# Negotiation on Data Allocation in Multi-Agent Environments *

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

December 28, 1999

**Abstract**

In this paper, we consider the problem of data allocation in environments of self-motivated servers, where information servers respond to queries from users. New data items arrive frequently and have to be allocated in the distributed system. The servers have no common interests, and each server is concerned with the exact location of each of the data items. There is also no central controller. We suggest using a negotiation framework which takes into account the passage of time during the negotiation process itself. Using this negotiation mechanism, the servers have simple and stable negotiation strategies that result in efficient agreements without delays. We provide heuristics for finding the details of the strategies which depend on the specific settings of the environment and which cannot be provided to the agents in advance. We demonstrate the quality of the heuristics, using simulations. We consider situations characterized by complete, as well as incomplete, information and prove that our methods yield better results than the static allocation policy currently used for data allocation for servers in distributed systems.

*This material is based upon work supported in part by the NSF under Grant No. IRI-9423967. Rina Azoulay-Schwartz is supported by the Israeli Ministry of Science. Some material in this paper appeared in preliminary form in [52]. We would like to acknowledge the assistance of Paul H. Chan and the staff of the NASA Goddard - Distributed Active Archive Center, for their assistance in conceptualizing the distributed information servers framework aspect of this project. We also thank Jonathan Wilkenfeld for his comments and Shifra Hochberg for editorial assistance.

# Contents

## 1 Introduction

In this paper, we consider the following problem. There are several information servers, in different geographical areas. Each server stores data which has to be accessible to clients from the server's geographical area or from other areas. The topics of interest of each client change dynamically over time, and clients may arrive and disappear. New data arrives during each period and has to be located in one of the servers in the distributed system. The servers have to determine the storage location of each data item.

If a central decision maker exists, then the problem is called the "file allocation problem," which is a well-known problem in computer science and operations research (e.g., [13, 1].) We consider a similar problem for a non-cooperative environment where there is no central controller and no global utility. Each server in the environment is independent and has its own commercial interests, but would like to cooperate with the other information servers in order to make information available to its clients. Since each server has its own preferences regarding possible data allocations, its interests may conflict with the interests of some of the other servers. There is a need to find a protocol in order to reach a common agreement on the location of new data. The protocol should yield an efficient and fair allocation.

A specific example of a distributed information system is the Data and Information System component of the Earth Observing System (EOSDIS) of NASA [43]. It is a distributed system which supports archival data and distribution of data at multiple and independent data centers (called DAACs). Each data center in NASA is independent and has its own budget. Thus, we can consider the data centers as *self-interested*, although they belong to the same organization. The current policy for data allocation in NASA is static: each DAAC specializes in certain topics. When new data arrives at a DAAC, the DAAC checks if the data is relevant to one of its topics, and, if so, it uses other criteria, such as storage cost, to determine whether or not to accept the data and store it in its database. The DAAC communicates with other DAACs in those instances in which the data item encompasses the topics of multiple DAACs, or when a data item presented to one DAAC is clearly in the jurisdiction of another DAAC, and then a discussion takes place among the relevant DAAC managers. However, this approach does not take into consideration the location of the information clients, and this may cause delays and transmission costs if data items are stored far from their potential users. Moreover, this method can cause rejection of data items if they do not fall within the criteria of any DAAC, or if they fall in the criteria of a DAAC which cannot support this new product because of budgetary limitations.

We propose a strategic negotiation model that takes into account the interests of all the servers in order to agree on a data allocation. Using this protocol, the servers have simple and stable negotiation strategies that result in efficient agreements without delays. We show that our methods yield better results than the static allocation policy currently used in EOSDIS.

In Section 2, we present some related work on distributed AI and on the file allocation problem. In Section 3, we define the data allocation problem. In Section 4, we consider the general situation, in which the servers have incomplete information about each other, and we suggest using a revelation step in which each server provides its private information before the negotiation takes place. Following the revelation approach, in the remainder of the paper, we consider the complete information case only. In Section 5, we suggest a negotiation mechanism for reaching agreements on the data allocation. In Section 5.3, we prove that in the case of simultaneous response for an offer, there is a large number of possible allocations that can be reached using stable strategies. Among them, we suggest to choose the allocation which maximizes a given social welfare criterion. The problem of finding an optimal allocation with respect to a social criterion is itself a difficult issue, and in Section 5.5 we prove that it is NP-complete [22] and suggest heuristics for finding a near-optimal solution. We test and compare these algorithms in Section 6.2 in order to evaluate their performance. Finally, in Section 7 we conclude and present questions for future work.

## 2 Related Work

Our work shares common research issues in the field of distributed artificial intelligence (DAI) and in research on the file allocation problem. In the following subsections, we present related work in these areas.

### 2.1 Related Work in DAI

Research in distributed artificial intelligence (DAI) is concerned with how automated agents can interact and solve problems effectively. Bond and Gasser [4] briefly describe the research areas of DAI[1] and divide them into two main parts:

1. **DPS (Cooperative Distributed Problem Solving)**, which considers how the work of solving a particular problem can be divided among several processors. Each of the processors is intelligent, but they all have a common goal and common preferences.

2. **MA (Multi-Agent systems)**, which coordinates intelligent behavior among a collection of autonomous intelligent agents. Each one may have different goals and different interests, which may conflict with the interests of other agents in the system.

Our problem pertains to MA, since in our environment, each server has its own preferences regarding the solution which will be reached, and the interests of different agents may conflict. There are different approaches for solving coordination problems and resource allocation problems in MA environments. Often, techniques from game theory and economics are applied to MA systems, since these fields are concerned with solving similar problems among human players, assuming that each player is self-motivated.[2]

Wellman [61] uses the term market-oriented programming as an approach to distributed computation based on market price mechanisms. Several allocation problems can be solved efficiently and in a distributed fashion by using the computational economy framework and finding its competitive equilibrium: flow problems, allocation of computation time, allocation of computational resources, provision of distributed information services, etc.

Mullen and Wellman [40] have broadened Wellman's model in order to solve a problem of information service provision. In this model, a popular information service (of which the

[1]For other surveys of DAI, see [23, 15, 62, 39].

[2]For introductory books in game theory see, for example, [21, 42, 47].

canonical example is Blue-Skies, a weather-information server based at the University of Michigan) is available on the Internet, and local agents decide whether to serve as mirror sites for the service. In Mullen and Wellman's economic model, there are producers such as *carriers* that produce transmission of information through the network, *manufacturers* that provide CPU access and disk storage, and *mirror providers* that have the capability of transferring local storage and other resources into provision of the information service at their local site. A consumer is an individual end user or an aggregate of all the users at a particular site. Mullen and Wellman suggest a competitive-market pricing of the transmission price (when no mirror site is established), as well as the price of establishing mirror sites. The competitive approach is applicable only when there are several units of each type of good, since it is not rational for the consumers and producers to ignore the effect of their behavior on the prices when they actually have an influence. In our case, each data item is unique, so a competitive approach cannot be used. Furthermore, in our model, there are few servers involved, so the competitive assumption does not hold. Thus, we will use negotiations for our problem.

Ephrati and Rosenschein [17] suggest using the Clarke Tax voting procedure in order to reach a consensus in MA systems. In this procedure, each agent expresses its cardinal utility for each possibility, and the one with the highest sum of utilities is selected. Taxes are taken from the agents in order to ensure truthful voting. The mechanism assumes an explicit utility transferability. The utility that is transferred out of the system (the tax) is actually wasted and reduces the efficiency of the overall mechanism.

Negotiation is proposed in DAI as a means for agents to communicate and compromise in order to reach mutually beneficial agreements. It is used in DPS environments in order to find a mutual agreement among agents with conflict knowledge and abilities. Durfee [16] uses negotiations for decision making in distributed systems, where the issues are: usage of local resources, knowledge of the system, and task distribution. Each agent represents its knowledge by its *partial global planning*, and sends knowledge updates to the other agents using this representation.

Several researches in DPS use negotiation for distributed planning and distributed searching for possible solutions to difficult problems. Conry [7] suggests multi-stage negotiation to solve distributed constraint satisfaction problems, when no central planner exists. Our file allocation problem can be viewed as a constraint satisfaction problem, but our agents are self-motivated: hence, this DPS approach will not be used.

Sen and Durfee [55] consider two commitment strategy options for the scheduling problem: committing or non-committing to a proposed time interval amounts to either blocking or not blocking valuable calendar resources until complete agreement is reached. They checked both strategies for different combinations of environmental factors, and found that no one option is dominant over another in the sense that it performs better under all circumstances on all performance metrics.

Decker and Lesser [11] compare communicating agents versus non-communicating agents in the distributed vehicle monitoring testbed (DVMT). They found that communicating agents are better in most of the cases. They tested the average relative increase in quality in the results of communicating agents versus non-communicating agents. They found that this average grows with the load of the system, as well as with the duration power. In our research, we also checked the effect of problem variance on the impact of negotiation, and we present our results in Section 6.2.

In the case of MA, negotiation is a tool for solving conflicts among the self-interested agents (e.g., [34, 66, 49]). It differs from negotiation in DPS (e.g., [8, 7, 16, 38, 33]), which aims to solve conflicts arising from different knowledge and abilities. In suggesting negotiation as a method to solve conflicts in MA, we have to refer to the negotiation protocol - that is, what the possible actions of each agent in the negotiation are, and how they will affect the result of the negotiation. Then we have to find the equilibrium of this protocol - that is, what a stable result will be, when each agent tries to do the best it can for itself.

Rosenschein and Zlotkin [49] describe possible environments in which conflicts exist between automated agents. They suggest that the agents reveal their preferences, and they use mechanism design methods to ensure truthful reports of the agents. As in [49], we will use negotiations in order to reach a fair and efficient result, and we will suggest using a revelation method for the incomplete information situation.

Sycara [59, 58] presents a model of negotiation that combines case-based reasoning and optimization of multi-attribute utilities. In her work, agents try to influence the goals and intentions of their opponents. Sierra et al. [57] present a model of negotiation for autonomous agents to reach agreements on the provision of service by one agent to another. The model defines a range of strategies and tactics, distilled from intuition about good behavioral practice in human negotiation, that agents can employ to generate offers and evaluate proposals. In contrast, we apply a formal game-theory model.

Zeng and Sycara [65] consider negotiation in a marketing environment with a learning process in which the buyer and the seller update their beliefs about the opponent's reservation price, using the Bayesian rule. However, they use a simple model, where there are only two parties in the negotiation (buyer and seller) and where the subject of the negotiation is the price of the item.

Kraus et al. [34] present a negotiation framework based on Rubinstein's alternating-offer negotiation model [50, 46]. They describe the negotiation process for environments of task allocation and resource sharing, when the time involved in the negotiation is valuable and prove that agreements will be reached during the first or second negotiation period in various cases.

Our research is based on the research of [34] and uses the alternating-offer negotiation model. However, the problem studied here is much more complex than the resource allocation and task distribution problems described in [34]. In our model, there are more than two servers involved in the negotiation, and agreement is not simply a partition of the resource usage time or a partition of the task, but actually specifies the location of each data item.

## 2.2 File Allocation

Our research is closely related to the file allocation problem in distributed environments. The file allocation problem, suggested first in [6], deals with how to distribute files and documents among computers in order to optimize the system performance. The various file allocation models are presented as optimization problems in terms of objective functions and constraints. The file allocation problem is known to be NP-complete [18], and many contributions have been made towards solving it. Dowdy and Foster [13] have given a brief and extended description, as well as mathematical formulations of the objective function and the constraints which may be considered in the file allocation problem, and they have described some known solution techniques.

Apers [1] investigates the problem of the allocation of the data of a database to the sites of a communication network and also discusses data transfers between files. He also mentions the distributed version of the problem and suggests that, in a distributed system, the problem solving should be done in a distributed fashion and that each site should solve part of the problem.

In this paper, we consider a simple version of the file allocation problem. In the classic file allocation problem, the number of copies of each file is not assumed to be fixed, and there is difference between retrieval and update transactions: while retrieval transactions are routed to only one copy of the file, update transactions are routed to all the copies. However, in this paper only one copy of each file is permitted. Thus there is no difference between retrieval and update transactions (any transaction is called a "query") and the objective function of each server is simple and does not consider the system load.

However, in our case the servers are self-motivated, so any suggested solution has to consider each server involved, and we have to prove that the solution is actually stable in the face of manipulations by the servers. In some cases we propose that the agents find allocations that maximize social welfare. Even in this case, the problem is different than the classical file allocation problem, since our constraints with respect to server $i$ depend on the entire allocation and not only on the datasets stored locally at server $i$, as in the file allocation problem. In Section 5.5.2, we prove that the problem we consider is NP-complete. In Section 6.1, we survey more related work on the file allocation problem and describe algorithms which are used to solve different variations of the file allocation problem. We also describe why they are applicable to our problem.

## 3 Environment Description

In the environment which we consider, there are several information servers, connected by a communication network. Each server is located in a different geographical area and receives queries from clients in its area. In response to a client's query, the server sends back information stored locally or information stored by another server, which it retrieves from that server. In our model, each server has its own interests and wants to maximize its own utility. Its utility function considers the location of each data item, including data which is stored by any other server.

We suggest using negotiation involving alternating offers to solve the allocation problem. In the model of alternating offers, *all* the agents negotiate to reach an agreement which specifies the location of *all* the relevant data items. Such a process may include several iterations, until an agreement is reached. During each time period, one agent makes an offer, and each of the other agents may either accept the offer, reject it, or opt out of the negotiation. If an offer is accepted by all the agents, then the negotiation ends, and this offer is implemented. If at least one of the agents opts out of the negotiation, then a predefined *conflict_allocation* is implemented, as defined below. If no agent has chosen "Opt" but at least one of the agents has rejected the offer, the negotiation proceeds to period $t+1$, another agent makes a counter-offer, the other agents respond, and so on.

In this section, we formally describe the data allocation problem which we consider and define its basic components. First, we define the environment in which the negotiation will take place.

**Definition 3.1** *An Environment is a tuple <SERVERS, DATASETS, U>, where"*

**SERVERS:** A set of information centers. Each server is located in a different geographical area and services requests of clients that are located in its geographic area. Each server has its own interests and tries to maximize its own utility. We assume that there are more than two servers.

**DATASETS:** The set of the new data items. The concept *dataset* refers to a collection of information units, corresponding to a *file* in the file allocation problem. Each dataset has its own keywords and attributes[3]. During each period, new datasets arrive and are stored in a temporary location until a permanent decision is made about the location of each dataset. Each dataset has to be allocated to one of the servers. Only one copy of each dataset is allowed, since the datasets considered are extremely large, and it is not efficient to allow multiple copies of a dataset.[4] In environments where old datasets may be reallocated, we can extend the model and consider periodic dataset reallocation. In this case, DATASETS denote the set of the new data items, as well as the old data items which are needed to be reallocated.

**U:** An array of the utility functions of the servers: $U = \{U_s\}_{s \in SERVERS}$.
$U_s$ is the utility function of server $s$, representing its preferences. It is a function from the set of possible allocations and possible time of an agreement, to the real numbers. $U_s(alloc, t)$ specifies the utility of server $s$ from allocation $alloc$, if it was agreed at time $t$.

**Time:** Negotiation is a process that may include several equidistant iterations. We assume that agents can take actions in the negotiation only at certain times $0, 1, 2, \ldots$ that are determined in advance.

**Allocation:** An *allocation* is an assignment of each dataset to one server, i.e., a function $alloc : DATASETS \mapsto SERVERS \cup \{none\}$, indicating the server in which each dataset will be stored. If a dataset is assigned *none*, then it will not be located in the system as a whole. In environments in which each dataset has to be allocated, we have: $alloc$ : DATASETS $\mapsto$ SERVERS. The set of all possible allocations of datasets to servers, which are used as offers in the negotiation, is denoted by $OFFERS$.

[3]In on-going research, we concentrate on the structure of the dataset attributes, how does it influences its usage, and how the usage of a dataset be learned using its keywords.

[4]If the datasets are small, it is possible to extend the model to allow multiple copies of each dataset.

