An Alternating Offers Bargaining Model for Computationally Limited Agents

Kate Larson
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
klarson@cs.cmu.edu

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
sandholm@cs.cmu.edu

ABSTRACT

An alternating offers bargaining model for computationally limited agents is presented. The gents compute to determine plans, but deadlines restrict them from determining an optimal solution. As the agents compute, they also negotiate over whether to perform a joint plan or whether to act independently and how, if implemented, the value of the joint plan would be divided. Their computing actions and bargaining actions are interrelated and both incorporated into each agent's strategy. We analyze the model for equilibrium strategies for agents under different conditions. It is shown that the equilibrium strategies for the alternating offers model where agents take turns making offers and counter-offers, even with its extremely large action space, are equivalent to those of a much simpler single shot, take–it–or–leave–it bargaining model. In particular, agents will compute and make no offers until the first agent's deadline.

Categories and Subject Descriptors

I.2.11 Multiagent systems

General Terms

Theory, Economics, Algorithms

1. INTRODUCTION

Recently, there has been a move from having multiagent systems with a central designer who controls the behavior of all system components, to having a system designer who can control only the *mechanism* (rules of the game), while allowing each agent to choose their own strategies. The efficiency of the system depends on the agents' strategies. So, to develop a system that leads to desirable social outcomes the designer must ensure that each agent is motivated to behave in the desired way. This can be done using the Nash equilibrium solution concept from game theory (or a refinement) [9]. The problem is that the equilibrium for rational agents does not generally remain an equilibrium for computationally limited agents. This leaves a potentially hazardous gap in game theory as well as automated negotiation since agents may no longer be motivated to behave in a desired way.

Decision making under settings of bounded resources is challenging even for single agents. The field of AI has long searched for useful techniques for coping with restricted resources. Herbert Simon advocated that agents should forgo perfect rationality in favor of limited, economical reasoning [19]. Considerable work has focused on developing *normative* models that prescribe how a computationally limited agent *should* behave (see, for example [14]). This is a highly nontrivial undertaking, encompassing numerous fundamental and technical difficulties. As a result most of those methods resort to various simplifying assumptions (see, for example [15, 3, 1]). While such simplifications can be acceptable in single-agent settings as long as the agent performs reasonably well, any deviation from full normativity can be catastrophic in multiagent settings. If the designer cannot guarantee that the strategy (including computing actions) is the best strategy that an agent can use, there is a risk that an agent is motivated to use some other strategy. Even if that strategy happens to be "close" to the desired one, the social outcome may be far from desirable.

Game theorists have also realized the significance of computational limitations (see, for example [13]), but the models that address this issue have mostly had a different focus, instead looking at such things as the complexity of computing strategies [5], both memory limitations and limited uniform-depth lookahead capability in repeated games [4], and showing that allowing choice in computation can undue the dominant strategy property of the Vickrey auction [16]. On the other hand, in many multiagent settings the limited rationality stems from the complexity of each agent's (optimization) problem, a setting which is ubiquitous in practice.

This paper studies the impact of limited computation on bargaining. Bargaining between agents has been studied in both the game theory literature [10] and the AI literature [6, 11, 7]. In non–cooperative game theory, the alternating offers model is a standard model. Much study has focused on the problem of delay in reaching agreement, when the value of the joint solution decreases over time and the values of the individual solutions do not change. Instead, in this paper we present and analyze an alternating offers bargaining model where agents have to use limited computational resources in order to determine what they are bargaining over. The agents' computational actions are interleaved with the bargaining actions and each agent's goal is to maximize its individual utility from the game. These utilities depend on what computation and bargaining strategies the agents use. We analyze the bargaining game for equilibria and show that the equilibrium strategies are in a certain sense equivalent to strategies agents would play in a simpler, take-it-or-

AAMAS'02, July 15-19, 2002, Bologna, Italy.

leave-it bargaining game with computational actions.

