Assessing Usage Patterns to Improve Data Allocation via Auctions*

Rina Azoulay-Schwartz[1] Sarit Kraus[1,2]
[1]Department of Mathematics and Computer Science
Bar-Ilan University, Ramat-Gan, 52900 Israel
{schwart, sarit}@macs.biu.ac.il
[2]Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742

Abstract

The data allocation problem in incomplete information *environments consisting of self-motivated servers responding to users' queries is considered. Periodically, the servers use auctions for allocation of new data items, and for reallocation of old data items. The utility of a server from storing a data item strongly depends on the usage of the item. However, each server has information only about the past usage of the data stored locally, but does not have information about the usage of data stored elsewhere. In this paper we propose that in order to improve the behaviour of the servers in the auctions, each server learns the expected usage of data items from information about past usage of its own data items. We implemented this type of learning process using neural networks. Simulations showed that our learning methods improve the results of the bidding mechanism, and they are better than the results obtained when learning via k-nearest neighbors algorithms.*

1. Introduction

In this paper, a bidding mechanism to determine the location of data items among self motivated information servers in a distributed information system is proposed. We proffer using learning techniques by the servers in order to estimate the future usage of data by different geographical areas. This information will be used by the servers when participating in an auction, for outcome improvement.

A specific example of a distributed knowledge system is the Data and Information System component of NASA's Earth Observing System (EOSDIS) of [13], which supports archiving and distribution of data at multiple and independent data centers (called DAACs). The current policy for data allocation in NASA is static: each DAAC specializes in specific topics. When new data arrive at a DAAC, it checks if the data are relevant to its topics, and if so, it uses other criteria, such as storage cost, to decide whether or not to accept the data and store it in its database. If the data is not relevant to the DAAC, it may forward it to another DAAC whose topics seem more relevant.

In the environment that we consider, the servers are self-motivated, have no common preferences and no central controller. In addition, a server is concerned with the data stored locally, but does not have preferences concerning the exact location of data stored in remote servers. Following our previous work [16], we suggest using a bidding mechanism in order to decide about the data allocation.[1] According to our approach, the location of each data unit will be determined using a bidding mechanism, where the server bidding the higher price for obtaining the data will actually obtain it.[2] We show in [16] that this approach yields an efficient and fair solution, its implementation is simple, and the bidders are motivated to offer efficient prices.

The utility of a server from storing a dataset strongly depends on its usage by clients from different geographic areas. However, the servers are uncertain about it. We extended our model by adding a learning module to the agents that represent the servers. The agents learn about the usage of datasets by users from different areas, and use this information in the auction. We tested the effect of the learning on the outcome of the agent that learns, and on the outcome of the other agents'.

After a brief discussion of related work in Section 2, we

*This material is based upon work supported in part by the NSF under Grant No. IIS-9820657. Rina Azoulay-Schwartz is supported by the Israeli Ministry of Science.

[1]Previous work on file (data) allocation in distributed systems (e.g., [6]) considers systems where a central decision maker exists, that tries to maximize the performance of the overall system. This assumption is not valid today in many cases, when the objective is to distribute information among self-motivated servers.

[2]Only one copy of each dataset is allowed, since the datasets considered are extremely large, and it is inefficient to allow multiple copies of a dataset. However, if the datasets are small it is possible to extend the model to allow multiple copies of each dataset.

will describe the data allocation problem in the beginning of Section 3. Section 3.1 presents the utility function which characterizes the servers' preferences. In Section 3.2 we describe the bidding mechanism to be used by the servers and in Section 4 we propose a learning mechanism, based on neural networks, to be run by the servers. The learning mechanism will enable a server to estimate its utility from storing data items locally. This helps the server to bid more efficiently for these data items. Finally, in Section 5 we present simulation results on the performance of the learning method we used, and the performance of the auction protocol if the servers involved are using this learning method. In our simulations, we found that as a server learns more, it improves its own utility, while there was a minor improvement, or a minor decrease, in the other servers' utility. In addition, if all the servers learn more, the servers' average utility increases, but if there was incomplete information, excess learning could reduce the average utility.

