS.

References

Baker, A.D. (1996). Market-based control, Chapter 8—Metaphor or reality: a case study where agents bid with actual costs to schedule a factory, pages 184–223. World Scientific, River Edge, NJ.

Communications of the ACM. (1994). Special issue on intelligent agents. vol 37, 7.

Decker, K., & Lesser, V. (1995). Macron: an architecture for multi-agent cooperative information gathering. Proceedings of CIKM Conference, Workshop on Intelligent Information Agents, Baltimore, MD.

Delgado, J., Ishii, N., & Ura, T. (1998). Intelligent collaborative information retrieval. Progress in Artificial Intelligence, IBERAMIA'98.

Durfee, E.H., & Montogomery, T.A. (1989). Mice: a flexible testbed for intelligent coordination experiments. Proceedings of the 1989 Distributed Artificial Intelligence Workshop, pp. 25–40.

Ferguson, I.A., & Karakoulas, G.J. (1996). Multiagent learning and adaptation in an information filtering market. In Proceedings AAAI Spring Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems, (pp. 28–32), Menlo Park, CA.

Karakoulas, G.J., & Ferguson, I.A. (1995). A computational market for information filtering in multi-dimensional spaces. In Proceedings AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval, (pp 78–83), Menlo Park, CA.

Mostafa, J., Mukhopadhyay, S., Lam, W., & Palakal, M. (1997). A multilevel approach to intelligent information filtering: model, system, and evaluation. ACM Transactions on Information Systems, 15, 368–399.

Moukas, A. (1996). Amalthaea: information discovery and filtering using a multi-agent evolving ecosystem. In Proceedings of the Conference on the Practical Application of Intelligent Agents and Multi-agent Technology, (pp. 177–186), London, UK.

Mukhopadhyay, S., & Thathachar, M. (1989). Associative learning of boolean functions. IEEE Transactions on Systems, Man and Cybernetics, 19, 1008–1015.

Raje, R., Mukhopadhyay, S., Boyles, M., Papiez, A., & Mostafa, J. (1997a). An economic framework for a web-based collaborative information classifier. Proceedings of the International Association of Science and Technology for Development SE'97 Conference, (pp. 362–366).

Raje, R., Mukhopadhyay, S., Boyles, M., Patel, N., & Mostafa, J. (1997b). D-SIFTER: a collaborative information classifier. Proceedings of the International Conference on Information, Communications, & Signal Processing, (pp. 820–824).

Raje, R., Mukhopadhyay, S., Qiao, M., Palakal, M., & Mostafa, J. (2000). Experiments with a distributed information filtering system. Proceedings of the 4th World Multi-conference on Systems, Cybernetics and Informatics, SCI-2000 (pp. 591–596).

Salampasis, M. (1995). An agent-based hypermedia model [On-line]. http://osiris.sund.ac.up/cs0msa/hyp30100.htm.

Salton, G. (1988). Automatic text processing. New York: Addison-Wesley.

Sycara, K., & Zeng, D. (1996). Coordination of multiple intelligent software agents. International Journal of Cooperative Information Systems, 5, 181–211.

Sycara, K., Decker, K., Pannu, A., Williamson, M., & Zeng, D. (1996). Distributed intelligent agents. IEEE Expert, 11(6), 36–46.

Thathachar, M., & Sastry, P. A new approach to the design of reinforcement schemes for learning automata. (1985). IEEE Transactions on Systems, Man, and Cybernetics, 15, 168–175.

Tou, J., & Gonzalez, R. (1974). Pattern Recognition Principles. New York: Addison-Wesley.

Appendix A. Qualitative Explanation of the Method (Pursuit Learning) Described in Section 4.2

This method determines the optimal action from a set of actions based on repeated trials and the outcomes of those trials. It maintains an action probability vector (denoted p, where p_i is the probability of choosing the i^{th} action). Initially, all elements of the p vector have values equal to $1/n$, where n is the number of actions, since nothing is known as to which action is good or bad. An action is probabilistically chosen according to the action probabilities, and is performed. The outcome of that action is a reward (denoted β_k) which denotes how good the action is. On the basis of the measured outcome, an estimated reward vector (denoted $\hat{d}$, where $\hat{d}_j$ is the estimated reward for the j^{th} action) is updated. The unit vector E captures the current estimate of the "optimal" action based on the $\hat{d}$ vector. The element of E that corresponds to the maximum element of $\hat{d}$, has a value of 1, and all other elements of E have zero values. The updating of the action probability vector p is carried out in such a way that it is moved by a small step towards the current E vector. As this process is carried out a large number of times, the p vector eventually converges to the optimal action, i.e., the element corresponding to the optimal action will converge to a value 1, and all other elements converge to a value 0. This probabilistic action selection and updating ensure that all actions are tried sufficiently often and the optimal action is discovered during operation. In the context of this paper, the actions correspond to all other agents who are potential acquaintances, and the optimal action correspond to the best performing acquaintance, discovered during operation.

Appendix B. Non-Technical Explanation of Some Technical Terms

Agent: An intelligent computer program that provides a service to a human user, possibly by means of collaboration with other remote agents.

Acquaintance: A remote agent, discovered through interaction, that has provided high-quality service in the past.

Thesaurus: A linear list of terms with synonym structure to describe a domain for which an agent is designed.

Document Representation: Conversion of a free-text document to a numeric or other form that can be manipulated in the computer.

Clustering: A grouping of items based on some similarity measure.

Unsupervised Learning: Discovery of good decisions by means of similarities, and not based on explicit human-provided training data.

Centroid: A representative document for a document category.

Time-out: The amount of time an agent will wait for result after requesting classification service from a remote agent. Without a maximum limit to such waiting time, an agent may wait forever if a remote agent has been deactivated or has failed.

Reinforcement Learning: A trial-and-error method of discovering the best action, based on experiences of outcomes of the trials.

Local Classifications: Those document classifications that could be performed locally by an agent for its user.

Remote Classifications: Those document classifications that are performed by a remote agent on the request of the local agent.

JAVA-RMI: Remote Method Invocation utilities for invoking remote computer programs in the JAVA computer programming language.