1.1 Example Application

To make the presentation more concrete, we now discuss an example domain where these methods are needed. Figure 1 is an example of a multiagent vehicle routing problem with two geographically dispersed dispatch centers that are self-interested companies. Each center is responsible for certain tasks (deliveries) and has a certain set of resources (vehicles) to take care of them. So each agent—representing a dispatch center—has its own vehicles and delivery tasks. Each agent's *individual problem* is to minimize transportation costs (driven mileage) while still making all of its deliveries and maintaining constraints involving maximum route length, maximum load weight and volume, delivery of all parcels and insisting that vehicles return to a depot of its center. This problem is $\mathcal{NP}$-complete [17].

Figure 1: ***Small example problem instance of the multiagent vehicle routing problem. This instance has two dispatch centers represented in the figure by computer operators. They receive the delivery orders and route the vehicles. The light dispatch center has light tasks and trucks while the dark dispatch center has darker tasks and trucks. The dispatch centers receive all of their delivery orders at once, and then have some deadline by which they must have determined some delivery route. For example, in some practical settings, the delivery tasks are known by Friday evening and the route plan for the next week has to be ready by Monday morning when the trucks need to be dispatched.***

There is a potential for savings in driven mileage by pooling the agents' tasks and resources—e.g., because one agent may be able to handle some of the other's tasks with less driving than the other due to adjacency. The objective in this *joint problem* is to again minimize driven mileage. This problem is again $\mathcal{NP}$-complete.

The agents independently decide which problem they each should spend some of their computation resources on. While they compute solutions for the problems, they engage in bargaining. The agents take turn making proposals where a proposal is an amount x that the other agent would be paid if the two agents coordinate their actions and execute the joint solution. The agent not making the proposal gets to decide whether to accept the offer or not. If the offer is accepted, the joint solution is implemented. If the offer is rejected, the game continues to the next round, each agent can take another step of computation and the agents switch roles for the bargaining stage. If no agreement is reached by the agents' deadlines, then each agent uses the solution it computed for its own problem.

1.2 The General Setting

The multiagent vehicle routing problem is only one example where the methods of this paper are needed. In general, they are needed in any 2-agent setting where each agent has an intractable *individual problem*, and there is a potential savings from pooling the problems, giving rise to an intractable *joint problem*. We also assume that the value of any solution to an agent's individual problem is not affected by what solution the other agent uses to its individual problem. Other applications, beyond the vehicle routing problem, include manufacturing, classroom scheduling, scheduling of scientific equipment among multiple users, and bandwidth allocation and routing in multi-provider multi-consumer computer networks, to name just a few.

Computation plays several strategic roles in the game. First, it improves the solution that is available—for any one of the three problems. Second, it resolves some of the uncertainty about what future computation steps will yield. Third, it gives information about what solution qualities the opponent can expect, and how the opponent is likely to allocate its computation. In equilibrium, an agent may want to allocate computation on its individual problem, the joint problem, and even on the opponent's problem.

In recent work we modeled a 2–agent bargaining setting where computing actions were treated strategically [7]. We analyzed simple bargaining settings for equilibrium strategies and provided algorithms for computing (off–line) the online computing strategies. Our focus was on the modeling of computing actions as part of the agents' strategies and we used a very simple bargaining model where agents' bargaining actions were restricted in that only one agent was allowed to make a single proposal at a specified deadline. In this paper we study a richer and natural bargaining model – the alternating offers model. Computing and bargaining actions are intertwined and agents can make proposals throughout the game. We discuss the importance of signaling and belief revision that occur in such a setting. In the next section we present the model and describe the computing and bargaining stages. We then analyze the game for equilibrium strategies.