## 3.1 Utility in the First Time Period

The components of the utility function which we consider are often used in the distributed file allocation community [5, 13, 14]. The utility function of a server consists of payments by the user for answering their queries and for retrieval costs, answering costs, and storage costs. Suppose server $i$ receives a query from one of its clients, and in order to answer this query, server $i$ needs a document which is located in server $j \neq i$. In this case, server $j$ will send server $i$ the required document. The cost to server $i$ for retrieving the required document from server $j$ is termed retrieval costs [5]. The costs to server $j$ for sending the required document to server $i$ are called answering costs. Both answering costs and retrieval costs are assumed to depend on the virtual distance between $i$ and $j$. Each server receives a constant payment for each document which it sends in response to a query of one of its clients. (In our example, server $i$ will receive payments from the client who sent the query.) Finally, each server incurs costs for storing a dataset locally, which depends on the dataset size.

The usage of a dataset by each server influences the utility function, as follows. As the usage of a local dataset by remote clients becomes higher, the answering costs become higher. As the usage of a remote dataset by local clients becomes higher, the retrieval costs become higher. $usage : SERVERS \times DATASETS \mapsto R^+$ is a function which associates with each server and dataset the expected number of documents of this dataset which will be provided to clients in the area of this server. However, *usage* is only estimated[6], and the servers are uncertain concerning the real usage of a dataset, since the query flow is uncertain. Thus, the payments for queries are not certain. In our work, we assume that the servers are risk neutral, i.e., the utility of each server is determined only by using its expected profits [20]. The exact structure of the utility function varies for different environments. In Appendix A, we describe a specific utility function for agreements reached in the first time period. We use this function in our simulations, described in Section 6.2.

## 3.2 Cost over Time

For a server participating in the negotiation process, the time when an agreement is reached is very important, for two reasons. First, there is the cost of communication and computation time spent on the negotiation. Second, there is the loss of unused information: until an agreement is reached, new datasets cannot be used. Thus, the servers wish to reach an agreement as soon as possible, since they receive payment for answering queries. These costs can be considered in the utility function of the server in different ways [50, 46, 34].

Examples of the influence of time on the utility function are specified below. In both examples, we assume that there are fixed loses of $C$ per time period, due to the negotiation process.

**Example 3.1** *The following utility functions can be used in environments that we consider.*

[5]There are cases when clients are autonomous and send their queries directly to server $j$, which has the requested documents. For such cases, we suggest in [53] the use of a bidding mechanism for data allocation.

[6]For details, see Section 4.

**Constant discount ration:** *Consider a server $i \in SERVERS$. For $alloc \in OFFERS$ and $t \in Time$, $U_i(alloc, t) = (1-p)^t \cdot U_i(alloc, 0) - t \cdot C$, where $C > 0$ is the constant negotiation cost for each time delay, and $0 \leq p < 1$. In this example, we assume that there is a cost of unused information, but we also assume that the utility of a dataset is reduced over time, with a discount ration $1 - p$ for each time period.*

**Financial system with an interest rate $r$:** *Consider a server $i \in SERVERS$. For $alloc \in OFFERS$ and $t \in Time$, $U^i(alloc, t) = {\frac{1}{1+r}}^t \cdot U^i(alloc, 0) - C \cdot \frac{1+r}{r}(1 - {\frac{1}{1+r}}^t)$, $0 \leq r < 1, C > 0$.*

*Here we assume that each server is able to borrow or to lend money at the current interest rate* r. *Using the interest rate $r$, $U_s(alloc, t)$ is evaluated as the net present value (NPV) of the future utility of the allocation, computed with respect to the interest rate* r. *The NPV is used in financing systems in order to find the value of an investment. It is computed by discounting the cash flows at the firm's "opportunity cost" of capital [10].*

Other structures of utility functions may be considered too. However, there are some assumptions of the utility function over time (described in Section 5.1) which we will need for our analysis. These assumptions hold for a large set of functions, including the ones specified above.

## 4 Incomplete Information Environments

The problem we consider deals with allocation of data among autonomous agents, each with its private knowledge and preferences. In real world situations, if there is neither a central statistical unit nor the ability to enforce true reporting, the servers will not have complete information about each other. In this section, we will consider environments in which the agents have private information about the usage level of the datasets.

We assume that other parameters of the utility function are common to the servers in the environment. This assumption is not too strong, since the cost of storing data depends on the cost of memory, which can be assumed to be similar for different servers. Also, the cost of transmission of data from one location to the other is common knowledge and can be easily identified. We also assume that the structure of the utility function considered by different servers in the environment is the same, or, alternatively, that the utility structure of each server is known by the other servers.

Note that in all the situations which we consider, the servers are uncertain about the future usage of the datasets. In the *complete information* environment, the servers know the past usage of each dataset by the clients located in each area, but do not know the actual future usage. In the *incomplete information* environment, considered in this section, each server does not know the mean usage of each dataset by clients of each area. Each server knows only the past usage of the datasets stored locally, and it also knows the past usage of datasets by clients in its area.

There are several approaches to dealing with incomplete information decision making [32, 17, 41], but each of them has several limitations when applied to complex problems such as the data allocation problem. In [51], we describe some approaches and discuss their limitations for our domain. However, the strongest limitations are the problem of applying the protocols of these approaches to problems with a large number of possible solutions, and that of the lack of a central arbitrator or agency to run the protocol.

In this paper, we suggest using a revelation mechanism which will promise truthful reports on the part of the agents. The revelation mechanism includes the following steps. In the first step, all the agents are asked to report, simultaneously, all of their statistical information about past usage of datasets. Given this information, each server calculates the expected usage function, i.e., for each server the expected usage of each dataset for the clients around it. After this step, the negotiation will proceed as in the complete information case. In order to avoid manipulative reports in the first step, we have to ensure that when using such a revelation process no server will have an incentive to lie.

In presenting the revealing protocol and discussing its properties, we will use the notion of *local dataset* with respect to server $i$ to denote a dataset stored in server $i$. We will use the concept, *remote dataset* w.r.t. server $i$, to denote a dataset stored in another server. Furthermore, *local users* of server $i$ are the users located in its geographical area. We propose that the servers use the following protocol:

1. Each server $i$ will broadcast the following:

   (a) for each dataset, $ds$, the past usage of $ds$ by clients in the area of server $i$;

   (b) for each server, $j$, and for each local dataset $ds$ with respect to $i$, the past usage of $ds$ by clients in the area of server $j$.

   The servers send these reports simultaneously, and no communication is allowed between the servers before and during this step.

2. Each server will process the information obtained from the other servers, and then negotiation will take place, as in the case of complete information.

The following defines situations in which two servers give different reports concerning the same fact.

**Definition 4.1 Conflicting reports:** *reports of server $i$ and server $j$ are in* conflict *if there is a local dataset $ds$ w.r.t. server $i$, such that $i$'s and $j$'s reports on the usage of dataset $ds$ by server $j$ are different.*

Conflicting reports by servers $i$ and $j$ indicate that at least one of the servers is lying. In such cases, a high penalty for both servers provides an incentive for truth-telling in most of the reports. The penalty imposed on servers $i$ and $j$ should be distributed evenly among the other servers. The following lemma shows that a server will tell the truth about its own usage of remote datasets and about the usage of other servers of its local datasets.

**Lemma 4.1** *If the servers follow the pre-negotiation protocol described above, and if there is a high penalty exacted for servers with conflicting reports, then there is a Nash equilibrium wherein each server is telling the truth about its usage level of remote datasets and about other servers' usage of its local datasets.*

**Proof:**

Each server can tell the truth about its usage of each remote dataset, or say something else. Each server can tell the truth about the others' usage of each local dataset, or deviate from the truth. Suppose that all the servers always tell the truth. If server $i$ changes its report about the usage of one remote dataset, and say something else, then this difference will be revealed immediately, because the server in which the remote dataset is stored will report truthfully, and server $i$ will be penalized severely. Thus, it is not worthwhile to lie. The same can be said with respect to usage of one of its local datasets by one of the other servers. ▮

The problematic case is that of a server's report on its usage of its own local datasets, which are motivated by queries of its local users. The server is able to lie about its usage of local datasets since there are no other reports on the same facts; thus a lie will not be immediately revealed. One possibility is to ignore all reports of any server about its own usage of a local dataset. This yields inefficiency in the negotiation outcome, but it prevents manipulations by the servers.

If the servers do use the data about self usage of local datasets, a server may change its reports if it expects a higher utility when doing so. However, if it reports a lower usage of a dataset $ds$ than is true, the other servers will believe that it has lower retrieval costs when $ds$ is stored in a remote server. When using an allocation mechanism which maximizes the sum or the product of the servers' utilities, as we suggest in Section 5.3.2, such a lie may lead to an allocation in which those datasets will be stored far from the liar (since it reported a low usage), and thus the liar's utility will be lower than its utility from the allocation that might have been chosen had it told the truth.

If a server reports more usage of a dataset $ds$ than the real one, then it may cause datasets similar to $ds$ to be stored by the liar. This may cause the liar's storage and answer costs to be higher than the retrieval costs if it had been stored elsewhere (based on an honest report). Moreover, in such cases, there may be datasets similar to $ds$ which are allocated remotely according to the static allocation. In this case, if server $i$ reports heavier usage of $ds$, it causes the other servers to believe that it has higher retrieval costs of those datasets which are similar to $ds$ than it really has. Thus, the other servers believe that $i$'s utility from opting out is lower than it actually is. Therefore, the agreed upon allocation may be worse for the liar than opting out, and again, worse than the situation reached when reporting the truth.

Thus, under our assumptions[7], without knowing the exact reports of the other servers concerning their usage of their local datasets, lying may harm the server. Moreover, even if it has an estimation about the other servers' usage, it needs unreasonable computation time in order to compute which lie is beneficial, since the problem of finding an optimal allocation

[7]There can be examples where a heuristic to lie can be found easily. For example, if there is only one dataset $D$ to be allocated, and it is not worthwhile for agent $A$ to store it, then agent $A$ may report a very low usage for local datasets with topics close to $D$, in order to prevent the storage of $D$ by agent $A$ itself. However, if there is more then one dataset to be stored, then lying becomes, in general, intractable, for two main reasons. First, changing the usage of a dataset will also influence other new datasets, which the agent may want. Second, reducing the usage of $D$ will also reduces the utility of agent $A$ for the conflict allocation, a fact that may cause the new allocation to be all the worse for it.

itself is NP complete, as proved in Section 5.5.2, and the problem of finding which lie will lead to an allocation which is more beneficial than if it had told the truth is, in general, much more complicated. Thus, finding a beneficial lie is intractable.

In this paper, we also assume no collusion. Collusion can be prevented if a high enough award is given to a server which reports on attempts to form a coalition, and the involved servers will be punished. The best policy for the server, in this case, is to report about the coalition formation: this is just an N-person version of the prisoner's dilemma [42]. If collusion is not prevented, then false reports may be given.

In the remainder of this paper, we will consider complete information environments, or environments where the private data of each server was revealed during the revelation process.

## 5 The Negotiation Model

In this section, we suggest a solution method for the data allocation problem using negotiation. A solution method has to identify an allocation which will be acceptable to all the servers and that the servers will be able to implement. Note that if there is incomplete information, the negotiation is performed after the revelation step, so during the negotiation process, each server has complete information about the other servers.

We will consider two variations of the model: (1) a protocol of simultaneous response to an offer in which an agent responding to an offer is not informed of the other responses during the current negotiation period; and (2) a protocol of sequential responses in which an agent responding to an offer is informed of the responses of the preceding agents (assuming that the agents are ordered). In Section 5.1 we specify our assumptions for both variations.

### 5.1 Features of both the Sequential and Simultaneous Models

#### 5.1.1 Assumptions of both Models

As was discussed briefly in Section 3, the mechanism we use for negotiation is the alternating-offers model [46, 34]. There are $N \geq 3$ agents, where each represents one server in the negotiation.[8] The agents have to reach an agreement on the location of each $ds \in$ DATASETS. $j(t)$ will denote the agent that makes an offer at time period $t$. In each period $t \in Time$, unless the negotiation has been terminated earlier, agent $j(t)$ will offer a possible allocation for all the datasets considered (i.e., $alloc \in$ OFFERS), and each of the other agents may either accept the offer (choose Yes), reject it (choose No), or opt out of the negotiation (choose Opt). A fair and reasonable implementation is to order the servers randomly, before the negotiation begins,[9] to denote them, $1, .., |SERVERS|$, and to let $j(t)$ be $t \bmod |SERVERS| + 1$.

If a proposed offer is accepted by all the agents, then the negotiation ends, and this offer is implemented. Opting out by one of the agents ends the negotiation, and the predefined conflict allocation is implemented (see Assumption 5.2). If no agent has chosen *Opt*, but at least one has rejected the offer, the negotiation proceeds to period $t+1$, and agent $j(t+1)$

[8]In the remainder of the paper, we will use a server and its agent interchangeably.

[9]A distributed algorithm for randomly ordering the agents can be based on the methods of [2].

will make a counter offer, the other agents will respond, and so on. In some situations, there is a time period $\hat{T}$, after which the negotiation should terminate, and the conflict allocation should be implemented (finite horizon). In other situations, the negotiation may continue forever, i.e., $\hat{T} = \infty$ (infinite horizon). In the infinite horizon case, if the agents actually continue to negotiate indefinitely, we refer to the outcome as "Disagreement."[10]

The negotiation mechanism provides a framework for the negotiation process and specifies the termination condition. However, each agent needs to decide on its negotiation strategy. In general, an agent's negotiation strategy is any function from the history of the negotiation to its next move. Given a specific situation, we would like to be able to find simple strategies that can be recommended to *all* agents, such that no agent will benefit by using another strategy. Before looking into this problem, we present a number of assumptions concerning the utility functions of the agents. The first assumption states that agents prefer any agreement during any given time period over the continuation of the negotiation process indefinitely.

**Assumption 5.1 Disagreement is the worst outcome:**
*For every* $alloc \in OFFERS$, $t \in Time$, *and* $s \in SERVERS$, $U_s(alloc, t) > U_s(Disagreement)$.

As mentioned above, disagreement can occur if the negotiation continues forever, in environments where $T = \infty$. However, in an environment where there is a time period $\hat{T}$ in which negotiation ends, a predefined *conflict allocation* will be implemented. It will also be implemented if, at any step of the negotiation, any agent has decided to opt out of the negotiation. Our negotiation process will reach an agreement which will be preferred by all the servers over the *conflict utility*, i.e., the utility derived by each server from the agreed allocation will be, at the very least, the utility it can gain without negotiation. We suggest two possibilities for the conflict allocation:

**Assumption 5.2** *If negotiation has reached time* $\hat{T}$*, in the finite horizon model, or if an agent opts out, the* conflict_allocation *is implemented. It can be implemented as follows:*

**No dataset is allocated:** $\forall ds \in DATASETS, conflict\_allocation(ds) = none$. *In this case, no dataset will be allocated to any server.*

**Static allocation:** *Each dataset will be located in the server that has topics "closest" to the dataset; i.e., each dataset will be stored in the server which stores other datasets with the most similar topics. For example, a dataset about "atmosphere" will be allocated to the server who deals with this topic. The "static allocation" is well-known to the servers before the negotiation process starts and is a valid offer itself, i.e.* $static\_allocation \in OFFERS$.

The conflict allocation may be better for an agent than other offers. However, in assumption 5.3 we assume that the utility from the conflict allocation is never negative. This assumption is necessary in order to ensure that servers will prefer reaching an agreement sooner rather than later when their utility function behaves as described in Section 3.2.

[10] In fact, we will prove that since disagreement is the worst outcome, the agents will never reach this situation.

**Assumption 5.3** *The utility of each server from the conflict allocation in the first time period is always greater or equal to zero.*

If the conflict allocation means no allocation of any dataset, then the utility that the servers derive from the conflict allocation is always zero. However, we also assume that in the case of the static allocation, the utility for each server is non-negative. A non-negative utility can be achieved when the price of the queries is high enough. The following attribute states that reaching an agreement earlier is better than reaching it later, whenever the utility derived by the server from this agreement during the first time period of the negotiation is non-negative.

**Assumption 5.4 Agreements over time**
*For every* $t1, t2 \in Time$*,* $s \in SERVERS$ *and for every* $alloc \in OFFERS$ *such that* $U_s(alloc, 0) \geq 0$*, if* $t1 < t2$ *and* $t1, t2 \leq \hat{T}$ *in the finite horizon model, then* $U_s(alloc, t2) \leq U_s(alloc, t1)$*.*

The following assumption considers the relationship between the utilities of two different offers over time, if both yield non-negative utilities at time 0. It states that the relations between the utility of two offers are independent of time. This trait is important for the structure of our negotiation process.