2. Related Work

Auctions are suggested for a wide range of resource allocation problems in DAI [2, 19], since they lead to a fair, simple and efficient solution, with no need for a central authority. Vickrey auction [18] is widely used in DAI [9, 16] for the case of one-to-many auctions. This protocol is incentive compatible, which means that each bidder has incentives to bid truthfully and this is one of the reasons that we also use it. Sandholm [14] surveys the existing auction protocols, and discusses certain known and new limitations of the protocol for multi-agent systems, such as the possibility of bidder collusion and a lying auctioneer. AuctionBot [21] uses a double auction protocol for the case of multiple buyers and multiple sellers of the same item. In AuctionBot, buyers and sellers can bid on a certain item according to a multilateral competitive negotiation protocol.

Auctions can fail when applied to hard resource allocation problems, such as scheduling, [19] since there are dependencies between the resources. For such problems Clarke's mechanism can be used [7]. Its disadvantage is the requirement that the agents have to pay taxes, and this means that utility is transferred out of the system thus reducing the efficiency of the overall mechanism.

In this paper, we use the Vickrey auction mechanism for reaching agreements on data allocation among autonomous servers. Using English auctions [14] is not applicable in our environments since the servers' utilities are correlated. Thus, server A may learn from the bids of server B about its own utility, but B can exploit it and bid strategically instead of bidding its true values.

The problem considered in this paper is the information problem. How will the server learn about a dataset if it does not store it locally? Research on multi-agent learning includes [4, 20]. For space reasons, we do not elaborate on the related work here, but in Section 4 we describe relevant work on machine learning that is related to our research.

3. Environment Description

We consider an environment in which there is a set of several (more than two) information servers, denoted by SERV, connected by a communications network. The information stored in each information server is clustered in datasets.[3] Each dataset is characterized by a set of keywords and contains a large number of documents. The set of datasets in the system in a given time period is denoted by DS.

Each client retrieves information directly from the server in which it is stored, and s/he pays this server for the retrieved information. Therefore, a server is concerned only with the datasets stored in its local databases, and is indifferent to the exact location of datasets not stored locally. New datasets arrive frequently, and the usage frequency of old datasets changes over time. The servers have to determine the location in which each new dataset will be stored and they are also able to change the location of old datasets. Since the datasets considered are very large, each dataset can be stored only in one server, and it is forbidden for a server to store a dataset unless it has become its legal "owner." Conflicts among the servers may arise when two or more servers would like to store the same dataset locally.

In a centralized system, the location of each dataset will be determined so as to maximize the profits of the entire system. But a dataset allocation which is beneficial for the entire system may be non-beneficial for some of the servers. Since each server has its own interests, it will follow the centralized solution only if it is beneficial for it. Moreover, since each server has private information about the usage of some datasets, the centralized solution is not applicable: it cannot rely on the reports of the servers, since it may be beneficial for the servers to lie in their reports. Thus, any proposed protocol must be fair, consider the preferences of all the servers, and not be based on reports of the server, unless it is clear that they have no incentives to lie.

In this paper, we consider an environment where all the servers are uncertain about the future usage of all the datasets. There is also *incomplete information*, since each server knows the past usage only of the datasets stored locally, and can only partially estimate the past usage of datasets stored by other servers.

3.1 Utility Function

The following is a brief description of the structure of the utility derived by a server from storing a dataset. More de-

[3] A dataset corresponds to a *cluster* in information retrieval, and to a *file* in the file allocation problem.

tails are presented in [16]. Recall that each server receives queries from clients and answers them by sending back documents which belong to its local datasets. The clients pay the server a *query_price* per document that is retrieved as an answer to a query. We assume that there is a monetary system in the environment which is used for this payment, as well as for other payments described below.

The cost of sending a document to a client depends on the virtual *distance* between the client and the server. It is measured in terms of delivery time, which plays an important role in loaded systems in which the documents are very large (e.g., images). For simplicity, we assume that each client is located in a geographical area of one of the servers, and in order to compute the distance between server i and a client located in the geographical area of server j, we use the distance between servers i and j. The term $answer\_cost$ specifies the cost for a server providing one document over one unit of distance to a client from another area.

$Usage : SERV \times DS \mapsto R^+$ associates with each server and a dataset the number of documents belonging to this dataset which will be requested by clients located in the geographical area of that server, during one time period (e.g., one week, one month, etc.). In addition, we denote by *storage_cost* the cost of locally storing one data unit for one time period[4] and the function *dataset_size* specifies the size of each dataset in data units. Each server calculates the utility it obtains from storing a dataset, given its estimation of the expected query flows related to this dataset. Note that $storage\_cost$, $answer\_cost$, $dataset\_size$ and $distance$ are common knowledge to all the servers.