2. THE MODEL

We study a setting where there are two agents, α and β, who are negotiating to reach agreement on whether to coordinate their actions in order to accomplish some plan, and if coordination occurs, on how to split the proceeds. As the agents negotiate they also compute possible solutions to the three different problems (own, opponent's, and joint). By some deadline, T, agreement must be reached or else both agents must implement their individual solutions with no coordination of actions. The process is divided into discrete stages. At each stage agents can compute one step on one of the three problems. After each agent has made the single computing step, one agent is allowed to make an offer to the other, specifying how much it would be willing to pay if the other agent agreed to cooperate on the joint problem, using the solution computed by the proposing agent. The agent receiving the proposal can either accept or reject. If the offer is accepted, the joint solution is implemented and the game ends. If the offer is rejected, the game continues for another stage, where the roles of the agents are switched. While both agents can observe all proposals and responses, the computational actions of the agents are private, i.e., an agent cannot directly observe what the other has computed on, but can only try to deduce this information from the other agent's offers and accept–reject decisions.

2.1 Normative Models of Deliberation Control

When agents have restrictions on their computational capabilities, there are tradeoffs that must be made when it comes to using the resources in the best way possible. In this section we present a way for agents to normatively control their computation.

We study a setting where computation improves the solution to problems. We assume that each agent has *anytime algorithms*.

Anytime algorithms, paired with a meta-level control procedure that determines how long to run an anytime algorithm, and when to stop and act with the solution obtained, are a model that allow for the trading off of computational resources for quality of results. The meta-level control procedures are called performance profiles, and are models of how the quality of a solution produced by an algorithm improves with computation time.

There has been a lot of work on performance profile based control of computation [20, 2, 1, 3]. To represent performance profiles we use a *performance profile tree* [7]. The advantage of this approach is that, unlike earlier approaches, it allows optimal conditioning on solution quality so far, the results of execution so far, as well as conditioning on the problem instance and other solution characteristics that may be deemed to be important.

We index a problem (z, i) by i and z, where i is an agent and z is the problem to be computed on. For each problem, (z, i), there is a performance profile tree, $\mathcal{T}^{(z,i)}$. This represents the fact that the tree may be conditioned on features of the problem instance. Figure 2 exemplifies one such tree.

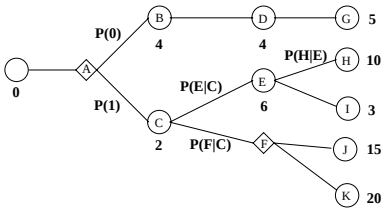

Figure 2: ***A stochastic performance profile tree for a valuation problem. The diamond shaped nodes are random nodes and the round nodes are solution nodes. At random node A, the probability that the random algorithm uses the number 0 is $P(0)$ while the probability that the algorithm uses number 1 is $P(1)$. The edges from any solution node have positive weight, while the edges from any random node have zero weight.***

A performance profile tree consists of two types of nodes, as well as weighted edges connecting nodes. Nodes can be either *solution nodes* or *random nodes*. Solution nodes store the solution that has been computed at a certain point in time. A solution node may store only the value of the solution, or may contain information about other solution features, such as slackness in schedules for example. Random nodes occur in the tree whenever a random number was used to chart the path of an algorithm run. For example, an anytime algorithm for the Travelling Salesman Problem (TSP) randomly chooses cities and swaps them in the route. A random node would be inserted into the tree whenever the algorithm randomly selected cities to swap. All edges in the trees have both a weight and a label associated with them. Every edge that originates from a solution node has a positive weight, while every edge emanating from a random node has a weight of zero. The weights correspond to the number of computational steps it takes to reach the child node. Edges from random nodes have zero weight since we assume that the choosing of a random number takes no computation steps for the agent. All edges in the tree are also labelled. The label on an edge is the probability of reaching the child node, given that the parent was reached. The labels (probabilities) on edges emanating from solution nodes are obtained from statistics obtained from running the algorithm in the path. The labels (probabilities) on edges emanating from random nodes are the probabilities that specific random numbers were used. The labels on the edges allows one to compute the probability of reaching any particular future node in the tree given any node. This is done by simply multiplying the probabilities on the path between the nodes. Another important aspect in computing is how much time it will take to obtain a certain solution. The weights on the edges correspond to the amount of time to reach the solution at the child, given that one has reached the parent. Therefore, the amount of time to reach a certain solution is simply the sum of the edge weights on the path to the solution.