**Assumption 5.5 Relations between offers:**
*For each* $s \in SERVERS$*,* $o1, o2 \in OFFERS$*,* $t_1, t_2 \in Time$*, such that* $t1 < t2$ *and* $t1, t2 \leq \hat{T}$ *in the finite horizon model, if* $U_s(o_1, t_1) \geq 0$ *and* $U_s(o_2, t_1) \geq 0$*, then* $U_s(o_1, t_1) > U_s(o_2, t_1)$ *iff* $U_s(o_1, t_2) > U_s(o_2, t_2)$*.*

The above assumption holds for any type of utility function where the utility of an agreement reached at time $t+1$ is a linear transformation of the utility agreed at time $t$, i.e., $U_{t+1} = aU_t + b$, where the coefficient $a$ is positive. In particular, it holds for the utility functions described in Example 3.1.

### 5.1.2 Attributes of both Models

An important property presented in the next corollary is that the utility derived from the conflict allocation is reduced over time.

**Corollary 5.1 Opting out costs over time**
*Consider a model that satisfies assumptions 5.2— 5.4. For every* $t1, t2 \in Time$ *and* $i \in SERVERS$*, if* $t1 < t2$*, and* $t1, t2 \leq \hat{T}$ *in the finite horizon model, then* $U_s(conflict\_allocation, t2) \leq U_s(conflict\_allocation, t1)$*.*

The following is a corollary of the assumptions and attributes above concerning the relationships over time between the utility of any offer and the utility of opting out.

**Corollary 5.2 Conflict allocation and other offers:**
*Consider a model that satisfies assumptions 5.2, 5.3, and 5.5. For each* $s \in SERVERS$*,* $alloc \in OFFERS$*,* $t1, t2 \in Time$*,* $t_1, t_2 \leq \hat{T}$ *in the finite horizon model,*
*if* $U_s(alloc, t1) > U_s(conflict\_allocation, t1)$*, then*
$U_s(alloc, t2) > U_s(conflict\_allocation, t2)$*.*

Since the relation between the utilities of offers and the utility of the conflict allocation does not change over time, the set of offers, which are not worse for agent $s$ than is the conflict allocation, is independent of the negotiation time. In the following, we define the set of offers which are not worse for agent $s$ than is the conflict allocation, and the set of offers which are not worse for all the agents than is conflict allocation.

**Definition 5.1 Offers that are not worse than the conflict allocation:** *For every $s \in SERVERS$, we define*

$$\overline{OFFERS}^s = \{offer | offer \in OFFERS,\ \forall 0 \le t \le \hat{T}, U_s(offer, t) \ge U_s(conflict\_allocation, t)\}.$$

*We denote by $\overline{OFFERS}$ the set of all offers that are* individually rational*, namely, the offers that are not worse than the conflict allocation is for all of the agents:*

$$\overline{OFFERS} = \bigcap_{s \in SERVERS} \overline{OFFERS}^s.$$

## 5.2 Equilibria Definition for both Models

We are now ready to consider the problem of how a rational agent will choose its negotiation strategy in such an environment, given the attributes and the assumptions specified above above. A useful notion is the Nash Equilibrium [45, 35], which is defined as follows:

**Definition 5.2 Nash Equilibrium:** *A strategy profile $F = \{f_1, ..., f_N\}$ is a* Nash equilibrium *of a model of alternating offers, if no agent $i$ has a different strategy yielding an outcome that it prefers to that generated when it chooses $f_i$, given that every other player $j$ chooses $f_j$. Briefly, no agent can profitably deviate, given the actions of the other players.*

This means, that if all the agents use the strategies specified for them in the strategy profile of the Nash equilibrium, then no agent is motivated to deviate and use another strategy. However, the use of Nash equilibrium is not an effective way of analyzing the outcomes of the models of Alternating Offers, since there may be some points in the negotiation of which one or more agents prefer to diverge from their Nash equilibrium strategies. Nash equilibrium strategies may be in equilibrium only at the beginning of the negotiation, but may be unstable in intermediate stages. In the following, we will define the concept of subgame perfect equilibrium (SPE) [47], which is a stronger concept, and which will be used in order to analyze the negotiation.

**Definition 5.3 Subgame perfect equilibrium:** *A strategy profile is a* subgame perfect equilibrium *of a model of alternating offers if the strategy profile induced in every subgame is a Nash equilibrium of that subgame.*

This means that in any step of the negotiation process, no matter what the history is, no agent is motivated to deviate and use any strategy other than that defined in the strategy profile. We will use this notion to analyze two types of negotiation protocols: negotiation where all agents respond simultaneously to an offer and negotiation where they respond sequentially.

## 5.3 Simultaneous Response Model

Consider the negotiation process, after a revelation step was performed, if required. In this section, we assume that the response process to an offer is simultaneous; each agent, when responding, is not informed of the other responses to the offer during this time period.

**Assumption 5.6** *After an agent makes an offer, all the other agents respond simultaneously.*

We will now define the strategy of an agent participating in the negotiation, when it does not know the other responses at the time it responds.

**Definition 5.4 Negotiation Strategies - Simultaneous Case***:*
*A strategy is a sequence of functions. In the simultaneous response case, the domain of the i'th element of a strategy is a sequence of offers of length i, and its range is the set* $\{Yes, No, Opt\} \cup OFFERS$.
*For each* $s \in SERVERS$ *, let* $f_s = f^t\,{}_{t=0}^{\infty}$*, such that*
*for* $s = j(t), f_s^t : OFFERS^t \mapsto OFFERS$*, and*
*for* $s \neq j(t), f_s^t : OFFERS^t \times OFFERS \mapsto \{Yes, No, Opt\}$.
*A strategy profile is a collection of strategies, one for each agent [47].*

The problem is how to implement a simultaneous response protocol. If we assume broadcasting of responses, one might receive the other responses before the server responds, so it has additional information.

The simplest method for preventing such effects is to ensure that each server sends its response before it reads the new mail it received after the time point at which the offer had been broadcast. Such checking can be done automatically by checking the data maintained by the centralized communication system. Another possibility is that each agent sends an encoded response, and will send the key to decode it only after all the other messages have been received.

### 5.3.1 Analysis of the Simultaneous Response Model

We will show that in the case of simultaneous response for any possible allocation of datasets that is preferred by all the agents over the conflict allocation, there is a *subgame-perfect equilibrium* which leads to this outcome. The proof is based on an idea of H. Haller [26]. This result can be used in other environments where there are at least three agents and where the utility functions of the agents satisfy assumptions 5.1-5.6 and attributes 5.4 and 5.5.[11]

**Theorem 5.1** *Consider a model that satisfies assumptions 5.1—5.6. If there are* $N \geq 3$ *agents, then for each offer* $x \in \overline{OFFERS}$*, i.e., for an offer which is preferred by all agents over opting out, there is a* subgame-perfect equilibrium *of the alternating offers negotiation with the outcome* $x$ *offered and unanimously accepted in period 0.*

[11]Other domains where these results can be applied include: scheduling problems for self-motivated agents, where one complete schedule should be found (e.g., [54]); complex task allocations (e.g., task assignment in self-management maintenance groups of an airline [25]); goods storage in environments of multiple suppliers; and air-pollution reduction by several companies when weather conditions require it.

**Proof:** For any time $t$ let us define a rule $R_t$ to use in time $t$ by agent $i = j(t)$ and a rule $E_t$ to be used by agents which are not $j(t)$:
For $i = j(t)$:

$R_t$: Offer $x$.

For agent $i \neq j(t)$:

$E_t$:

Accept, if $x$ was offered.
Opt out, if an offer other than $x$ was made.

These rules define a strategy $f_i$ for any agent $i$. The claim is that $(f_1, ..., f_N)$ is a subgame-perfect equilibrium with the outcome that $x$ is offered and unanimously accepted in period 0. Clearly, $(f_1, ..., f_N)$ leads to this outcome. It remains to be shown that $(f_1, ..., f_N)$ is a subgame-perfect equilibrium. Let $t \geq 0$ and $i \in SERVERS$. Consider a subgame starting in period $t$, where $i$ has to make the first move.

*Offer case:* If $i$ has to make an offer: violation of $R_t$ by $i$ leads to opting out by the other agents. Thus, the conflict allocation is implemented, but this is not better for $i$ (and probably worse) than reaching an agreement $x$. Therefore, $i$ has no better response than opting out in that case.

*Responding case:* If agent $i$ has to respond, then:

*Either* $R_t$ was violated by $j(t)$. Then, since $N \geq 3$, there is an agent $k \notin \{j(t), i\}$ who will opt out from the negotiation, according to the strategy $f_k$. Since $i$ is not decisive, deviating from opting out will not improve its outcome.

*Or* $R_t$ was followed by $j(t)$, who offered $x$. If $i$ accepts, since all other agents will accept it too, according to their strategies, the offer is accepted and $i$'s utility is $U_i(x, t)$. If $i$ opts out, the opting out solution will be implemented, but $i$ prefers $x$ over opting out.

If $i$ rejects, then it cannot get more than $U_i(x, t+1)$: if it deviates from $R_s$ at some $s > t$, it will get $U_i(opt)$. Otherwise, it will get $U_i(x, t+k)$ with $k \geq 1$. But $i$ prefers $x$ today, since it is losing over time and hence, accepting $x$ is optimal.

Therefore, following $E_t$ is optimal for $i$ in the responding case. Consequently, for any $i \in SERVERS$, following the rules is optimal in any subgame, if the other agents follow the rules. Hence, $(f_1, ..., f_N)$ is a subgame-perfect equilibrium. The same proof holds for both the finite horizon negotiation process and for the infinite one. ▮

According to the above result, the order in which the agents make offers does not influence the outcome of the negotiation. However, the above theorem shows that the number of equilibria can be very large. The selection of one equilibrium in such situations is very difficult. Since the agents are self-motivated, there is no one equilibrium that is the "best" for all the agents. In addition, one might say that the strategies in the equilibria of the theorem are strange: given an allocation $x$, the strategy of agent $i$ in the equilibrium associated with

$x$ makes $i$ opt out for any offer which is different than $x$, even though the offer is better for $i$ than opting out. However, the agent will follow this strategy if it knows that the other agents will also follow it, and given that the others will follow these strategies it cannot benefit from deviation. That is, if it is known that this equilibrium was chosen by all the agents, deviation by one of them is not beneficial.

Equilibrium selection theories have been developed in game theory, which can discriminate between Nash equilibria (e.g., [27, 29, 9]). A significant amount of work has been performed on the evolution of conventions in games that are played repeatedly within a population (e.g. [64, 63, 31, 3]). These conventions lead to a selection of one equilibrium by the agents. Another approach is using "cheap talk", which may be roughly defined as non-binding, non-payoff relevant pre-play communication [19, 30] for selecting an equilibrium.

We propose that the convention for selecting an equilibrium be agreed upon by the designers of the agents, rather than evolve from repeated interactions. It should lead to an equilibrium that is Pareto optimal. We will discuss the convention which we propose in the next section and, due to the complexity of the data allocation problem, we will propose using a "cheap talk" stage to enable the agents to start the negotiation without long delays[12].

### 5.3.2 Simultaneous Response Model: Choosing the Allocation

As has been shown, if the agents follow the negotiation protocol described above, any offer $x$, which gives each server at least its conflict utility, has a subgame perfect equilibrium which leads to the acceptance of $x$ during the first period of the negotiation. People use conventions that emerge over time, and we propose that the designers of the agents agree on a convention. Since the allocation set and the servers' utilities strongly depend on the exact settings of the negotiation session, a convention cannot be a specific equilibrium, but should be a mechanism for *choosing* one.

In particular, we propose that the designers of the servers agree in advance on a joint technique for choosing $x$ that, on the average, will give each agent a beneficial outcome. Thus, in the long run this convention is beneficial. We propose that the designers decide upon a mechanism which will find an allocation which must, at the very least, give each agent its conflict utility and, under these constraints, maximize some social welfare criterion, such as the sum of the servers' utilities, or the generalized Nash product of the servers' utilities, i.e., $\pi(U_s(x) - U_s(conflict\_alloc))$.[13] These methods will lead to Pareto optimal allocations. However, the designers may agree on other methods as well, preferably ones which will lead to Pareto-optimal outcomes. Note that the utility functions of the servers will be calculated according to the usage revealed by the servers in the revelation process. In addition to mechanisms for choosing $x$, the designers will provide their agents with the strategies of Theorem 5.1, which are in perfect equilibrium and which lead to the acceptance of a chosen allocation in the first negotiation period.[14]

[12]As a result of this equilibrium, if no agreement can be found which is in $\overline{OFFERS}$, then the conflict allocation will be agreed by the agents, during the first stage.

[13]This solution was proved by Nash as the unique solution, given some basic axioms and when the set of possible outcomes of the bargaining problem is compact and convex. In our situation, the set of allocations is not convex, but we use the generalized Nash product as a reasonable bargaining outcome.

[14]It is possible to reach results similar to ours using other negotiation protocols (e.g., [49]). However, our

The main problem in the mechanisms mentioned above is that as we will prove in Section 5.5.2, the problem of finding an allocation maximizing a welfare criterion with restrictions on the servers' utilities is NP-complete. Searching for an optimal solution is not practical in a large system and thus, a possible tractable mechanism is to search for suboptimal solutions.

If the problem can be solved in reasonable time or if there is a known deterministic algorithm which achieves good results, then each agent can run the same algorithm and find the same $x$, which will be the allocation offered and accepted during the negotiation. Nevertheless, as will be described below, we found that randomized methods may be more beneficial for all the agents than deterministic ones. However, the agents must jointly agree on the same allocation $x$, and if they use a random mechanism, each may find a different allocation.

In order to solve this problem, we suggest dividing the negotiation protocol into two phases in situations where randomized methods are beneficial. In the first phase, each server will search for an allocation, using any search algorithm and resources it has. All the agents will simultaneously broadcast their findings to the other agents at the end of the phase, and the one with the highest value of the social welfare criteria which were agreed upon by the designers will be designated as the chosen $x$. Using game-theory concepts, the exchange of messages in this phase can be referred to as "cheap talks". In the second phase, the negotiation will be performed using the perfect equilibrium strategies with respect to $x$.

We propose that in the first phase all the agents try to maximize the social welfare criterion. However, when using such a protocol, wherein each agent makes an offer and the one that maximizes a welfare criterion is chosen, it may be worthwhile for a server which has significantly more computational power than the others to deviate and try to find an offer which is primarily good for itself, and only secondarily for society. But, if the servers have similar computation powers, then their strategies which do not maximize the social welfare criterion are not stable. If we promise a bonus which is high enough for the server whose suggestion has been accepted in the first phase, then maximizing the social welfare criterion becomes the best strategy for all agents.

In any event, even if the servers do deviate, and each tries to maximize its own utility function, the results obtained are still good. In Section 6.2.3, we present the result of testing the case in which each server maximizes its own utility. We found out that the results are still not worse than the ones obtained by the static allocation.

## 5.4 Sequential Response Model

In a negotiation process, where responses are sequential, each agent responding to an offer is informed of the responses of all the agents that responded before it. We will assume that the same order as the one used for making offers is used. As in the simultaneous case, the negotiation process runs only after the revelation process has been performed and hence, we consider an environment of complete information.

protocol puts fewer restrictions on the agents' strategies, which is preferable in environments of autonomous agents. We do not restrict the length of the negotiation, and we allow each server, in turn, to propose an offer.

It is clear that an agent will choose "no" only in cases where it expects a better offer in the future and will choose "opt" only when the current offer is worse than the "conflict allocation" is and when it expects no better future outcome. The strategy profile described in Section 5.3.1 is not stable in most of the cases of sequential response: we recall from Section 5.3.1 that the strategy for each agent was to opt out if an offer other than $x$ was made. In the case of simultaneous response, we proved that this behavior is stable, but in the case of sequential response, if the agent is the last to respond and all the other agents responded "yes", it will answer "yes", too, if the offer is not worse for it than opting out. Thus, opting out is not automatically stable.

**Assumption 5.7** *After an agent makes an offer, the other agents respond sequentially; i.e., the agents are ordered, and agent $i$, when responding, will be informed of the responses of agents $1,..,i-1$ (excluding $j(t)$, if $j(t) < i$).*

Note that in sequential response environments the order in which the agents make offers may influence the outcome of the negotiations. However, the order of responses of agents will not influence their position in the negotiation. We will discuss these issues in Section 5.4.1, after providing the equilibrium of the negotiation. We will modify the definition of a strategy (Definition 5.4) in order to take into consideration the fact that each agent is informed of the responses of the servers before it.