Using the above terms, the profits which server *s* expects to obtain from storing *ds* locally for one time period is

$$\begin{aligned} V_s(ds) = \; & -storage\_cost \times dataset\_size(ds) + \\ & \textstyle\sum_{s' \in SERV} (usage(s', ds) \times \\ & (query\_price - distance(s, s') \times answer\_cost)). \end{aligned}$$

In order to calculate the profits which server s expects to obtain from storing the dataset over time, we assume that there is a monetary system in which each server is able to borrow any required amount of money at the current interest rate *r*. Using this interest rate, the expected profits are evaluated as the net present value [3] of future income from queries related to this dataset. Given the interest rate r, the utility of s from storing *ds* locally from time 0 until time N is $U_s(ds) = \sum_{t=0}^{N} \frac{V_s(ds)}{(1+r)^t}$, assuming a risk neutral server. If the environment is considered to exist forever, then $N = \infty$. Note that if *ds* is not stored locally, then *s* acquires zero utility from it.

[4] For simplicity, we assume that storage space is not restricted.

3.2 The Trading Mechanism

Bidding sessions are carried out during predefined time periods. When new datasets arrive, they are stored in a temporary buffer until the next bidding session, at which time their location will be decided upon. Each server is represented by an automated agent, which participates in the bidding session. In the remainder of the paper we will use a server and its agent interchangeably.

Each server is responsible for the datasets it stores, and the initial responsibility for each new dataset is determined according to a static policy: the server with the areas of interest closest to a new dataset will be responsible for it.[5] The agents are able to borrow money at the current interest rate *r*. Thus, they do not have budget constraints.

At the beginning of a bidding session, each agent broadcasts an announcement for each new dataset it is responsible for, and also for some of its old local datasets. An announcement of the availability of dataset d by agent s, denoted as the contractor, indicates that agent s would like to *sell* dataset d.

In the next step, for each dataset d which was announced, each agent s' which is not the contractor of d, can send a *sealed bid* to contractor s. A bid contains the price which s' is willing to pay s in order to store d locally, (i.e., to *buy* the dataset.)[6] All the bids must be sent within a predefined deadline. The contractor of each dataset collects all the bids related to this dataset within the deadline.

In the third step, called the awarding step, the winner of each announcement is determined by its contractor. The winning agent of each announced dataset is the agent with the highest bid, but the price it pays will be determined according to the second highest bid.[7] As ascertained in the literature, bidding of true values is a dominant strategy in this protocol. That is, for each dataset, the best bidding strategy for each agent is to bid the price which is equal to its utility from storing this dataset, and independent of the other bids [18]. Each agent will broadcast an award message for all the announcements it made in the first step. The message should include the "winner" of the dataset, its bid value (the highest bid), the agent which sent the second highest bid, and its bid value.[8] We assume that each server sends its bid *before* it obtains the bids for its local datasets, and the awarding step is performed only after the bidding step is over for all the datasets.[9]

[5] The responsibility can be determined by the source of the information, that will direct the dataset to the server with the nearest topics

[6] If it does not want to buy this dataset, it can bid a negative price, which indicates the amount it wants to accept in order for it to agree to store this dataset locally

[7] If the second bid value for ds is less than the $U_{contructor}(ds)$ or if $|BIDDERS(ds)| \leq 1$, then the contractor stores ds locally.

[8] Specifying the agent that sent the second-highest bid in the award message prevents the auctioneer from overstating the second highest bid [14].

[9] Otherwise, if the usage of different datasets is correlated, a server may

In this paper, we assume that there is no collusion. There are two situations where collusion is possible. First, two or more servers may share information among them. In general, such a phenomenon is positive. In the best situation, two or more servers share all the information they have. In this case, we may consider them as one server during the learning process. However, it is hard to maintain such coalitions, since it may be beneficial for one server to lie to the others, concerning the past usage of its own datasets. In this paper we consider the worst case, in which there is no cooperation among servers to share information.

Second, servers may coordinate their bid prices so that the bids stay artificially low [14]. Collusion can be prevented if a high enough award is given to a server that reports attempts to form a coalition, and the involved servers will be punished. The best policy for the server, in this case, is to report the coalition formation: this is simply an N-person version of the prisoners dilemma [8].