We specify two different types of performance profiles; *stochastic* and *deterministic*. In a stochastic performance profile there is uncertainty as to what solutions will be obtained with future computation. At least one node in the tree has multiple children. The uncertainty can arise from the use of randomized algorithms, variation of algorithm performance on different problem instances, or some combination of both. In a deterministic performance profile the algorithm's performance can be projected with certainty. That is, the performance profile tree is a branch – all nodes have at most one child. Before taking any computation steps, an agent can determine what the resulting solution will be after any amount of computation is devoted to the problem.

Agents use the performance profile trees to help in making decisions about how to use their computational resources. As agents allocate computational time to an algorithm, the solutions returned move the agents from node to node in the performance profile tree. The performance profile trees provides information about how the solution is likely to improve with future computation. In practice, it is unlikely that an agent knows the solution for every time allocation without actually doing the computation. Rather, there is uncertainty about how the solution improves over time. A performance profile tree captures this uncertainty. The tree can be used to determine $P(v^{(z,i)}|t)$, denoting the probability that running to algorithm for t time steps results in a solution of value $v^{(z,i)}$. In particular, the performance profile tree supports conditioning on the path of solution quality. The performance profile tree that applies given a path of computation is the subtree rooted at the current node n. This subtree is denoted by $\mathcal{T}^{(z,i)}(n)$. If an agent has reached node n, which has a solution of value v, then when estimating how much additional computation would improve the solution, the agent need only consider paths that emanate from node n. That is, the agent need only be interested in the subtree $\mathcal{T}^{(z,i)}(n)$. The probability, $P_n(n')$, of reaching a particular future node, n', in $\mathcal{T}^{(z,i)}(n)$ is simply the product of the probabilities on the path from node n to node n'. The expected value of the solution after allocating t more time step to the problem, if the current node is n, is

$$\sum P_n(n') \cdot V(n')$$

where the sum is over the set $\{n'|n'$ is a node in $\mathcal{T}^{(z,i)}(n)$ which is reachable in t time steps$\}$, and $V(n')$ is the value of the solution at node n'.

Computation plays several roles. In particular, agents are free to devote computational time toward their own problems (individual or joint) in order to improve solutions or else gather information so as to have better knowledge as to what is the solution. Agents are also free to compute on other's individual problem in order to better be able to determine what solution their opponent has obtained. This information can be used during the negotiation process, or in the decision making process about where to devote more computation. However, there is one problem with agents computing on other agent's problem. An agent may not know which random numbers were used if the algorithm was randomized, nor may it have full information about which problem instances the other agent has encountered. Therefore, even if the same amount of computation is spent on the problem, different agents may get different solutions. To overcome this problem We allow agents to both *compute* on

problems and *emulate* problems. Computing on problems means to actually compute on a problem so as to obtain a usable solution. Emulation of an algorithm is slightly different. Instead of computing to obtain a usable solution, agents compute to see what solutions are possible. Whenever a random node is reached in the performance profile tree, an agent can choose which random number to use, instead of relying on a random number generator. This allows the agent to obtain information about what could happen along different pathes of the algorithm. When emulating, agents are also permitted to "back up" a performance profile tree, and use a different random number to see what other possible solution are obtainable. Solutions obtained while emulating can not be used by an agent as solutions to the actual problem, as they are *possible* solutions, not *actual* solutions.

An agent's *state of computation* has two components, the computing component and the emulating component. The computing component consists of one node for each performance profile tree, $\mathcal{T}^{(z,i)}$, which is the deepest node reached in the tree while computing on the problem (z, i). The emulation component consists of a set of nodes for each performance profile tree. Each set of nodes is the deepest node reached on the various paths explored by the agent. For both the computing and emulating components, no explicit path information is stored, as a node uniquely defines a path in the tree. The formal definition of the state of computation is as follows.