**Definition 5.5 Negotiation Strategies - Sequential Case***:*
*A strategy is a sequence of functions. For each agent $i \in SERVERS$ , let $f_i = f^t\,{}_{t=0}^{\infty}$,*
*for $i = j(t), f_i^t : OFFERS^t \mapsto OFFERS$.*
*For $i < j(t), f_i^t : OFFERS^t \times OFFERS \times \{Yes, No, Opt\}^{i-1} \mapsto \{Yes, No, Opt\}$.*
*For $i > j(t), f_i^t : OFFERS^t \times OFFERS \times \{Yes, No, Opt\}^{i-2} \mapsto \{Yes, No, Opt\}$.*

In the case of negotiation with a finite horizon, we can use backward induction in order to find a subgame-perfect equilibrium. This will be discussed in the next section.

### 5.4.1 Assumptions for the Sequential Response Model, Bounded Case

In the following, we will analyze the situation in which responses are sequential, but where there is a finite horizon of the negotiation. This happens if the datasets to be allocated cannot be stored in the temporary buffer anymore, or because of a regulation, or because of a reallocation phase which occurs every $\hat{N}$ time period. We will rely on the property of the bounded negotiation horizon in order to use backward induction in the proof. The case of a sequential negotiation with an unbounded negotiation horizon remains for future work.

**Assumption 5.8** *There is a period during which negotiation can no longer continue. During that time period, the conflict allocation is implemented. We will denote this period by $\hat{T}$.*

We will first show that if the negotiation has not ended during prior periods, then an agreement will be reached in the period prior to that in which no further agreement can be reached, i.e., in period $\hat{T} - 1$.

**Lemma 5.1** *An agreement is reached prior to the time period during which an agreement is no longer possible.*

*Consider a model that satisfies assumptions 5.1—5.5, 5.7, and 5.8. If the agents are using their perfect equilibrium strategies, the negotiation process is not over until time $\hat{T}-1$, and it is agent i's turn to make an offer at time $\hat{T}-1$ it will then offer $o^{\hat{T}-1} \in \overline{OFFERS}$, such that for all $alloc \in \overline{OFFERS}$, $U_i(o^{\hat{T}-1}, \hat{T}-1) \geq U_i(alloc, \hat{T}-1)$, and the other agents will accept this offer.*

**Proof:**

In period $\hat{T}$ and later, there is no acceptable agreement. Therefore, the only possible outcome in period $\hat{T}$ and later is either opting out or disagreement. Since disagreement is the worst outcome (assumption 5.1), and the agents prefer opting out sooner rather than later (corollary 5.1), if the agents reach period $\hat{T}$ of the negotiation, all the agents will prefer opting out at this point. Thus, there will be at least one agent that will opt out at period $\hat{T}$. But, we know from assumption 5.8, that in period $\hat{T}-1$ $\forall alloc \in \overline{OFFERS}$, $\forall s \in SERVERS$, $U_s(alloc, \hat{T}-1) \geq U_s(Opt, \hat{T}-1)$ and that since the agents prefer opting out sooner rather than later, we can infer that $\forall s \in SERVERS$, $U_s(alloc, \hat{T}-1) \geq U_s(Opt, \hat{T})$. Thus, since opting out is the expected outcome in $\hat{T}$, the agents will accept any offer earlier. Since it is $i$'s turn, it can choose the best allocation from its point of view and make this offer, and the agents will accept it, considering that the alternative is opting out during the next period. ▮

If there are several offers with maximum utility for $i$, and $i$ is indifferent about them, a possible stable strategy is to choose the one which maximizes some social welfare criterion - such as the sum or product of the utilities. Recall from corollary 5.1 that during each time period earlier than $\hat{T}$, there is always the same set of possible agreements which are acceptable to all the agents. The agent whose turn it is to make an offer should choose the best of these agreements, according to its utility function, and make this offer. However, it needs to take into consideration the offers that the other agents may make in the future and offer an allocation which will not be worse for any of the other agents than any possible offer in the future.

### 5.4.2 Analysis of the Sequential Response Model, Bounded Case

We first define the sets of acceptable agreements by backward induction on $t$. Recall that $conflict\_allocation$ denotes the allocation which will be reached if any agent opts out, and will also be reached if time $\hat{T}$ is reached.

**Definition 5.6 Acceptable agreements:**

*Let $o^{\hat{T}} = conflict\_allocation$.*

*For every $t \in Time, t < \hat{T}$,*

*let $X^t = \{x | x \in \overline{OFFERS}$ and $\forall i \in SERVERS, U_i(x,t) \geq U_i(o^{t+1}, t+1)\}$. Let $o^t = argmax_{x \in X^t} U_{j(t)}(x,t)$. If there is more than one offer in $X^t$ which maximizes $j(t)$'s utility, then the one which maximizes the average utility of the servers will be chosen.*

This definition is sound, since $X^t$ is not empty for any time period before $\hat{T}$. We will prove this in the next lemma.

**Lemma 5.2** *If the model satisfies assumptions 5.1-5.5, 5.7 and 5.8, then for $t \in Time, t < \hat{T}$, $X^t \neq \emptyset$.*

**Proof:**
We will show by backward induction on $t$ that for all $t < \hat{T}, X^t \neq \emptyset$.

Base case ($t = \hat{T} - 1$):
It is clear from assumption 5.8 that at time $\hat{T} - 1$ there is at least one offer in which there is not a single server which is worse off than it is in the conflict allocation.

Inductive case ($t < \hat{T} - 2$):
By the inductive hypothesis, $X^{t+1} \neq \emptyset$. Therefore, $o^{t+1}$ is well defined. But, according to assumption 5.4, $U_i(o^{t+1}, t) \geq U_i(o^{t+1}, t+1)$. It is also clear from corollary 5.1 that if an offer is not worse than opting out during period $t+1$, it is also not worse than opting out during the previous period. Thus, $o^{t+1} \in X^t$. ▮

We will show now that during any time period $t$, all the agents will accept $o^t$, and the agent whose turn it is to make an offer will actually offer $o^t$. The intuition behind this is that $o^t$ is preferred by the agents over opting out, and it is not worse than any agreement that can be reached in the future.

**Lemma 5.3** *Consider a model that satisfies assumptions 5.1—5.5, 5.7, and 5.8. At any time period $t < \hat{T}$, the agents will accept any offer $o \in X^t$, and the agent whose turn it is to make an offer will offer $o^t$.*

**Proof:** The proof is by backward induction on $t$.

Base case ($t = \hat{T} - 1$): This is clear from lemma 5.1.

Inductive case ($t < \hat{T} - 2$): By the induction hypothesis, if an agreement isn't reached during this time period, the outcome of the negotiation process will be $(o^{t+1}, t+1)$. But, each agreement of $X^t$ at time period $t$ is preferred by all agents over $(o^{t+1}, t+1)$ and is not worse than opting out at $t$; the proof proceeds as in the basic case. ▮

We summarize the result of the sequential case in the following theorem.

**Theorem 5.2** *Given a model satisfying assumptions 5.1—5.5, 5.7, and 5.8, if the agents use their perfect equilibrium strategies, then during the first time period, the first agent to make an offer will suggest $o^0$, and all the other agents will accept it.*

**Proof:** This is clear from lemma 5.3. ▮

### 5.4.3 Discussion and Limitations of the Sequential Response Model

Recall from Section 5.1 that the agents are numbered 1,2,...,N, and each agent represents one server in the negotiation. This numbering determines the order of suggesting an offer, and also the order of responding to an offer, when it was suggested: Recall from Section 5.1 that at each time $t$, the suggesting agent will be $j(t) = t \ mod \ |SERVERS| + 1$. Recall from assumption 5.7 that in the sequential model, after an agent makes an offer, agent $i$, when responding, is informed of the responses of agents $1,..,i-1$.

Using the equilibrium, we have found that the order of responding to an offer does not influence the result. Agent $j(t)$ will suggest an offer which will be better for all agents than $o^{t+1}$, and there is no benefit to responding sooner rather than later. However, the order of making offers does influence the results. Agent $j(\hat{T}-1)$, which sends the offer $o^{\hat{T}-1}$ at time step $\hat{T}-1$, can suggest any offer which is better for all the other agents than the conflict allocation, and it will suggest the one which is best for itself. Agent $j(\hat{T}-2)$ can suggest any offer which is better for all agents than obtaining $o^{\hat{T}-1}$ in the next time step, and it will suggest the one which is best for itself.

It seems that if the cost of time is low, then the set $X^{\hat{T}-2}$ will be much smaller than the set of the set $X^{\hat{T}-1}$. This means that the last agent to send an offer before the conflict allocation is implemented benefits, since it can significantly improve its position when sending an offer (it can send the best offer for itself which is better for the other than a conflict), while the other agents can do much less (they have to send an offer which is better for the other than the offer which will be suggested at the next time period). Even though in the equilibrium the negotiation will not reach period $\hat{T}-1$, this property is carried out until $X^0$.

The following example demonstrates the benefits of the agent which makes an offer at $\hat{T}-1$.

**Example 5.1** *Suppose there are three agents, denoted by $s_0, s_1$ and $s_2$. Their cost over time is 1, and the utility of each agent from the conflict allocation implemented at time t, is 5-t. In the following table, the set $OFFERS$ is presented, and the utility for each agent from the acceptance of each offer at each time t is provided.*

| *offer* | *utility*$(s_1)$ | *utility*$(s_2)$ | *utility*$(s_3)$ |
|---|---|---|---|
| *A1* | *10-t* | *5-t* | *5-t* |
| *B1* | *9-t* | *6-t* | *5-t* |
| *C1* | *9-t* | *5-t* | *6-t* |
| *A2* | *5-t* | *10-t* | *5-t* |
| *B2* | *6-t* | *9-t* | *5-t* |
| *C2* | *5-t* | *9-t* | *6-t* |
| *A3* | *5-t* | *5-t* | *10-t* |
| *B3* | *6-t* | *5-t* | *9-t* |
| *C3* | *5-t* | *6-t* | *9-t* |

*For example, the utility for server $s_0$ from offer A1, if it is accepted at time 0, is 10, while the utility for server $s_1$ from the acceptance of offer A1 at time 0 is only 5. Furthermore, the utility for server $s_0$ from the acceptance of A1 at time 1 is 9, and its utility from the acceptance of A1 at time 2 is 8, etc..*

*Suppose also that $\hat{T} = 6$. This means that the last time when an offer can be made is at time period 5. So, agent $j(\hat{T}-1)$; i.e., agent $j(5)$ is the last to suggest an offer. Recall that $j(5) = s_{5\ mod\ 3+1} = s_3$.*

*At time 5, $s_3$ can suggest any offer which is better for $s_1$ and $s_2$ than the conflict allocation. Thus, agent $s_3$ will offer $A3$, which maximizes its utility, and $A3$ will be accepted immediately at time $5$, since otherwise, time $\hat{T} = 6$ will arrive, and the conflict allocation will be implemented. But agents $s_1$ and $s_2$ prefer obtaining a utility of $5-5=0$ from allocation $A_3$ at time $5$, over obtaining a utility of $5-6=-1$ from the conflict allocation at time $6$. Thus, they will agree to $A3$ at time $5$.*

*At time $4$, it is agent $s_2$'s turn to make an offer. Any offer will be accepted by agent $s_1$, since any offer gives $s_1$ a utility of at least $1$, so it will prefer it over accepting $A3$ at time $5$ with utility $0$. However, $s_2$ must suggest an offer which is at least the same for agent $s_3$ as $A3$ at time $5$; i.e., it must suggest an offer which gives agent $s_3$ at least a utility of $10-5=5$. Thus, $s_2$ can suggest $A3$ (giving $s_3$ a utility of 6), $B3$ (giving $s_3$ a utility of 5) or $C3$ (giving $s_3$ a utility of 5). Agent $s_2$ will offer $C3$, since it maximizes its utility over this third options.*

*Following the same rules, agent $1$ at time $3$ will offer $B3$, agent $3$ at time $2$ will return to $A3$, agent $2$ at time $1$ will again offer $C3$ and agent $1$ at time $0$ will offer $B3$. Finally, $B3$ will be offered and accepted at time $0$. Graphically, it can be viewed in Figure 1, Scenario 1.*

*Similarly, using backward induction, we can show that if $\hat{T} = 7$, then agent $s_1$ will be the last to make an offer, and the offer which will be suggested and accepted at time $0$ will be $A1$. We show this in Figure 1. Similarly, if $\hat{T} = 8$, then agent $s_2$ will be the last to make an offer, and the offer which will be suggested and accepted at time $0$ will be $B2$. As can be seen, the most important factor is the agent that will make an offer at time $\hat{T}-1$.*

*Similar examples can be found for other environments, with a different set of offers.*

Therefore, we suggest in Section 5.1 that the ordering of the servers be determined randomly, before the negotiation begins, so there will be no server with a constant unfair advantage over the others.

### 5.4.4 Complexity Problems for the Sequential Response Model

In the previous section, we have provided a mechanism for reaching an agreement if responses to offers are sequential and if the negotiation process has a bounded horizon. Implementing our mechanism means calculating $o^{\hat{T}-1}$, then calculating $o^{\hat{T}-2}$ and so on until $o^0$, which will be the offer of agent $j(0)$ during the first negotiation period. This agreement will be accepted by all the other servers.

However, the problem of determining $o^t$ in each time period is difficult, and in Section 5.5.2 we will prove that it is NP complete. Heuristic methods can be used in order to find an allocation which is near optimal, but there is no guarantee that the allocation found will really be $o^t$.

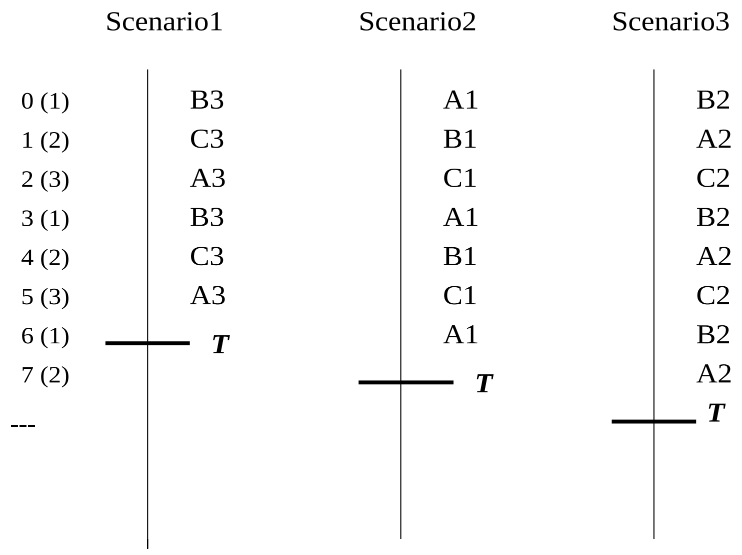

Figure 1: Three scenarios of the backward induction process. They differ in the agent which makes an offer during the last negotiation time period: Scenario 1 happens if agent 1 makes an offer at time $\hat{T}-1$. Scenario 2 happens if agent 2 makes an offer at time $\hat{T}-1$, and Scenario 3 happens if agent 3 makes an offer at time $\hat{T}-1$.

If we allow each agent to use any heuristic method, or if the heuristic method used is nondeterministic, then the backward induction will not work, since determining $o^{t-1}$ is impossible without exact knowledge about $o^t$, and nobody can know $o^t$ without knowing how it is going to be determined.

We should consider the protocol in which each agent $j(t)$ is asked to report to all the other agents its $o^t$. First, agent $j(\hat{T}-1)$ will be asked to report its suggestion at time $\hat{T}-1$. Then, the agent before it is required to determine $o^{\hat{T}-2}$, according to $o^{\hat{T}-1}$ which had already been reported, and so on, until agent $j(0)$ is asked to calculate $o^0$. In that point, we do not allow the agents to suggest anything different than what they had previously reported.

But, there are cases in which it is worthwhile for the agent to report an allocation other than the one which is best, from its point of view, as can be seen in example 5.2. So, deciding which allocation to report may become a complicated decision.