Another issue related to the bidding protocol is how an agent should choose a beneficial set of datasets to announce for bidding. If there is no cost associated with dataset announcement and bidding, then each agent will announce all its datasets, including all its old datasets. However, the announcements can be expensive, due to costs of time and communication. Alternatively, the bidding protocol can limit the number of datasets which an agent can announce, for environments with a large number of datasets. In such cases, each agent will have to carefully select which datasets to announce. Thus, the agent has to estimate which prices it will receive as bids, and choose the datasets with the highest expected best bids. We discuss this issue in [16].

4. Learning to Estimate The Usage Of Datasets

In order to learn to accurately predict the value of a dataset, the server can use information on the history usage of this dataset, or other datasets with similar attributes (keywords). We assume that each server collects information about past usage of datasets stored in its site. Using this information, it can try to estimate the future usage of these datasets, and the future usage of other datasets, new or those stored by other servers. We also assume that each dataset is described by a vector of keywords and that the usage of a dataset is a function of the keywords it contains.

The set of all possible keywords is denoted $K = \{k_1, k_2, .., k_n\}$, and each dataset i is characterized by a subset $K_i \subseteq K$. The server would like to estimate for each agent s, the function $Usage_s : 2^K \mapsto R^+$, which associates with each possible subset of keywords, the usage of dataset i that is characterized by these keywords by clients in the area of s. Next, we describe several learning techniques which can be used for this type of learning.

The technique that we found to be the most appropriate for learning the usage function is the Multi-Layer Neural Network, using the Back-propagation Algorithm. The back-propagation algorithm learns the weights for a multi-layer network, given a network with a fixed set of units and interconnections [11]. Consider a neural network composed of several layers of identical processing elements. Each layer contains several units called *perceptrons*. A perceptron takes a vector of real-valued inputs, and calculates a linear combination of these inputs, giving a weight to each input. Our neural network contains two layers, and each perceptron of the network has a *sigmoid* unit [11]. The termination criterion we use attains the required accuracy level on the training data. We denote the maximum allowed error as ϵ. When the error of the network on the inputs is smaller than ϵ, the learning process terminates. As the value of ϵ decreases, the accuracy level increases, but the time to train the network also increases.

We chose the neural network method for learning the usage function because of its ability to learn complex structures of dependency, and to also learn real functions. The neural network also enables us to control the accuracy of the learning by choosing different values of ϵ. The weakness of the neural network is that it requires many trials to achieve good estimations. However, in our environment the time requirement is acceptable since the learning process can be run off-line by the servers, before the auction starts.

Other well known learning methods are not appropriate for data usage learning. The k-nearest neighbors algorithm [12]:8.2 is easy to implement and doesn't require a long process of learning, but does require a complex process in order to predict a value. In our experiments, the neural network based algorithm reached results better than the k-nearest neighbors algorithm, as shown in Section 5.

Reinforcement learning can be used for several learning tasks, in situations where the learner has to learn behavior through trial-and-error interactions in a dynamic environment [10]. Schaerf et al. [15] use reinforcement learning techniques for load balancing in a distributed system. They use adaptive resource selection rules that the agent applies to select a resource for a new job. Reinforcement learning [10, 12] is applicable for our learning task, if we consider the possibility of purchasing a dataset only in order to have information about its future usage. However, this situation seems unreasonable in our domain, and we have not studied it further.

Weiss and Indurkhya [20] use a rule-based method to predict real value functions. The method induces solutions from samples in the form of ordered disjunctive normal form decision rules. They also compare their approach with other techniques for function prediction, for different domains. In

update its expectations concerning a dataset usage, according to the bids for the previous datasets.

future work, we intend to compare our approach, which is based on Neural Networks, with the approach presented in [20], and with linear regression (e.g., [5]) using real data of file usage.

5. Simulation Results

In order to test the bidding technique and the effect of the learning process on the outcome of the auctions, we designed and implemented a simulation tool of the servers' environment. We, first, compared the neural network method with the k-nearest neighbors approach and found that the neural network method yields better results. Then, we compared our approach of bidding based on learning via the neural network method, to the static allocation policy currently used for data allocation by servers in distributed systems. In most of our simulations, the utilities of the servers when they used our approach were, on average, higher than the utilities of the servers that used the static allocation policy. However, if the server's approximation is far from the real future usage, then the allocation due to the auction process, may be worse on average than the static allocation. We describe our main results in the following section.