DEFINITION 1. *The* state of computation *for agent i at time i is*

$$\theta_i(t) = (\theta_i^c(t^c), \theta_i^e(t^e))$$

with $t = t^c + t^e$.

The computing component, $\theta_i^c(t^c)$, is a list of nodes, one for each performance profile tree, that agent i has reached by computing;

$$\theta_i^c(t^c) = \langle n^{(z,j)} \rangle \quad \textit{where } n^{(z,j)} \textit{ is a node in } \mathcal{T}^{(z,j)}$$

with $\sum_{(z,j)} \text{time}(n^{(z,i)}) = t^c$.

The emulating component, $\theta_i^e(t^e)$, is a list of sets of nodes, one set for each performance profile. These are the nodes that agent i has reached by emulating,

$$\theta_i^e(t^e) = \langle \{n^{(z,j)}\} \rangle \quad \textit{where } \{n^{(z,i)}\} \textit{ is a set of nodes in } \mathcal{T}^{(z,i)}$$

and with $\sum_{(z,j)} \sum_{n \in \{n^{(z,j)}\}} \text{time}(n) = t^e$.

In our equilibrium analysis, we assume that agents have the same performance profile trees for all problems, that is, $\mathcal{T}^{(z,\alpha)} = \mathcal{T}^{(z,\beta)}$ for all problems z. This models a real world setting where the agents have been solving similar problems in the past, and have been improving their algorithms for those problems resulting in comparable algorithms among the agents. This allows one to drop the agent indices on the trees, solution values and allocated time variables. However, all the results still hold if agents have different performance profiles for the same problems, as long as this is common knowledge.

2.2 Bargaining

The term *bargaining* is used to refer to a situation in which

1. Agents have the possibility of concluding a mutually beneficial agreement,
2. There is a conflict of interests about which agreement to conclude, and
3. No agreement may be imposed on any agent without its approval.

In our setting agents bargain over how to divide the surplus (or cost) associated with implementing the joint solution. Each agent prefers to receive more rather than less and each has the possibility to opt out of the bargaining procedure and to implement its individual solution with the associated value v_i. If the agents reach agreement, then they implement the solution to the joint problem.

An agent i's goal is to maximize its own utility, u_i. Its utility is determined by what solution is finally implemented and the cost or revenue generated by the solution. If agent i decides to implement its individual solution that has a value of v_i then the agent's utility is $u_i = v_i$. If, instead, the joint solution is implemented then the agent's utility depends not only on the value of the solution but also on what has been negotiated. If an agent accepts a proposal of x then its utility is $u_i = x$. If it has made an offer of an amount x which has been accepted, and has computed a joint solution valued v_{joint}, then its utility is $u_i = v_{\text{joint}} - x$.

2.3 Formal Model

The game is divided into stages. In every stage, both agents are allowed to perform one computational action on any problem they want. The computational action is followed by bargaining actions. In each stage one agent is the proposer and the other agent is the responder. The proposer makes an offer, x. If the offer is accepted by the responder, the game ends. The joint solution is executed, the utility for the responding agent is x and the utility for the proposing agent is the value it has computed for the joint solution at that point in time minus x. If the proposal is rejected then in the next stage, the two agents switch roles for the bargaining portion. The actions and strategies in such a game can be defined formally as follows.

DEFINITION 2. *Assume that at time t agent i is the proposer. An* action *for agent i at time t is*

$$A_i(t) = (a^z, x)$$

where $a^z \in \{a^\alpha, a^\beta, a^{\text{joint}}\}$ is a computing action and $x \in \mathbb{R} \cup \emptyset$ is an offer. The empty offer, $\emptyset$, signals that the agent does not want to implement the joint solution at time t.

At time $t + 1$, agent i's action is

$$A_i(t+1) = (a^z, \text{res})$$

where a^z is a computing action and $\text{res} \in \{\text{yes}, \text{no}\}$ is agent i's response to the opposing agent's offer.