**Example 5.2** *There are three possible allocations: A,B, and C. The negotiation has two time periods before time $\hat{T}$ during which offers can be made. We denote them by $\hat{T}-2$ (the first one) and $\hat{T}-1$ (the last one). The constant cost of delay for one period is 10. There are three agents, whose utilities from A,B, and C during the first time period are as follows:*

| | *agent 1* | *agent 2* | *agent 3* |
|---|---|---|---|
| *A* | *100* | *45* | *100* |
| *B* | *60* | *50* | *80* |
| *C* | *110* | *40* | *80* |

*Suppose also that agent 2 knows all of the information in this table, but agent 1 only* *knows the information in its own column and that of agent 3. Suppose at time $\hat{T} - 1$ it is agent 2's turn to make an offer, and agent 1 makes an offer at time $\hat{T} - 2$. Thus, at time $\hat{T} - 1$, $o^{\hat{T}-1} = B$, since it is the best allocation for agent 2 in this period. Then, at time $\hat{T} - 2$ it is the turn of agent 1 to make an offer. It must find an offer which gives agent 2 at least 40, and agent 3 at least 70, so they will have no incentive to reject its offer in order to receive B during the next time period. The offer that follows this constraint, which is best for agent 1, is C. Thus, $o^{\hat{T}-2} = C$. But, if agent 2 reports that it will offer A, then agent 1 in the period before cannot offer C during the period before, since it is not good enough for agent 3; i.e., it is not better for agent 3 than obtaining A during the next time period. Thus, agent 1 will be forced to offer A as well. Agent 2 prefers A over C. Thus, it will report A instead of B, although B gives it a higher utility.*

The example above demonstrates that, given incomplete information concerning the utility of possible allocations, the agents may benefit from deviating from reporting the best allocation they can suggest when it is their turn. However, it is possible to force the agents to follow their report and to offer the allocation they declared they would suggest. This does not prevent manipulation, but it does promise an agreement during the first time period of the negotiation. Furthermore, in order to send a manipulated report, the agent needs enough time by itself in order to fabricate a report which will achieve better results than those resulting from simply reporting the best response. This process is much more complex than reporting the best found allocation, which is itself intractable.

It is not stable to enforce a regulation in which all the agents use the same deterministic algorithm with the same amount of resources in order to find $o^T$. This non-stability occurs because each agent has incentives to deviate and to use a better algorithm with more resources, in order to find an offer which will be better for itself, as well as for the other agents, so that it will be accepted by them. A one-step negotiation protocol which ignores the problem of the bounded abilities may cause delays in the negotiation. For example, if agent $j(t_1)$ underestimates $o^{t_2}$, and suggests an offer which is worse for agent $j(t_2)$ than $o^{t_2}$ at time $t_2$, then agent $j(t_2)$ prefers to reject the offer. However, if an agent rejects an offer, it takes a risk that its offer will also be rejected by some other agent, which will thinks more and finds a better allocation for itself. Thus, it cannot ensure benefits from rejecting the offer.

To summarize, we found that using a sequential responses protocol of negotiation if the problem size is large may cause different problems, since we cannot force the agents to use a specific non-deterministic optimization model, and without doing so, negotiation may result in delays. In the rest of the paper, we will continue with the simultaneous responses protocol. However, in the following section, we prove the NP-completeness of both finding an offer in the simultaneous responses negotiation (SBDA and PBDA), and finding an offer in the sequential responses negotiation (ABDA).

## 5.5 Complexity of the Allocation Problem for Both Models

### 5.5.1 The Optimization Problem

Consider the negotiation process, after the revelation stage described in Section 4. In Section 5.3, we proved that in the case of a simultaneous response to an offer, any offer which is not worse for any of the servers than the conflict allocation can be implemented. We suggest choosing an allocation which maximizes a given social criterion.

In theorem 5.2 of Section 5.4, we presented the perfect equilibrium strategies of the agents, in the case of sequential response and finite horizon, and we proved that at time 0, agent $j(0)$ will offer $o^0$. Recall from 5.6 that $o^{\hat{T}} = conflict\_allocation$, and for each time period $t \leq \hat{T} - 1$, $o^t$ is the best offer for $j(t)$, which is better for all the other agents than $o^{t+1}$ at time $t+1$.

In both cases, the agent has to find an allocation, given some constraints on the utility of all the agents: in the simultaneous response protocol, the allocation should be better for all the agents than is the conflict allocation, and in the case of sequential response, the allocation should be better for all the agents than the offer during the next time period. In the following, we will present the formal problem of finding an appropriate allocation. We will use the term $guaranteed\_utility_s$ to denote the utility which any selected offer should guarantee for server $s$. In the case of simultaneous response, $guaranteed\_utility_s$ is the utility for server $s$ from the $conflict\_allocation$, since, if it will not derive at least the same utility, it will prefer to opt out instead of accepting the offer. In the case of sequential response and finite horizon, $guaranteed\_utility_s$ in time period $t$ is the utility which $s$ derives from $o^{t+1}$ at time $t+1$, since it will reject a lower offer in order to get $o^{t+1}$ during time $t+1$.

In order to find a specific offer to make in the negotiation process, during time period $t$, an agent has to find the best allocation according to a utility function f, which needs to be individual-rational:

$$Pos_t = \{alloc | alloc \in OFFERS,\ and\ \forall s \in SERVERS, U_s(alloc, t) > guaranteed\_utility_s\}$$
$$alloo^t = argmax_{alloc \in Pos_t} f(alloc)$$

Function $f$ can be a social welfare function, or the utility of one of the agents, e.g., the one to make an offer in the negotiation process. The exact structure of $f$ depends on the specific model and equilibrium. In particular, we consider the following possible functions:

- Utility of one server - $U_s(alloc, t)$. This is applicable in the case of sequential response and finite horizon, or when an agent deviates in the simultaneous response environment.
- The sum of the servers' utilities - $\sum U_s(alloc, t)$ - can be used in the case of simultaneous response as a method to find an offer. In particular, we are looking for $alloo^0$, i.e., $\sum U_s(alloc, 0)$ can be used.
- The generalized Nash product of the servers' utilities

  $$\pi(U_s(alloc, t) - U_s(conflict\_allocation, t))$$

  for the case of simultaneous response. Similarly, the relevant case is when t=0.

In the following, we will prove that the problem described above is NP-complete, and we will suggest heuristic and sub-optimal approaches for solving it.

### 5.5.2 NP-Completeness

In this section, we will define the decision versions of the problems described above and prove that they are NP complete.

**ABDA: Agent Best Dataset Allocation:** the decision version of this maximization problem is:

**Instance:** the set $SERVERS$, an agent $s \in SERVERS$, number of datasets $n \in N$, the utility function $U$, time period $t \in Time$, $\{guaranteed\_utility_s\}$ for each server $s$, and $C \in R$.

**Question:** find out whether there is an allocation $x$ in which each server $s \in SERVERS$ gets at least $guaranteed\_utility_s$, and $U_s(x,t) \geq C$.

**SBDA: Sum Best Dataset Allocation:** the decision version of this maximization problem is:

**Instance:** the set $SERVERS$, number of datasets $n \in N$, the utility function $U$, $\{guaranteed\_utility_s\}$, and $C \in R$.

**Question:** find out whether there is an allocation $x$ where each server $s \in SERVERS$ gets at least $guaranteed\_utility_s$, and the value of the sum of utilities $\sum_{s \in SERVERS} U_s(x,0)$ is greater than $C$.

**PBDA: Product Best Dataset Allocation:** the decision version of this maximization problem is:

**Instance:** the set $SERVERS$, number of datasets $n \in N$, the utility function $U$, $\{guaranteed\_utility_s\}$, and $C \in R$.

**Question:** find out whether there is an allocation $x$ where each server $i \in SERVERS$ gets at least $guaranteed\_utility_s$, and the value of the product $\Pi_{s \in SERVERS}(U_s(x,0) - guaranteed\_utility_s)$ is greater than $C$.

The decision problem can be no harder than the corresponding optimization problem. Clearly, if we could find an allocation which maximizes the sum of utilities in polynomial time, then we could also solve the associated decision problem in polynomial time. All we need to do is to find the maximum utility and compare it to $C$. Thus, if we could demonstrate that SBDA is NP-complete, we would know that the original optimization problem is at least as difficult and that the same holds for the two other problems.

**Theorem 5.3** *ABDA, SBDA, and PBDA are NP complete.*

**Proof:** ABDA, SBDA, and PBDA are in NP: when a solution $x$ is given, it can be checked in polynomial time to ensure that it is better for all the agents than the initial allocation, and that the sum, or product, is higher than the required value $C$. In order to prove that they are NP-hard, we can show a reduction from the multiprocess scheduling. The Multiprocessor Scheduling problem (MS) is defined in [22] as follows:

**Instance:** A finite set $A$ of $n$ tasks, a length $l(a) \in Z^+$ for each task $a \in A$, a number $m \in Z^+$ of processors, and a deadline $D \in Z^+$.

**Question:** Is there a partition $A = A_1 \cup A_2 \cup ... \cup A_m$ of $A$ into $m$ disjoint sets such that $max_{1 \leq s \leq m}\{\sum_{a \in A_s} l(a)\} \leq D$ ?

The problem is NP-complete, and its proof is immediate from the Partition Problem [22]. The reduction from MS to ABDA, SBDA, and PBDA is as follows. Each server corresponds to one processor, and each dataset corresponds to one task. The utility function for server $s$ from each dataset location will be: $D/n - l(a)$ when the dataset is allocated to it, $D/n$ if it is allocated to another server, and $-D$ if it is not allocated at all. Furthermore, for each server $s$, $guaranteed\_utility_s = 0$. Finally, we can take $C$ to be 0 for all of the maximization cases.

If there is an allocation which provides for each server, at the very least, its conflict utility, then each server has at least a utility of 0. This means that all the datasets are allocated, and for each server $s$, $\sum_{alloc(a)=s} l(a) \leq D$, since otherwise the utility of $i$ would be negative.

If there is an allocation of the tasks, then the same allocation will be feasible for ABDA, SBDA, and PBDA, since the utility for each server will be, at most, 0, but all the datasets are allocated. Thus, the sum of $l(a)$ is at most D, so the sum for all datasets will be $n \cdot D/n - \sum_{a \in A_s} l(a)$, and $\sum_{a \in A_s} l(a)$ is at most $D$ if the task allocation solution is feasible. ▮

Since the problem is NP-Complete, we suggest the following heuristic algorithms for finding sub-optimal solutions for the problem.

## 6 Heuristic Methods and Simulation Results for the Simultaneous Response Model

In the previous section, we proved the NP-completeness of maximizing the sum, the product, or the utility of one of the servers under the constraints on the utilities of each of the agents. Hence, obtaining an optimal allocation of datasets to servers is not computationally feasible. Thus, we suggest using sub-optimal methods. In the next section we will discuss methods used in the literature of the file allocation problem, and then we will propose heuristic algorithms that can be used for finding sub-optimal solutions for solving each of the problems above.

### 6.1 Related Algorithms and Heuristic Methods

The file allocation problem has different variations [5, 1, 14], but most of them deal with maximizing a global function, under storage limitations. That is, the goal is to optimize the performance of the distributed system as a whole, but it is assumed that the amount of memory which can be effectively installed at a single node is limited [5]. The problem that we have is different. We have a global function to maximize, as in the classic file allocation problem. However, the constraints associated with server $i$ are not related only to the files located at server $i$, but also to all the files in the system. That is, we don't assume constraints on the storage space, but we have constraints on the utility of each server, which is a function of the overall allocation. Thus, we cannot automatically adopt the solutions of the file allocation problem as solutions of our problem.

Ceri et al. [5] suggest a method for the solution to the file allocation problem, which is based on the isomorphism to the multiple choice constraint knapsack problem. The method is based on an operations research method for solving the knapsack problem.

Our problem is equivalent to another version of the Knapsack problem, called *the multi-dimensional 0/1 Knapsack problem* [60]. However, the algorithms presented in [60] that achieve the optimal solution, are intractable.

The classical approach in operations research [28] to solve problems such as the different variations of the 0/1 knapsack problems is by a branch and bound algorithm. The assignment of an object to a knapsack is considered a binary variable, and the branch and bound algorithm assigns a value for each binary variable, and backtracks when required.

In our problem, we may consider $x_{i,j}$ to be a boolean variable, indicating that dataset $i$ is allocated at server $j$. However, in the variation we studied, only one copy of a dataset is allowed. Thus, if $x_{i,j} = 1$, then $x_{i,k} = 0$ for each $k \neq j$. This makes the branch and bound algorithm simpler, since whenever 1 is assigned to one of the variables, all the variables related to the same dataset are cleared. Alternatively, we may use a numerical variable, indicating for each dataset the number of the server where it is assigned, and run a search algorithm with backtracking in order to find the assignment for each variable. We used this approach, and in Section 6.1.1 we describe the exact structure of the backtracking algorithm we used.

Another approach to solving the file allocation problem is using a greedy algorithm. In the approach taken by Due and Maryanski [14], a distributed candidate selection algorithm using the historic data from each server is proposed. An allocation algorithm is then run to choose the optimal assignment using heuristic benefit functions and a greedy search strategy. However, the algorithm is based on the fact that the constraints are related to storage limitations, and that there is a minimal number of copies for each file. In this case, deleting and then adding a file will usually not violate the storage constraints, while in our case, moving a file from one location to another may violate the constraints of a certain server.

Another approach has been taken by Siegelmann [56], who used genetic algorithms for the *multiprocessor document allocation problem*, and by March [36] for the case of database design. Also, in our work, we have implemented a genetic algorithm and compared its outcome to other approaches. The details of the algorithm we used are presented in Section 6.1.4.

Another point of view of the problem is to consider it as a constraint satisfaction problem (CSP) and solve it using constraint satisfaction methods. The classic algorithm for solving CSP problems is a tree search algorithm with backtracking [12], and several improvements have been made to make the search more efficient [48, 37]. Our problem can be considered a CSP, but it has a special structure, since each constraint involves all of the variables. This made some of the improvements to the basic backtracking algorithm irrelevant for our model. In our simulations, we tested the behavior of a backtracking algorithm and compared it with the other methods we used for large problems.

Another approach for solving CSP is using a min-conflict hill-climbing algorithm [37]. This approach was developed for binary constraints satisfaction problems, where each constraint involved at most two variables. It starts with a random assignment of values to the variables. Then it selects one variable whose value is in conflict with the value of other variables and assigns it a different value that minimizes the number of variables which are in conflict with the chosen variable. We have adapted this algorithm for our problem, where each constraint involves all the servers, and have tested it on different configurations. For details, see Section 6.1.3.

In summary, there are several different methods for solving problems similar to ours. We have considered three different approaches: backtracking, the genetic algorithm, and hill-climbing. There are no results of simulations for problems identical to ours that are discussed in the literature, so we cannot compare our detailed results to others. However, the algorithms we used are adapted from algorithms for similar problems, as described above. In the following, we will describe in detail the different algorithms used and will then provide the simulation results. We have also checked the behavior of the algorithms for different sizes of problems, and different values of parameters. The main question is whether our results can be used when trying to maximize other utility functions. This seems plausible if the utility function is additive, as ours is. However, more research is required in order to test the algorithms for different structures of utility functions, e.g., if the utility function is not additive, or with other parameters of the utility function.

### 6.1.1 Backtracking Search

The backtracking search algorithm will sequentially decide on the allocation of each dataset. Given a partial assignment of a set of datasets and a dataset which should be assigned next, the algorithm will look for a feasible assignment, i.e., an assignment which satisfies the constraint that each agent will obtain more than its *guaranteed_utility*. If there is no feasible assignment for the dataset to be assigned, then the algorithm will backtrack and try another allocation for the previous dataset. After finding a solution, the search can backtrack and try to find a better solution. If no more solutions can be found, then the optimal result is reached. This method is a simple search algorithm [48]. The assignment which will be developed is the one with the highest value of the social welfare criterion to be maximized. The algorithm can be found in [51].