5.1 Comparison of Different Learning Methods

The neural network approach was compared with the classical k-nearest neighbors method [12]:8.2. Two variations of the k-nearest neighbors method were considered: a *nearest neighbors with similarity* method, where the similarity between two datasets is used to determine neighborhoods, and a *k nearest neighbors with distance* method, where the distance between any two datasets is used to determine neighborhoods. In this set of experiments, the servers trained their neural networks only until an accuracy of 0.1, in order to reduce the overall run time of the neural network method, and consequently to reduce the gap between it and the run time of the k-nearest neighbors methods. For both variations of the k-nearest neighbors, the neural network method achieves better results. In addition, we observed that a change in the number of datasets does not significantly change the relative improvement of the servers' utility in all the methods that we checked (the neural network, and both variations of k-nearest neighbors). However, as the percentage of the unknown keywords increases, the neural network performance improves, while the performance of the nearest neighbors algorithms remains unchanged. The graph of Figure 1 presents the change in the relative error of the three methods, as a function of the unknown keywords percentage. As the relative error decreases, the server has better knowledge about the usage of the datasets, and the outcome of the auction is, on average, larger.

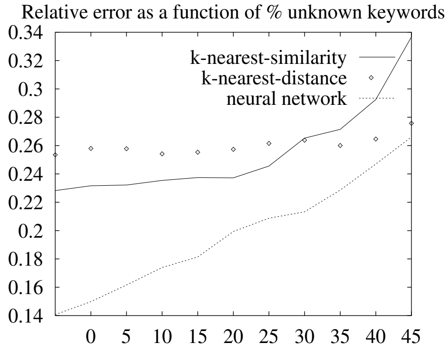

Figure 1. Changing the percentage of unknown keywords. The x axis is the average percentage of unknown keywords, and the y axis is the servers' relative estimation error.

5.2 The Effect of Learning in the Complete Information Case

We tested environments where the agents had almost complete information concerning the past usage of the old datasets. However, they were uncertain about the future usage of the old and new datasets. This was obtained by running the simulation in an environment where there were 4 servers, 100 available keywords, and 2000 old datasets. The servers didn't specialize in specific domains, but had locally stored datasets associated with all the available keywords. Thus, a server had information about all the available keywords in the environment. In our simulation, the queries are sent to the servers using a simple function of the keywords it contains. Thus, having information on all the keywords led to complete information on the usage of the old datasets.

Our first test was to check the influence of different learning levels by one server on the outcomes of all the servers. Recall that the termination criterion of our network is the level of accuracy of the network, ϵ. The value of ϵ is the sum of the errors between the network output values and the target values for its inputs. For larger ϵ's, the learning process is terminated earlier, since the sum of the errors can be larger.

In each run, we chose an arbitrary server, s, terminated its learning at several different levels (i.e., at different values of ϵ) while the learning level of all the other servers was kept fixed at 0.05. For each learning level of s, we simulated an auction, and collected the auction's results. In all our experiments, we found a large increase in the utility of server s as its learning level increases. In addition, the average utility of all the servers also increased as s's learning level increases. This increase was much smaller than the increase in s's utility. Recall that this change happens when the other servers' learning level is kept fixed. Server s is the only one who continues to learn, and this not only improves its own utility, but, quite surprisingly, also the utility of the other servers. This can be explained as follows: the initial allocation, before server s improves its own learning, was not Pareto optimal [17]. Thus, as server s learns more, it improves its own outcome, and as a side effect, the outcome of all the other agents improve as well. Example 1 demonstrates a special case of this, which actually appears in our simulation (we simplified the example by slightly modifying the utilities.). Figure 2 presents these results.[10]

Example 1 *There are 3 servers, called s_0, s_1, and s_2. The default location of dataset d is server s_0. The real utility they acquire from storing dataset d is 4, 5 and 6, respectively. However, the estimated utility is 8, 5.1 and 9.9, respectively.*

In this situation, s_0 will obtain the following bids for d: 5.1 from s_1 and 9.9 from s_2. Since the servers use the second bid auction, s_0 will decide not to sell the dataset.

Suppose now that server s_0 learns more, and estimates that its utility from d is 4.2. In this case, it will agree to sell d to server s_2, and to obtain 5.1 for it. Thus, the utility of server s_0 increases by 1.1, and the utility of s_2 increases by 4.9, (it pays 5.1 for d, and its real utility from it is 10.)