A *history* describes the computing state of both agents at time t and the offers and responses of both agents.

DEFINITION 3. *At time t, a* history, $h(t)$, *is $h(t) = \theta_\alpha(t-1) \times \theta_\beta(t-1) \times ((x, \text{res})_i)_{i=0}^{t-1}$.*

There may be many histories at time t since there are many possible computation states each agent may be in (recall that computing actions are private). Let $\mathcal{H}_i(t) = \{h(t)\}$ be the set of all possible histories at time t, as perceived by agent i.

A strategy for an agent specifies an action for every possible history after which it has to move.

DEFINITION 4. *A* strategy *for agent i, S_i, is*

$$S_i = (\sigma_i(t))_{t=0}^{T}$$

where T is agent i's deadline and $\sigma_i(t) : \mathcal{H}_i(t-1) \to A_i(t)$.

We also allow for mixed strategies where $\sigma_i(t)$ is a mapping from the set of histories at time $t-1$ to a probability distribution over actions at time t.

In general, a *strategy profile* is a vector $s = (s_1, \ldots, s_n)$ which specifies one strategy for each player i in the game. The notation $s = (s_i, s_{-i})$ is used to denote the strategy profile where agent i's strategy is s_i and $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. Each agent wants to maximize its utility, $u_i(s)$, which is determined once all agents have chosen their strategies and the game has been played. Game theory is interested in finding stable points on the space of strategy profiles. These stable points are called *equilibria*. There are many types of equilibria but in this paper we shall be interested in some of the most common ones, dominant strategy equilibria, Nash equilibria, Bayes-Nash equilibria and perfect Bayesian equilibria.

A strategy is said to be *dominant* if it is a player's strictly best strategy against any strategies that the other agents might play. Formally, it is defined as follows.

DEFINITION 5. *Agent i's strategy s_i^* is a dominant strategy if*

$$u_i(s_i^*, s_{-i}) > u_i(s_i', s_{-i}) \; \forall s_{-i} \; \forall s_i' \neq s_i^*.$$

If all agents have a dominant strategy then there is a dominant strategy equilibrium.

DEFINITION 6. *A* dominant strategy equilibrium *is a strategy profile consisting of each player's dominant strategy.*

Agents may not always have dominant strategies and so dominant strategy equilibria do not always exist. Instead a different notion of equilibrium is often used, that of the Nash equilibrium.

DEFINITION 7. *A strategy profile s^* is a* Nash equilibrium *if no agent has incentive to deviate from its strategy given that the other players do not deviate. Formally*

$$\forall i \; u_i(s_i^*, s_{-i}^*) \geq u_i(s_i', s_{-i}^*) \; \forall s_i'.$$

The Nash equilibrium concept assumes that all agents have full information about the other agents in the game. This may not always be true. Instead, agents may have secret information that influences their choice of strategies, yet is unknown by the other agents. An agent's choice of actions may provide a signal as to what its private information is. These signals can be used by the other agents to update their beliefs about the private information. A Bayes-Nash equilibrium is a Nash equilibrium where agents update their beliefs about other agents' private information after observing signals [9]. We analyze the bargaining model using the *perfect Bayesian equilibrium* [9]. This solution concept requires that players form a complete system of beliefs about the opponents' types at each decision node that can be reached. The system of beliefs is updated according to Bayes' rule whenever possible. Given each player's system of beliefs, the strategies form best responses to one another in the sense of an ordinary Bayes-Nash equilibrium. A perfect Bayesian equilibrium is a profile of complete strategies and a profile of complete beliefs such that; *i)* Given the beliefs, the strategies are unilaterally unimprovable at each potential decision node that might be reached, and *ii)* the beliefs are consistent with the actual evolution of play as prescribed by the equilibrium strategies. When we include the computational actions into the agents' strategies, we obtain a *perfect Bayesian deliberation equilibrium* [7].