**Lemma 6.1** *In the worst case scenario, the backtracking algorithm will need exponential time in the number of the datasets to be allocated in order to find the first solution.*

**Proof:**
The run of the algorithm can be viewed as searching a tree, as is demonstrated in Figure 2. Each level corresponds to one dataset. A path in the tree from the root to the leaves means an allocation of all the datasets. In the worst case, all the allocations are examined for all the datasets. In this case, the number of combinations examined are $N^M$ when $M$ is the

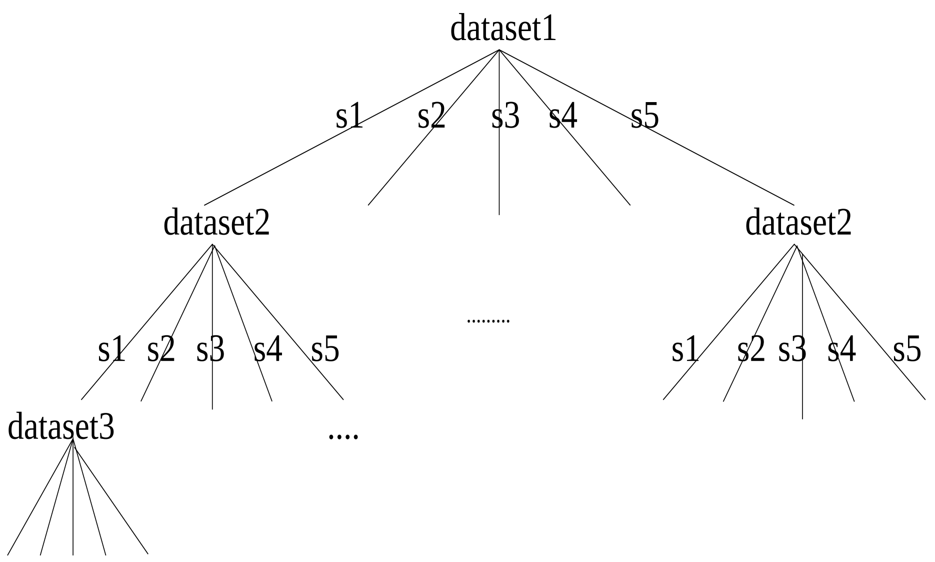

Figure 2: Search space of the allocation problem.

number of datasets, and $N$ is the number of servers. This grows exponentially as $M$ grows. ∎

### 6.1.2 Backtracking on Sub-Problems

Since the backtracking algorithm is exponential in the number of the datasets, we suggest running the algorithm on sub-problems and merging the solution into one allocation. That is, if there are $M$ datasets to allocate, we suggest arbitrarily dividing them into $L$ subsets and running the backtracking algorithm for each subgroup separately. In order to run this, we assume that the input consists of a specific allocation (e.g., the conflict allocation) and that the resulting allocation must achieve a higher utility for each agent than the given allocation. The algorithm can be found in [51]. In our simulations, we found that when there is a time constraint, this algorithm achieves better results than the regular backtracking algorithm does. However, the hill-climbing algorithm (see Section 6.1.3), which is non-deterministic, could achieve even better results. We propose using back-tracking on sub-problems of large problems, when a deterministic algorithm has to be used.

### 6.1.3 Hill-Climbing

We use the method of 'random restart hill-climbing' [37]. The hill-climbing search starts with an arbitrary allocation and attempts to improve it, i.e., it tries to change assignments of datasets in order to satisfy more constraints, until there is no possibility of improving the solution.

The random-restart hill-climbing conducts a series of hill-climbing searches from randomly generated initial states, running each until it halts or until it makes no significant progress. It saves the best result found so far from among all of the searches. There are several stop criteria which can be used, such as running a fixed number of iterations, or running until the best saved result has not been improved for a certain number of iterations. We run the algorithm for fixed CPU time units, in order to be able to compare its results with the results of the other algorithms we have implemented. The algorithm is presented in [51].

### 6.1.4 The Genetic Algorithm

In the following, we suggest using a genetic algorithm [24] for the data allocation problem. A genetic algorithm was proposed in [56] for the multiprocessor document allocation problem, and it was used in [36] for allocating data and operations in a distributed database. We have implemented a genetic algorithm for the problem of datasets allocation in a multiple agent system.

The simulated population contains a set of individuals. Each individual has a data structure that describes its genetic structure. Given a population of individuals corresponding to one generation, the algorithm simulates natural selection and reproduction to obtain the next generation. The algorithm involves three basic operations: *reproduction*, in which individuals from one generation are selected for the next generation; *crossover*, in which genetic material from one individual is exchanged with the genetic material of another individual; and *mutation*, in which the genetic material is altered. All three operations make use of randomization.

In our simulation, each individual represents a possible allocation of the datasets to the servers. After a new generation is created, an evaluation process finds the value of each individual. In our algorithm, we evaluate the value of an individual to be a combination of the value of the objective function and the value of the violated hard constraints.[15] After this phase, a fitness value is determined for each individual, according to its value. The fitness value we used for each individual was $\frac{1}{1+error}$, where $error$ is the difference between the value of the best individual in this generation and the value of this individual.

The reproduction phase creates a new generation by reproducing individuals from the previous generation, performing a cross-over phase on other individuals, and mutating individuals with a low probability. First, two random individuals are chosen, with a high probability for those which achieve a high fitness. Then, the algorithm decides randomly whether or not to copy the individuals from the last generation, or to perform the cross-over phase, which is done by randomly replacing locations of datasets between the two individuals. Finally, a mutation phase is performed with a low probability, causing a small random change in the individuals. The algorithm is presented in [51].

As mentioned above, the value of each individual is calculated by combining the objective function value and the value of the violated hard constraints in this individual. In our case, we assigned a high weight to the value of the hard constraints when calculating the value of an individual. However, we found that the preferred values for the hard constraints, as well as the best value of other parameters of the genetic algorithm — the number of individuals in each generation, the probability of the cross-over phase, the probability of the mutation

[15]The hard constraints of an allocation are the constraints on the utility of each server: the utility of each server should be, at least, its *guaranteed_utility*.

phase — depend heavily on the domain parameters. This prevents the genetic algorithm from being the preferred algorithm for the automated negotiation used by the agents, since its performance depends on parameters which cannot be chosen in advance by the designers of each possible environment.

## 6.2 Simulation Evaluation

The backtracking algorithm, the genetic algorithm, and the hill-climbing algorithm were implemented in C++ and tested on a Solaris machine. The environment's attributes, including the usage frequency of datasets by different areas, the distance between any two servers, etc., were randomly generated. We implemented the conflict allocation as the static allocation; i.e., each dataset is stored by the server which contains datasets with similar topics. However, we did not take into consideration the 'none' option; the static allocation, as well as the allocation algorithms, ensures that each dataset will be located to a server.

We implemented the utility function that is defined in Appendix A. In this function, the utility from the assignment of one dataset is independent of the assignment of the other datasets, and it satisfies all the assumptions and attributes of Section 5.

In most of the simulations, we used a measurement which excludes the gains from queries and the storage costs, since their influence on the sum of the servers' utilities does not depend on a specific allocation. In particular, we denote by *vcosts*(*alloc*) the variable costs of an allocation, which consists of the transmission costs due to the flow of queries. Formally, given an allocation, its variable costs are defined as follows:

$$\begin{aligned} &vcosts(alloc) = \\ &\quad \sum_{ds \in \text{DATASETS}} \sum_{s \in \text{SERVERS}} usage(s, ds) \cdot distance(s, alloc(ds)) \cdot \\ &\qquad (answer\_cost + retreive\_cost). \end{aligned}$$

The actual measurement which we use is denoted by *vcost_ratio*, the ratio of the variable costs when using negotiation and the variable costs when using the static allocation mechanism. The efficiency of the negotiation technique increases as the *vcost_ratio* decreases. We use *vcost_ratio* as a measurement instead of the sum of the utility function values in order to ignore the influence of irrelevant parameters on the utility, such as the money obtained for answering queries, and the storage costs, since these parameters are the same for different locations. It is clear that given the same parameters, as *vcost_ratio* decreases, the average utility of the servers increases.

We ran simulations on various configurations of the environment in order to demonstrate the applicability of our techniques in various settings. The number of servers was 11, since this is the current number of servers in NASA, but we also considered environments where the number of servers varied between 3-14. The number of datasets to be allocated varied between 20 and 200. We assumed simultaneous response, and in these environments, the exact cost over time has no influence on the results. All the outcomes that are presented are the average of the results which were obtained from runs of randomly generated environments according to the specification of the specific simulation, as described in the next sections.

### 6.2.1 Performance of the Algorithms

The purpose of the first set of tests was to compare the performance of the proposed algorithms. First, we ran simulations with randomly generated environments, with 11 servers and 20-200 datasets to allocate, for a limited period of time, where the time limit grows linearly as the number of datasets grows. All of the algorithms attempted to maximize the sum of the servers' utilities. Figure 3 presents the vcost_ratio as a function of the number of datasets for the different algorithms used. Note that the performance of an algorithm improves as the vcost_ratio decreases. For all the configurations which were considered, the backtracking algorithm was not able to find any allocation which is better than the conflict_allocation; thus the vcost_ratio was equal to one for all these cases. The graphs of the figure specify the behavior of backtracking on the sub-problems algorithm, the hill-climbing algorithm and the genetic algorithms. We found that the hill-climbing algorithm achieved significantly better results than the other algorithms used.

We used a paired t-test to test the difference between the vcost-ratio of the *backtracking-on-sub-problems* algorithm and the vcost-ratio of the *hill-climbing* algorithm. In the same way, we tested the difference between the vcost-ratio of the genetic-algorithm and the vcost-ratio of the *hill-climbing* algorithm. In both cases we found that the vcost-ratio of the *hill-climbing* was significantly smaller, for any significance level greater than 0.01. In the first test, t-statistic was 11.15658. In the second test, t-statistic was 11.15658. In both cases, $t_{0.01,99}$ is close to 2.364.

We also tested the algorithms when they attempted to maximize the generalized Nash product; i.e., the objective function was $\sqrt[|SERVERS|]{\prod_{i \in SERVERS}(U_i(alloc) - U_i(conflict))}$. In Figure 4, we present the average of the Nash product for backtracking on the sub-problems algorithm, the hill-climbing algorithm, and the genetic algorithm as a function of the number of datasets in the environments considered. Again, the simple backtracking algorithm did not find appropriate allocations in the cases we studied, while the hill-climbing algorithm achieved the highest Nash product, namely, the best results.

### 6.2.2 The Influence of the Objective Function

In the rest of the simulations, we used the hill-climbing method, as that seems to be the most promising method, and also since we found that the genetic algorithm is very sensitive to changes in its parameters. First, we studied more closely the effect of the choice of a social-welfare criterion on the hill-climbing algorithm's results. We ran simulations with the hill-climbing algorithm on randomly generated environments with 11 servers and 30 new datasets.

The social welfare criteria which we studied were: the sum of the servers' utilities, the generalized Nash product[16], and the sum of the servers' utilities without any constraint on the utility obtained by each server, which leads to the optimal solution of a centralized system (optimal) [17]. We compared these results with the servers' results from the static allocation,

[16]The product maximization method was proven in [44] to be the unique solution that satisfies some basic axioms, as described in Section 2.1.

[17]Finding the allocation which maximizes the sum of servers' utilities, when no constraints exist, was a polynomial problem, since, when using our utility function (see Appendix A), the sum of utilities from the

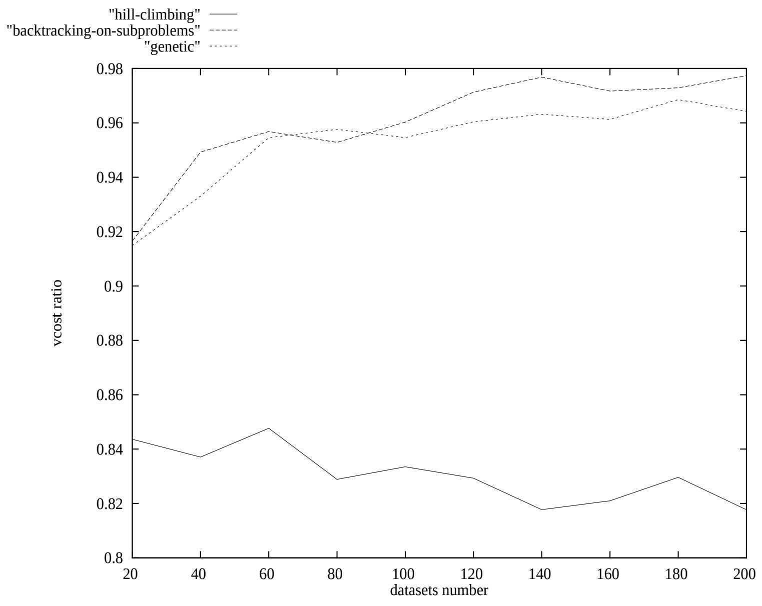

Figure 3: vcost_ratio as a function of the number of datasets when the different algorithms attempt to maximize the average utility of the servers.

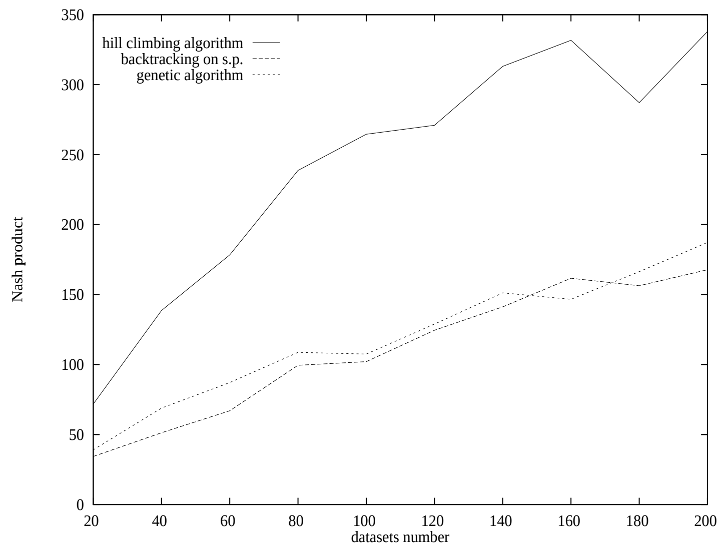

Figure 4: The Nash product of the servers utilities when the number of datasets varies.

| method | vcost ratio | CU | CI |
|---|---|---|---|
| static allocation | 1 | 0.148 | * |
| maximizing the sum of the servers' utilities | 0.77 | 0.118 | 0.94 |
| maximizing the generalized Nash product | 0.78 | 0.116 | 0.87 |
| optimal solution without constraints | 0.71 | 0.171 | 1.65 |

Table 1: Comparison of different social criteria.

which can be obtained without negotiation (and also served as our opting-out outcome). The results are presented in Table 1. The second column specifies the average of *vcost_ratio*. The third column (CU) indicates the average of the relative dispersion of the utility among the agents (std/mean), and the last one specifies the average of the relative dispersion of the added benefit (CI) of the negotiation among the agents. The results indicate that maximizing the sum achieves a slightly lower vcost ratio vs. maximizing the Nash product and is not far from the optimal solution. Maximizing the Nash product achieves a lower dispersion of the utilities and of the gains due to the negotiation, and they both do much better than the optimal w.r.t. dispersion of the utility among the servers.

### 6.2.3 The Influence of the Agent's Deviation

In Section 5.3.2, we presented a protocol in which each servers propose an offer which maximizes a predefined social welfare criterion, and the best offer according to that criterion will be agreed upon in the negotiation. We mentioned that a server may try to find an offer which is primarily good for itself, and only secondarily for society. It is important to know how such a deviation influences the negotiation results. In our test, we assume that each server tries to maximize its own utility, under the hard constraints of giving all the servers, at the very least, their conflict utility. The regulation was not changed; i.e., the allocation with the highest sum of utilities is chosen. Deviation of all the agents towards maximizing their own utility function is the worst possible case with regard to the maximization of the welfare criteria. We compare the results obtained in this case with the results obtained when each server tries to maximize the social welfare criterion, i.e., sum of the servers' utilities, and check the vcost ratio for each situation. We ran 100 runs on randomly generated environments, and the results are presented in Figure 5. The first bar, "self," represents *vcost_ratio* obtained when each server tries to maximize its own utility function. The second bar, "sum," represents *vcost_ratio* when each server tries to maximize the sum of the servers utilities, and the third bar, "optimal," is the lower bound of *vcost_ratio*, which means the result of maximizing the sum of the servers utilities ignoring the hard constraints We found that the results obtained when all the agents try to maximize their own benefits ("self") are still significantly better than the static allocation.

assignment of one dataset was independent of the assignment of the other dataset. Finding the allocation in this case is easily done, by allocating each dataset separately just at the location at which the sum of servers' utilities, with respect to this dataset, is the highest. This allocation does not necessary provide each server with more than its conflict allocation.