In the second test, we studied how the increase in the learning level of all the servers effects their average utility. In each run, we terminated the learning process of all the servers at several different levels, and for each level of learning, we simulated an auction, and studied its results. In this set of experiments, we found an improvement in the average utility of the servers as their learning level increases (i.e., ϵ increases.) These results also were quite unexpected given the well known observations in economics that investing in technology by all companies simultaneously doesn't improve their profits [17]. The explanation for this result is based on the fact that the initial allocation, where the servers learn until $\epsilon = 0.2$, is far from Pareto optimal, but as the servers learn more, they improve their knowledge about the environment, and they obtain results which are, by average, better for all of them.

5.3 Results for Incomplete Information Environments

Second, we checked different types of environments, where the servers had only partial information about the usage associated with different keywords by users from different geographical areas. In all our tests there were 5 servers and 2000 datasets. We ran 50 runs of each simulation where each of the servers has datasets which contain only part of the available keywords.

[10] The relationships between the utility of the learner and the average utility of the other servers depend on parameters of the simulation, e.g., the distance table and the initial allocation.

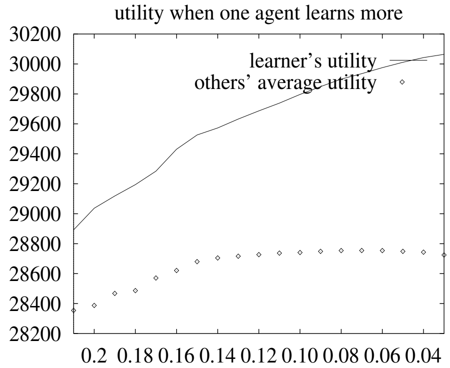

Figure 2. Complete Information Environment: one server improves its learning. The x axis shows the ϵ learning level, and the y axis shows the average utility.

The choice of the location of the old datasets and their associated keyword should be made carefully to control the incomplete information, and to provide the agents with enough samples. As the number of keywords increases, the number of samples required in order to train the network increases significantly. As studied by [11], the number of training samples required is at least five to ten times the number of weights to be trained. Baum and Haussler [1] suggest that the number of training samples will be $P > \frac{|W|}{1-a}$, where P denotes the desired number of samples, $|W|$ denotes the number of weights to be trained, and a denotes the expected accuracy on the test set.

In our case, the samples of the servers are the datasets stored locally, and the number of weights to be studied is $|W| = |K| \times |H| + |H|$ where $|H|$ is the number of nodes in the hidden layer, and $|K|$ is the number of keywords in the network. (This number includes the weights from the input nodes to the hidden nodes, and the weights from the hidden nodes to the output nodes.) Thus, as the number of keywords increases, the neural network requires more datasets, and again, the network will have information about all the available keywords.

In order to have keywords that the server will not know, we organized our simulation as follows. We provided the servers with a large set of samples (datasets) associated with only a subset of the keywords. This was done by defining for each server s a set of keywords K_s, and for each dataset ds, if ds is stored in s, the keywords of ds were only from K_s. In this manner, the set of samples can be sufficiently large, and we can control the amount of incompleteness according to the size of K_s.

We ran simulations where each server has datasets which contain only about 50% of the possible keywords. We performed 50 runs of the simulation. In each run, all the servers excluding one learned until an accuracy of $\epsilon = 0.05$. But,

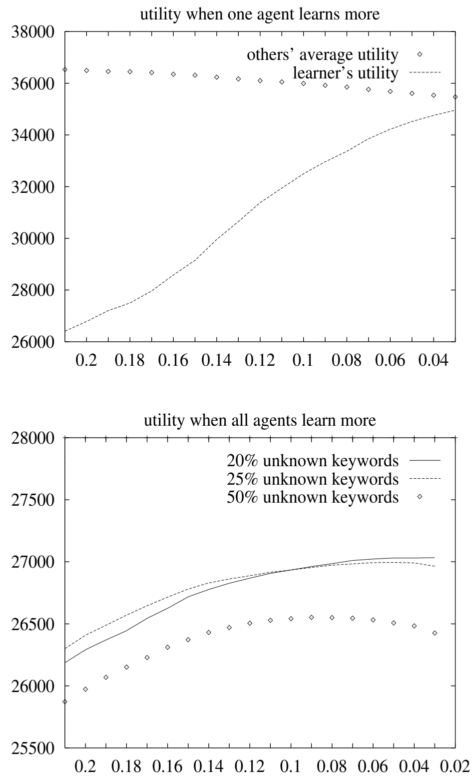

Figure 3. Incomplete Information Environment. **3-up:** one server improves its learning. **3-down:** all servers improve their learning. The x axis shows the ϵ learning level, and the y axis shows the average utility.

the *learner server*'s accuracy of the learning varied between $\epsilon = 0.2$ and $\epsilon = 0.02$, as in the complete information case.