2.4 Agents' Beliefs and the Role of Signaling

Bargaining settings where agents are restricted to one round of communication at the end of the computing phase has been previously studied [7]. This form of bargaining has the advantage that agents did not leak information about their computation strategies. On the other hand, in the alternating offers model, each proposal and response provides information as to what problems the agents have computed on and what solutions have been obtained. Proposals and responses are *signals* that can be used by agents to update their beliefs about where the other has computed. Each agent uses this information, along with the values obtained by its own computing actions, to guide its future computing actions, proposals and responses.

Each agent's beliefs are centered on the state of computation that the other agent is in. Recall, that a state of computation is a list specifying which nodes in which performance profile trees an agent has reached by computing. An agent i's belief at time t, $B_i(t)$, is a probability distribution over the set of states of computation that agent j ($j \neq i$) may be in at time t. Let $b_i(\theta_j(t))$ be the probability with which agent i believes that agent j is in state of computation $\theta_j(t)$. An agent updates its beliefs whenever it receives some form of relevant information. This information comes from one of two sources; the agent's own computation, and the other agent's proposals and responses. The rest of this section describes how agents use these information sources to update beliefs.

First, we require that beliefs be consistent. For example, if at time $t-1$, agent i believed that $b_i(\theta_j(t-1)) = 0$ for some computation state $\theta_j(t-1)$, then agent i must have belief $b_i(\theta_j(t)) = 0$ if computation state $\theta_j(t)$ can only be reached by passing through $\theta_j(t-1)$. Secondly, agents obtain information from their own states of computation. This information can be used to update their beliefs about what computation states the other agent is in. In particular, given its own computation state at time t, $\theta_i(t)$, agent i is able to compute the probability with which it is possible to move from some state $\theta_j(t-1)$ to $\theta_j(t)$ by either reading the probabilities from the edges of the performance profile subtrees rooted at the nodes in $\theta_i(t)$, or from already having computed along pathes that contained nodes in $\theta_j(t)$. We denote by $P(\theta_j(t-1) \rightarrow \theta_j(t)|\theta_i(t))$ to be the probability that agent j may have moved from computation state $\theta_j(t-1)$ to $\theta_j(t)$, given agent i's own knowledge from computing. Then

$$b_i(\theta_j(t)) = \sum_{\theta_j(t-1)} P(\theta_j(t-1) \mapsto \theta_j(t)|\theta_i(t)) b_i(\theta_j(t-1)).$$

Agents also use information obtained from observing the bargaining actions of their opponent. Assume that agent i receives a proposal, x, from agent j at time t. Since we assume that agents are individually rational *in the sense that they would never knowingly make a proposal that, if accepted, would make them worse off than under no agreement*, agent j's proposals give information about the possible computation states it is in. In particular, it must be in a computing state $\theta_j(t)$ such that

$$v^{\text{joint}}(t^{\text{joint}}) - x \geq E[v^{\beta}(T - t^{\beta} - t^{\text{joint}})].$$

Therefore, once an offer, x, is observed, agent i can update its beliefs by using Bayes Rules. That is, given original belief $b_i(\theta_j'(t))$, agent i updates it to $b_i'(\theta_j'(t))$ by computing

$$b_i'(\theta_j'(t)) = \frac{P(x|\theta_j'(t)) b_i(\theta_j'(t))}{\sum_{o \in \{\theta_j(t)\}} P(x|o) b_i(o)}.$$

3. RESULTS

The bargaining settings differ based on whether agents have the same deadline or different deadlines, and whether the performance profiles are deterministic or stochastic. Without loss of generality, we assume that agent α gets to make the first proposal at time $t = 1$. This means that agent α proposes whenever time t is odd, and agent β proposes whenever time t is even.

3.1 Common Deadline

The setting where there is a common deadline T is analyzed first. The deadline is the time when the agents have to start execution of their solutions. Agreement between agents can be reached at any time before T, however if no agreement is reached by T, the agents must act on their individual solutions. No computation can occur beyond time T.