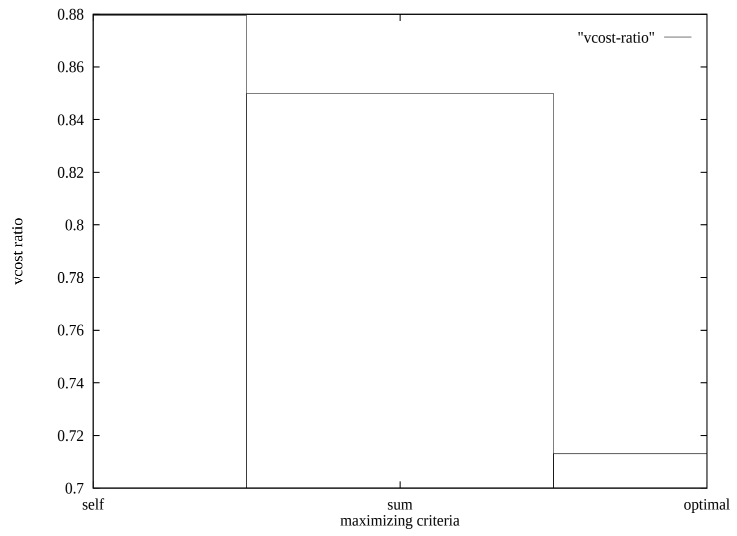

Figure 5: Deviation influence on the results: This figure presents the value of *vcost_ratio* for different situations. The first bar shows *vcost_ratio* obtained when each server tries to maximize its own utility, ignoring the social welfare criteria of maximizing the average of the servers' utilities. The second bar shows *vcost_ratio* obtained when maximizing the average of the servers' utilities, with constraints on the utility of each server. The third bar shows *vcost_ratio* obtained when maximizing the average of the servers' utilities, without any constraint.

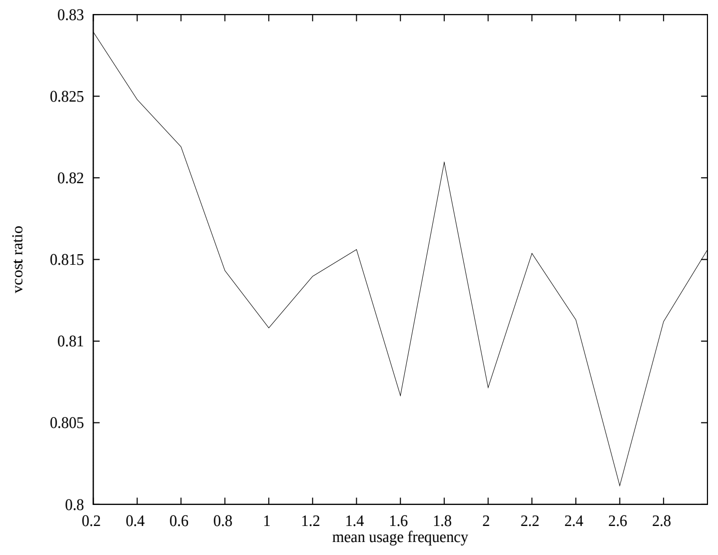

Figure 6: Changing frequency usage. This graph presents *vcost_ratio* obtained when the average frequency usage varies.

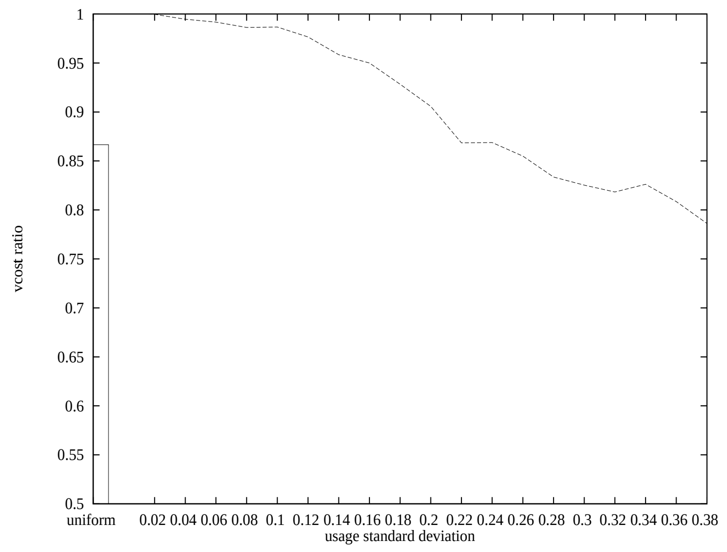

Figure 7: Changing frequency standard deviation. This graph presents *vcost_ratio* obtained when the average frequency usage varies. The bar describes the results when the usage is distributed uniformly.

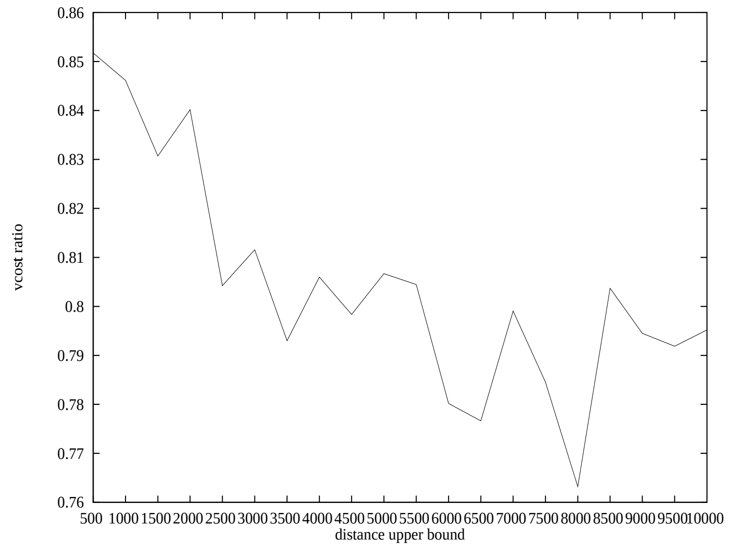

Figure 8: Changing Distance. This graph presents *vcost_ratio* obtained when the average distance between servers varies.

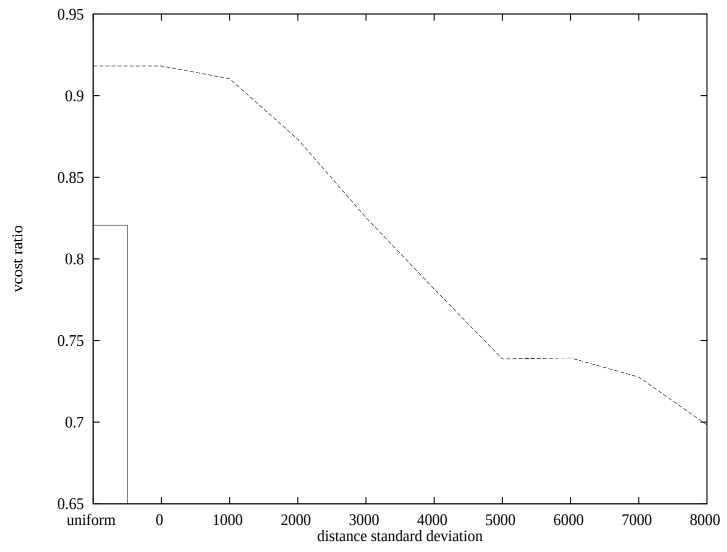

Figure 9: Changing std of distance. This graph presents *vcost_ratio* obtained when the standard deviation of the distances between sites varies. The bar describes the results when distances are distributed uniformly.

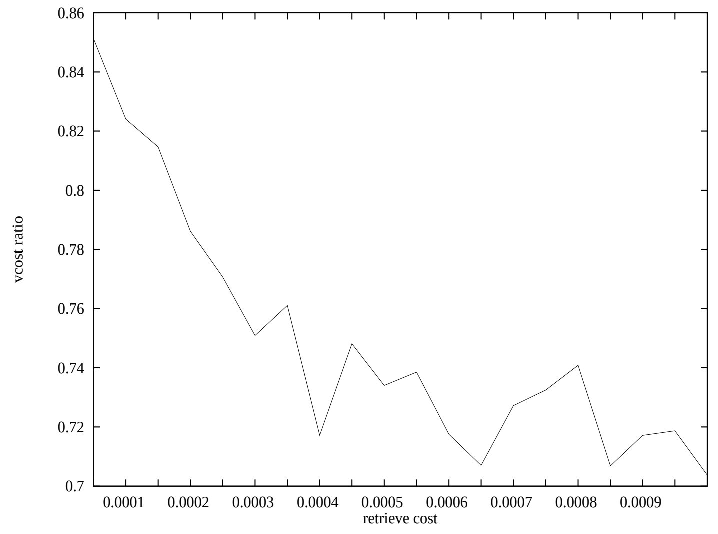

Figure 10: Changing retrieval cost. This graph presents *vcost_ratio* obtained when the retrieval cost varies.

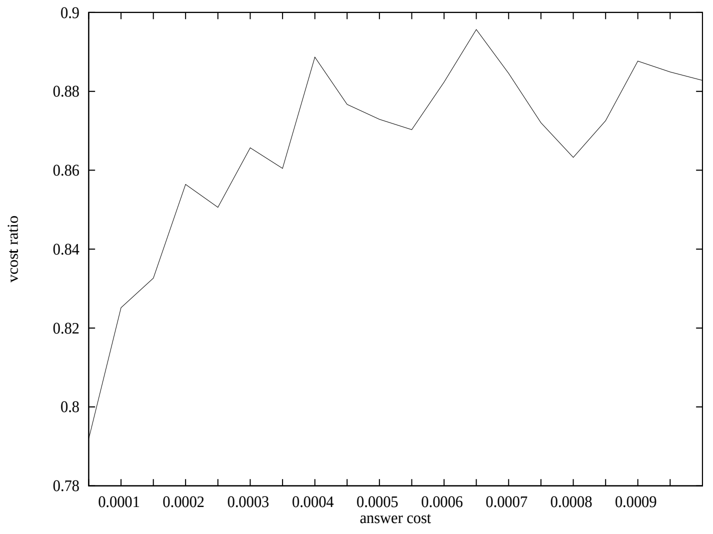

Figure 11: Changing answer cost. This graph presents *vcost_ratio* obtained when the answer cost varies.

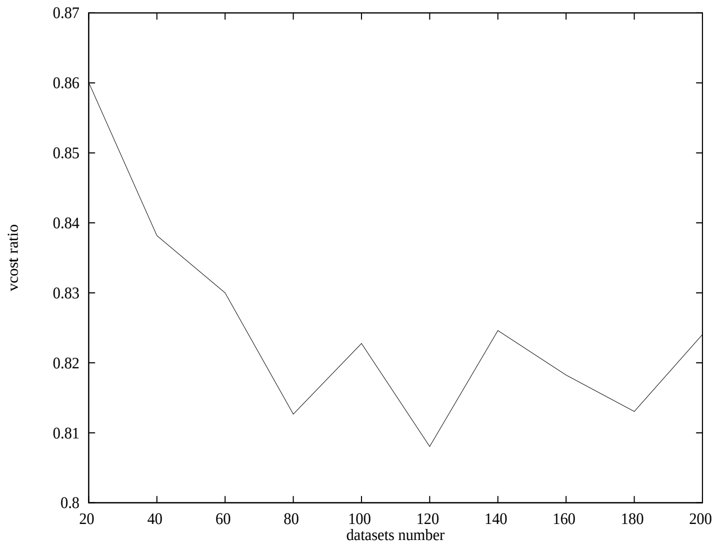

Figure 12: Changing the number of datasets. This graph presents *vcost_ratio* obtained when the number of datasets grows.

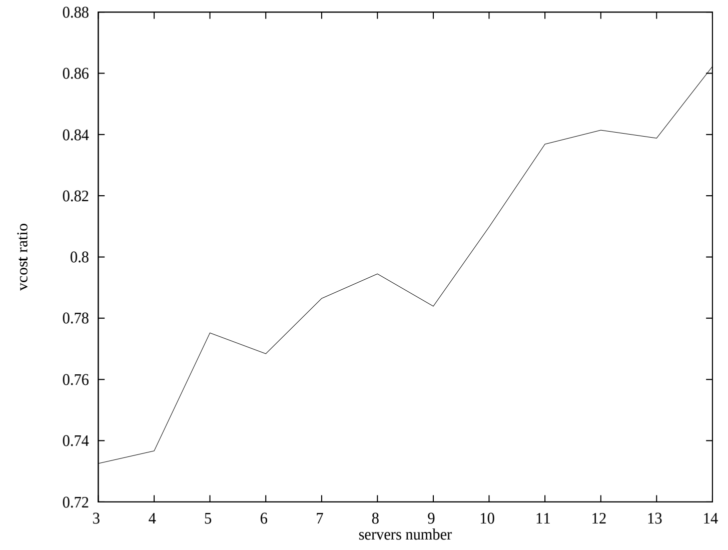

Figure 13: Changing the number of servers. This graph presents *vcost_ratio* obtained when the number of servers grows.

### 6.2.4 The Influence of Parameters of the Environment on the Results

Given a protocol and strategies for a problem, it is important to find out what the effect is of changing the parameters of the environment on the results of the protocol and strategies. In the following section, we examine by simulations the effect of several parameters of the environment on the results obtained when using negotiation. As above, the hill-climbing algorithm is used.

We examined how the change in several parameters of the environment influences *vcost_ratio*. In each set of simulations, we kept all but one of the parameters fixed. The randomly generated environments of the simulations include 11 servers and 100 datasets. We chose *answer_cost* to be similar to *retrieve_cost*, and thus, in general, theorized that each server prefers to store datasets in remote servers rather than locally.

We examined the effect of the distribution of the usage of datasets by servers. First, we considered situations in which, given a mean usage frequency, the usage frequency of each dataset by each server has a uniform distribution between 0 and 2·mean. As presented in the graph of Figure 6, changing the mean did not significantly influence the variable cost ratio. The reason for that can be the type of environments we have considered. The cost parameters we have chosen in the simulation cause the transmission cost to be the most significant component of the costs in the system. Changing the mean usage means a linear transformation over the transmission cost, and this does not significantly influence the results or the measurements.

The graph in Figure 7 presents *vcost_ratio* as a function of the standard deviation of the usage when the usage is derived from the normal distribution with mean 0.4. In this case, the variable cost ratio decreases as the std of the usage increases; i.e., negotiation is more beneficial when the usage of the datasets is more dispersed. The bar of Figure 7 describes the results when distances are distributed uniformly, rather than normally, with varied std as illustrated in the graph. In the case of uniform distribution, the usage frequency is generated between 0 and 0.8. Thus, its standard deviation is 0.23094. In fact, we see that the results obtained when the distances are uniformly distributed are similar to the results when the normal distribution with std is between 0.21 and 0.24. We can also see that as the standard deviation of the usage frequency increases, the results obtained are better. The reasoning behind this is that as the standard deviation of the usage decreases, the usage of a dataset by different areas is similar, and then the specific location of each dataset becomes less important.

We also studied the effect of the distances between any two servers on the *vcost_ratio*. We examined a uniform distribution between given minimal (min) and maximal (max) distances, and a normal distribution with a given std and mean of (min+max)/2. As presented in Figure 8, when the max distance (and thus the mean) of the distribution increases, negotiation becomes slightly more beneficial. However, as presented in Figure 9, the variable cost ratio decreases as the std of the distances increases; i.e., negotiation is of greater benefit when there is a greater difference between the distances of the servers. The reasoning behind this result is simple. As the standard deviation of the distances becomes smaller, the distances between any two servers becomes similar, and the location of each dataset becomes then less important. The first bar in Figure 9 represents the results when the distances are uniformly distributed. In this case the average standard deviation is 2742.41, and we actually obtained results similar to those that we obtained when the std was 3000 in the normal distribution case.

In addition, we examined the effect of the changes in the cost factors on the variable cost ratio. Changing the query price did not influence the results significantly. However, as Figure 10 demonstrates, as the *retrieve_cost* increases, the benefit of negotiation also increases.[18] Intuitively, this can be explained as follows. In our initial environment, the servers prefer not to store the datasets locally. However, when the *retrieve_cost* increases, the servers are more willing to store datasets locally, and thus, there are fewer constraints on possible allocation, and better agreements can be reached.