We found that as the server learns more, the average utility of the other servers does not decrease significantly, while the utility of the learner server increases. This phenomenon occurred in all the runs. Figure 3-up demonstrates these results. Note that the results here are different than the results obtained in the complete information case in which the average utility of the other servers also increases as the learner learns more. This can be explained by the fact that in the complete information case, as the learner server learns more, it comes closer to the real utility function, thus bidding better for itself, and the results of the bidding process are closer to be Pareto optimal. However, in the incomplete information case, when the server learns more, it learns more only about some of the keywords, and on the other keywords it has inaccurate information. The other servers also have information only on some of the keywords. Thus, the auction after the learner server learns more, improves its utility less than in the complete information case, and it may also reduce the other servers' utility.

Figure 3-down demonstrates the effect of different levels of learning of all the servers on their average utility. As ϵ decreases, the servers' average utility increases. There are different curves for different levels of incomplete information: when 20%, 25%, or 50% of the keywords are unknown.

In the case of 50% of unknown keywords, the curve goes down as ϵ decreases beneath 0.08. This happens also for a smaller percentage of unknown keywords. We also checked and found that the error of the learners decreases across the line - this means that if the servers learns for a higher accuracy, their average utility may decrease, even though their knowledge of the system increases. In the case of 25% of incomplete information, the phenomena of a decrease of the utility appears, but only later in the learning process. In a complete information environment this phenomenon was not found in the average utility.

Based on the results of the complete information simulations, we expected that learning will improve the average utility of the servers, since the learning moves the servers closer to a Pareto optimal allocation. However, in incomplete information, the opposite effect could happen too, as we demonstrate in the following example.

Example 2 *There are 3 servers. The default location of dataset* d *is* s_0*, and the real utilities of servers* s_0*,*s_1 *and* s_2 *from storing* d *are* $4, 6$ *and* 5*, respectively. The estimated utilities are* $3.5, 5.8,$ *and* 3.7*, respectively, but after improving the learning, the estimated utility becomes* $4, 5.9,$ *and* 3.9*, respectively.*

Before learning more, s_0 *will sell* d *to* s_1*, and obtain 3.7. After learning more,* d *will not be sold at all. The decrease in the utility is 2. (*s_1 *loses* 2.3*, and* s_0 *earns* 0.3*.) In this case, although all the servers learn more, the total utility decreases. The reason for this is the lack of information which* s_2 *has: if it could correctly estimate its utility from the dataset,* d *could be sold to server* s_1*. In fact, this phenomenon can also occur in the complete information case because of the uncertainty, but it occurs more often, and more significantly in the incomplete information case, since the servers' estimation is less accurate.*

5.4 The Effect of Different levels of Incomplete Information

In an additional set of simulations, we studied the influence of the incomplete information level on the results of the auctions. In order to do so, we created 40 random environments. For each environment, we ran 10 runs, where the only change from one run to the other was the number of unknown keywords. In the first run, all the keywords were known to all the servers, i.e., for each keyword k and server s, s has at least one local dataset d which contains the keyword k. In each additional run, there were more unknown keywords. In all the tests, the servers learn until an accuracy of $\epsilon = 0.05$. As shown in Figure 4, we found that as the percentage of the unknown keywords increases, the average utility of the server after bidding decreases. The reason for this is that as there are more unknown keywords, the server has less information about the datasets, and thus the results of the bidding and reallocation are less beneficial.

The question of what happens for other values of ϵ arises. But, according to the results in Figure 3-down, in incomplete information cases, too small a value of ϵ may also decrease the utility of the servers. Thus, if we decrease ϵ, the difference between the curves will increase. However, it is also clear that if we choose a larger epsilon, the utility of the server where there is less information cannot be better than the utility with more information, so the difference may remain positive.

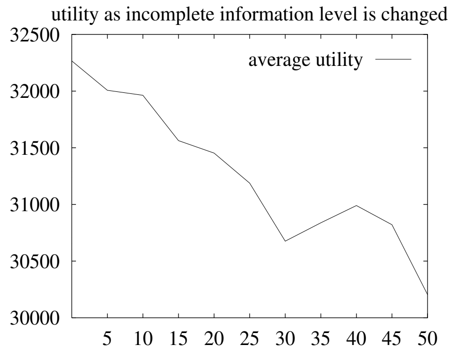

Figure 4. The average utility as a function of the percentage of unknown keywords.

6. Summary and Future Work

This paper presents a bidding protocol for the data allocation problem in multi-agent environments with incomplete information. The simulations demonstrate that the bidding protocol compiled with the learning mechanism almost always yields results better than those obtained by the static allocation policy. In addition, in almost all the cases learning improves the results of the auctions for the "learner" without significantly effecting the outcome of the other servers, and sometimes even improving the others' outcomes.

In future work, we intend to consider other auction protocols such as the English auction protocol. In the English protocol, a server may learn the value of the dataset from the bids of another server. We suggest to develop a sequential equilibria model for such systems.

It is also important to study the behaviour of the servers over time, allowing more and more reallocation processes to be performed, and testing how learning in such cases influences the outcomes of the auctions over time. The difference between multi-stage auction and one-stage auction is that in multi-stage auctions the location process of the old dataset is itself a result of an earlier auction. We would also like to consider the problem of enabling many copies of the same dataset. Here, a protocol is required in order to pay the current owners of a dataset whenever a new copy of the dataset is created.

References

[1] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[2] C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for the allocation of resources with complementarities. In *Proc. of IJCAI99*, Sweden, 1999.

[3] T. E. Copeland and J. F. Weston. *Financial Theory and Corporate Policy*. Addison-Wesley Publishing Company, 1992.

[4] K. Decker and K. Sycara. Intelligent adaptive information agents. *Information Systems*, 9:239–260, 1997.

[5] J. L. Devore. *Probability and Statistics for Engineering and Sciences*. Brooks/Cole Publishing Comp., California, 1991.

[6] X. Du and F. J. Maryanski. Data allocation in a dynamically reconfigurable environment. In *Proc. of the IEEE Fourth Int. Conf. Data Engineering*, pages 74–81, Los Angeles, 1988.

[7] E. Ephrati and J. S. Rosenschein. The Clarke tax as a consensus mechanism among automated agents. In *Proc. of AAAI-91*, pages 173–178, California, 1991.

[8] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[9] B. Huberman and S. H. Clearwater. A multi-agent system for controlling building environments. In *Proceedings of ICMAS-95*, pages 171–176, San Fransisco, 1995.

[10] L. P. Kaelbling and A. W. Moore. Reinforcement learning: A survey. *JAIR*, 4:237–285, 1996.

[11] K. Mehrotra, C. K. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. MIT Press, 1997.

[12] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[13] NASA. EOSDIS-Home-Page. http://spsosun.gsfc.nasa.gov /New_EOSDIS.html.

[14] T. W. Sandholm. Limitations of the Vickrey Auction in Computational Multiagent Systems. In *proc. of ICMAS-96*, pages 299–306, Japan, 1996.

[15] A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *JAIR*, 2:475–500, 1995.

[16] R. Schwartz and S. Kraus. Bidding mechanisms for data allocation in multi-agent environments. In M. P. Singh, A. S. Rao, and M. J. Wooldridge, editors, *Intelligent Agents IV, LNAI*, volume 1365, pages 61–75. Springer-Verlag, 1998.

[17] J. Tirole. *The Theory of Industrial Organization*. MIT press, 1988.

[18] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *J. of Finance*, 16:8–37, 1961.

[19] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. M. Mason. Some economics of market-based distributed scheduling. In *Proc. of ICDCS-98*, Amsterdam, 1998.

[20] S. M. Weiss and N. Indurkhya. Rule-based machine learning methods for functional prediction. *JAIR*, 2:383–403, 1995.

[21] P. R. Wurman, W. E. Walsh, and M. P. Wellman. Flexible double auctions for electronic commerce: Theory and implementation. *Decision Support Systems*, 24:17–27, 1998.