As presented in Figure 11, as the *answer_cost* increases, the benefit of negotiation decreases, and the datasets' mean size or storage costs have the same influence on the results.[19] This may be the result of the fact that increases in *answer_cost* cause the storage of datasets to be less beneficial, and thus it is more difficult to find a good allocation since the constraints are stronger, causing the improvement rate to be lower.

The number of servers and datasets also influences the results. As shown in Figure 12, when the number of servers is kept fixed, the results of the hill-climbing algorithm are more beneficial as the number of datasets increases. This is similar to the results presented in Figure 3, where a slightly different configuration was considered. On the other hand, Figure 13 demonstrates that when the number of datasets is fixed, the results are worse as the number of servers increases. We suspect that the reason for the first observation is that dataset allocations are independent of one another. Thus, as the number of datasets increases, there are more possibilities for increasing the benefit for all the servers. On the

[18]The *retrive_cost* is measured in dollars per distance unit; i.e., *retrieve_cost* $= 0.001$ means that transmission of 1 document over 1000 distance units costs the side which receives the document 1 dollar.

[19]*answer_cost* is measured in the same units as the *retrieve_cost*, i.e., dollars per 1 distance unit.

other hand, when the number of servers increases, there are more constraints on finding possible allocations, and it is much more difficult to find a sub-optimal allocation.

In summary, since the problem of dataset allocation in multi-agent environments is NP-complete, we have suggested heuristic algorithms for finding sub-optimal solutions to the problem: a deterministic backtracking algorithm, a backtracking algorithm on sub-problems, a random-restart hill-climbing algorithm, and a genetic algorithm. We found that the hill-climbing algorithm achieves the best results and that the backtracking algorithm on sub-problems can be used when a deterministic algorithm is needed. We also checked the influence of various parameters on the results.

### 6.3 Bidding v.s. Alternating Offers Negotiation

In [53], we considered a different variation of the data allocation problem. In the model described in [53], each server is concerned with the data stored locally, but does not have preferences concerning the exact storage location of data stored in remote servers. This situation occurs, for example, when clients send their queries directly to the server which stores the documents they need. For such an environment, we have proposed a bidding mechanism in order to achieve efficient results. We proved formally that our method is stable and yields honest bids, and we demonstrated the quality of the bidding mechanism, using simulations.

The bidding mechanism is not applicable in the environments considered in this paper, where each server is concerned with the exact location of each dataset. This is the case since, in the bidding mechanism, if a server would like to store a dataset it can make a high bid. However, there is no way for a server to influence the location of datasets which are not stored locally. The alternating offers protocol can be used in the environments considered in [53], where we use a bidding mechanism as a solution method. But, applying the negotiation model to the environments where servers do not care about the exact location of datasets not stored locally requires more computational resources than the bidding mechanism, and simulation results showed that using negotiations result in allocations which have lower average utility than the allocations which result from the bidding mechanism.

In conclusion, the bidding mechanism considered in [53] is not efficient in environments in which each server cares about the exact location of each dataset. However, using it in environments in which each server cares only about datasets stored locally, yields fair and efficient solutions, but requires a monetary system and does not guarantee a minimal utility, as the negotiation mechanism does.

## 7 Summary and Future Work

This paper presents negotiation protocols for the data allocation problem in multi-agent environments. We proved that negotiation is beneficial and that an agreement will be reached during the first time period. For situations of agents with incomplete information, a revelation process was added to the protocol, after which a negotiation takes place, as in the complete information case.

Although some of the steps of the proposed protocols can be carried out at a central source, our distributed model has several advantages. In our model, the agents use strategies that are in equilibrium, and thus our methods are appropriate for environments in which agents are trying to maximize their own expected utility. In addition, a centralized solution may become a performance bottleneck. More important, in order to use a centralized solution, the agents will need to agree on an entity (oracle) which will allocate the datasets, and the agents will not be able to act according to their self-interests.

In future work, we intend to deal with open questions of our model. Recall from Section 5.3.2 that we propose to divide the negotiation process into two phases after the revelation step. In the first phase, each server searches for an allocation using any of the algorithms and any of the resources available to it. In the second phase, the allocation with the highest value according to the predefined social-welfare criterion will be agreed upon by the servers. We propose that in the first phase each server offer an allocation which maximizes this predefined social welfare criterion. However, it may be worthwhile for a server to deviate and try to find an offer which is primarily good for itself, and only secondarily for society. We will study this problem and try to find stable mixed strategies for the behavior of the servers when searching for a feasible allocation, or, alternatively, we will try to use a more restricted protocol that will avoid this kind of behavior.

In our current data allocation model, only one copy of each dataset is permitted, since the datasets considered in systems such as EOSDIS are extremely large, and it is not efficient to allow multiple copies of a dataset. However, if the datasets are small, as in domains such as the Web, it is worthwhile to extend the model to allow multiple copies of each dataset and reallocation of old datasets. We leave this for future work.

We will also consider the details of the protocol for relocating old datasets by negotiation. Theoretically, at each time negotiation is considered, all the datasets can be considered in order to decide on the location of each dataset. However, this is not realistic when the number of datasets in the system becomes large. We have to find an efficient protocol for deciding which datasets will be relocated.

Another question that may be considered is finding better heuristics for the allocation which will be agreed upon in the negotiation and comparing it by using simulations with the methods we currently use. Furthermore, we have to study more rigorously the complexity of using sub-optimal techniques to solve the allocation problem for the cases of sequential response.

## A Details of the Utility Function

Each server, $i$, receives queries from clients in its area and answers them by sending back documents which may belong to a dataset located in $i$, or to a dataset located in another server, $j \neq i$. The clients pay server $i$ *query_price* per document which they received. If an answer document is located in a remote server, $j$, server $i$ needs to retrieve it from $j$. The cost for $i$ to retrieve a document from server $j$ depends on the virtual distance between $i$ and $j$, which is specified by the function *distance*. The virtual distance 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). We denote by *retrieve_cost* the cost for server $i$ of retrieving one document for one unit of distance.[20] Providing documents to other servers, when they need it for answering their queries, is also costly. We denote by *answer_cost* the cost for server $j$ to providing another server with one document over one unit of distance.

An important factor that plays a role in the utility function of a server from an allocation of a specific dataset is the expected usage of this dataset by the server and other servers. $Usage : SERVERS \times DATASETS \mapsto R^+$ is a function which associates with each server and dataset the expected number of documents of this dataset which will be requested by clients in the area of this server. Finally, we must consider the storage cost. We denote by *storage_cost* the cost of storing one data unit of a dataset in a server.[21] The function *dataset_size* specifies the size of each dataset in data units.

The utility function of an agent from a given allocation of a set of datasets is the combination of its utility from the assignment of each dataset. Taking the above parameters into consideration, the utility for a server from the assignment of one dataset, $(ds)$, to a certain location, $(loc)$, is as follows:

**Assumption A.1**

$$V_{server}(ds, loc) = \begin{cases} local(ds) & loc = server \\ remote(ds, loc) & otherwise \end{cases}$$

*where*

$$\begin{aligned} local(ds) = \; & usage(server, ds) * query\_price - \\ & storage\_cost * dataset\_size(ds) - \\ & \textstyle\sum_{d \in \mathrm{SERVERS}} usage(d, ds) * \\ & \qquad distance(d, loc) * answer\_cost \end{aligned}$$

*and*

$$\begin{aligned} remote(ds, loc) = \; & usage(server, ds) * query\_price - \\ & usage(server, ds) * distance(server, loc) * \\ & retrieve\_cost. \end{aligned}$$

The utility function of a server from any given allocation consists of the utility from the assignment of each dataset, as defined in Section 3.

We will first introduce $U_s$, ignoring the cost of negotiation delay.

**Assumption A.2** *For each server $\in$ SERVERS, and $S \in OFFERS$:*

$$U_s(S, 0) = \sum_{x \in \mathrm{DATASETS}} V_{server}(x, S(x))$$

According to the above definition, the utility from the assignment of one dataset is independent of the assignment of the other datasets. This property reflects that we do not take into consideration the overall load of the system, which may cause delays in transferring information and which yields more transfer costs. However, a severe load on one server is prevented, since, as we show later, the servers reach fair agreements.

[20]This cost is relevant only in environments in which each client is connected to the nearest server, which answers all his queries. In cases where the clients are autonomous and send their queries directly to the server which has the answer, $retrieve\_cost = 0$.

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

## References

[1] P. M. G. Apers. Data allocation in distributed database systems. *ACM Transactions on Database Systems*, 13(3):263–304, Sept 1988.

[2] M. Ben-Or and N. Linial. Collective coin flipping, robust voting games and minima of banzhaf values. In *Proc. 26th IEEE Symposium on the Foundations of Computer Science*, pages 408–416, Portland, 1985.

[3] V. Bhaska. Breaking the symmetry: Optimal conventions in repeated symmetric games. In *The 17th Arne Ryde Symposium on Focal Points: Coordination, Complexity and Communication in Strategic Contexts*, Sweden, 1997.

[4] A. H. Bond and L. Gasser. An analysis of problems and research in DAI. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[5] S. Ceri, G. Martella, and G. Pelagatti. Optimal file allocation in a computer network: a solution method based on the knapsack problem. *Computer Networks 6*, 6:345–317, 1982.

[6] W. W. Chu. Optimal file allocation in a multiple computer system. *IEEE trans. on computers*, C-18:885–889, 1969.

[7] S. E. Conry. Multistage negotiation for distributed constraint satisfaction. *IEEE Transaction on Systems Man and Cybernetics*, 21 (6):1462–1477, 1991.

[8] S. E. Conry, R. A. Meyer, and V. R. Lesser. Multistage negotiation in distributed planning. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[9] R. W. Cooper, D. V. DeJong, R. Forsythe, and T. W. Ross. Selection criteria in coordination games: Some experimental results. *The American Economic Review*, 80(1):218–233, 1990.

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

[11] K. Decker and V. Lesser. Analyzing a quantitative coordination relationship. *Group Decision and Negotiation*, 1993. Special Issue on Distributed AI.

[12] D.Frost and R. Dechter. In search of the best constraint satisfaction search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 301–306, 1994.

[13] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *Computing Survey*, 14 (2):289–313, 1982.

[14] 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.

[15] E. H. Durfee. What your computer really needs to know, you learned in kindergarten. In *Proc. of AAAI-92*, pages 858–864, California, 1992.

[16] E. H. Durfee and Victor R. Lesser. Negotiating Task Decomposition and Allocation Using Partial Global Planning. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 229–244. Pitman/Morgan Kaufmann, London, 1989.

[17] E. Ephrati and J. S Rosenschein. Deriving consensus in multiagent systems. *Artificial Intelligence*, 87(1–2):21–74, 1996.

[18] K. P. Eswaran. Placement of records in a file and file allocation in a computer network. In *Proceedings of the IFIP Congress on Information Processing, North Holland*, pages 304–307, North Holland, 1974.

[19] J. Farrell. Meaning and credibility in cheap-talk games. In M. Dempster, editor, *Mathematical Models in Economics.* Oxford University Press, 1988.

[20] S. French. *Decision Theory: An Introduction to the Mathematics of Rationality.* Ellis Horwood Limited, 1986.

[21] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, Cambridge, Ma, 1991.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freedman and Company, New York, 1979.

[23] L. Gasser. Social concepts of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47(1–3):107–138, 1991.

[24] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[25] J. R. Hackman, editor. *Groups that Work (and Those That Don't).* Jossey-Bass Publishers, San Francisco, Ca, 1991.

[26] H. Haller. Non-cooperative bargaining of $n \geq 3$ players. *Economics Letters*, 22:11–13, 1986.

[27] J. C. Harsanyi and R. Selten. *General theory of equilibrium selection in games.* MIT Press, Cambridge, Mass, 1988.

[28] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, 1995.

[29] John B. Van Huyck, Raymond, C. Battalio, and Richard O. Beil. Tacit coordination games, strategic uncertainty, and coordination failure. *The American Economic Review*, 80(1):234–248, 1990.

[30] J. C. Jamison. Valuable cheap-talk and equilibrium selection. In *The 17th Arne Ryde Symposium on Focal Points: Coordination, Complexity and Communication in Strategic Contexts*, Sweden, 1997.

[31] M. Kandori, G. J. Mailath, and R. Rob. Learning, mutation, and long run equilibria in games. *Econometrica*, 61(1):29–56, 1993.

[32] J. Kennan and R. Wilson. Bargaining with private information. *J. of Economic Literature*, XXXI:45–104, 1993.

[33] T. Khedro and M. Genesereth. Progressive negotiation for resolving conflicts among distributed heterogeneous cooperating agents. In *Proc. of AAAI94*, pages 381–386, 1994.

[34] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75(2):297–345, 1995.

[35] R. D. Luce and H. Raiffa. *Games and Decisions.* John Wiley and Sons, 1957.

[36] S. T. March and S. Rho. Allocating data and operations to nodes in distributed database design. *IEEE transactions on knowledge and data engineering*, 7 (2):305–317, 1995.

[37] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[38] T. Moehlman, V. Lesser, and B. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 2:161–191, 1992.

[39] B. Moulin and B. Chaib-Draa. An overview of distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 3–55. John Wiley & Sons, Inc., 1996.

[40] T. Mullen and M. Wellman. A simple computational market for network information services. In *Proc. of the First International Conference on Multiagent Systems*, pages 283–289, California, USA, 1995.

[41] R. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47(1), 1979.

[42] R. Myerson. *Game Theory.* Harvard University, 1991.

[43] NASA. EOSDIS Home Page. http://www-v0ims.gsfc.nasa.gov/v0ims/index.html.

[44] J. F. Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.

[45] J. F. Nash. Two-person cooperative games. *Econometrica*, 21:128–140, 1953.

[46] M. J. Osborne and A. Rubinstein. *Bargaining and Markets.* Academic Press Inc., San Diego, California, 1990.

[47] M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, Cambridge, Massachusetts, 1994.

[48] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[49] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, Boston, 1994.

[50] A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.

[51] R. Schwartz. Negotiation about data allocation in distributed systems. Master's thesis, Bar-Ilan University, Ramat-Gan, Israel, June 1997.

[52] R. Schwartz and S. Kraus. Negotiation on data allocation in multi-agent environments. In *Proc. of AAAI-97*, pages 29–35, Providence, Rhode Island, 1997.

[53] R. Schwartz and S. Kraus. Bidding mechanisms for data allocation in multi-agent environments. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, pages 61–75. Springer-Verlag, 1998.

[54] S. Sen, T. Haynes, and N. Arora. Satisfying user preferences while negotiating meetings. *Int. Journal on Human-Computer Studies*, 47(3):407–27, 1997.

[55] Sandip Sen and Edmund H. Durfee. The role of commitment in cooperative negotiation. *International Journal of Intelligent and Cooperative Information Systems*, 3(1):67–81, 1994.

[56] H. Siegelmann and O. Frieder. Document allocation in multiprocessor information retrieval systems. Technical Report IA-92-1, George Mason Univ., March 1992.

[57] C. Sierra, P. Faratin, and N. Jennings. A service-oriented negotiation model between autonomous agents. In *Proc. 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97)*, pages 17–35, Ronneby, Sweden, 1997.

[58] K. P. Sycara. *Resolving Adversarial Conflicts: An Approach to Integrating Case-Based and Analytic Methods.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1987.

[59] K. P. Sycara. Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242, 1990.

[60] H. M. Weingartner and D. N. Ness. Methods for the solution of the multi-dimensional 0/1 knapsack problem. *Operations Research*, 15 (1):83–103, 1967.

[61] M. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.

[62] M. J. Wooldridge and N. R. Jennings. Agent theories, architectures and languages: A survey. In *Intelligent Agents*, Lecture Notes in Artificial Intelligence No. 890, pages 1–39. Springer-Verlag, 1995.

[63] P. Young. The evolution model of bargaining. *Journal of Economic Theory*, 59:145–168, 1993.

[64] P. Young. The evolution of conventions. *Econometrica*, 61(1):57–84, 1993.

[65] D. Zeng and K. Sycara. Benefits of learning in negotiation. In *Proc. of AAAI-97*, pages 36–41, Providence, Rhode Island, 1997.

[66] G. Zlotkin and J. S. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transaction on Systems Man and Cybernetics*, 21 (6):1317–1324, 1991.