The expression "flattening out" is used in this section to refer to paths in the performance profiles. We say that a path of a performance profile to problem z has "flattened out" if for some t', $v^z(t') = v^z(t)$ for all $t \geq t'$.

3.1.1 Deterministic Performance Profiles

In a deterministic setting there exist situations (determined by the performance profiles) where there is a unique perfect Bayesian equilibrium, and other settings where there are multiple equilibria. The next theorem characterizes these different settings.

THEOREM 1. **Case 1***: Deadline T is odd.*

1. *$v^\alpha(T) \leq v^{\text{joint}}(T) - v^\beta(T)$ and either none of the performance profiles or only agent α's flatten out. Agent β has a dominant strategy $S_\beta = (\sigma_\beta(t))_{t=1}^T$ where*

$$\sigma_\beta(t) = \begin{cases} (a^\beta, \text{yes}) & \text{if } t \text{ odd, } t \leq T, \text{ and } x \geq v^\beta(T) \\ (a^\beta, \text{no}) & \text{if } t \text{ odd, } t \leq T, \text{ and } x < v^\beta(T) \\ (a^\beta, \emptyset) & \text{if } t \text{ is even} \end{cases}$$

That is, agent β computes only on its individual solution, makes no offers and only accepts an offer if it is greater than or equal to the value it expects to obtain for its individual solution.

Agent α has a unique best response, $S_\alpha = (\sigma_\alpha(t))_{t=1}^T$ where

$$\sigma_\alpha(t) = \begin{cases} (a^{\text{joint}}, \emptyset) & \text{if } t \text{ odd, } t < T \\ (a^{\text{joint}}, v^\beta(T)) & t = T \\ (a^{\text{joint}}, \text{no}) & t \text{ even, } t < T, \text{ and} \\ & x < v^{\text{joint}}(T) - v^\beta(T) \\ (a^{\text{joint}}, \text{yes}) & \text{otherwise} \end{cases}$$

That is, agent α computes only on the joint solution, makes no offers and accepts only offers that are greater than the difference between the best computable joint solution and the best computable solution for agent β's individual problem.

2. *$v^\alpha(T) \leq v^{\text{joint}}(T) - v^\beta(T)$ and the solution to β's individual problem flattens out. Agent β has multiple equilibrium strategies. It computes until it obtains the maximum value for its own problem and then may compute on any of the problems. It makes empty proposals and accepts any offer greater than its fallback. Agent α's best response strategy remains the same as in the first situation.*

3. *$v^\alpha(T) \leq v^{\text{joint}}(T) - v^\beta(T)$ and the solution to the joint problem flattens out. Agent β has a dominant strategy as in the first situation. However, α no longer has a unique best response strategy. Once it has computed the maximum value for the joint problem it can then compute on any of the three problems. The bargaining part of its strategy remains the same as in the first situation.*

4. *$v^\alpha(T) > v^{\text{joint}}(T) - v^\beta(T)$ and either none of the performance profiles or else only the one for the joint problem flattens out. Agent β has the same dominant strategy as in the first situation. Agent α also has a dominant strategy which is to compute on its own problem, make empty offers when t is odd and accept any proposal greater than $v^\alpha(T)$.*

5. *$v^\alpha(T) > v^{\text{joint}}(T) - v^\beta(T)$ and β's performance profile flattens out. There is no longer an undominated strategy for agent β. Once it has computed the maximum value for its individual problem then it may continue computing on any of the three problems. It makes empty proposals and accepts any offer which is greater than its maximum fallback value. Agent α has the same dominant strategy as in situation 4.*

6. *$v^\alpha(T) > v^{\text{joint}}(T) - v^\beta(T)$ and α's performance profile flattens out. β has the same dominant strategy as in the first situation. Agent α has no unique best response strategy. Once it has computed the maximum value for its own problem, it may then compute on any of the three problems. It will make the empty offer whenever t is odd and accept any offer greater than $v^\alpha(T)$.*

7. *If all performance profiles flatten out before T then it is possible that in equilibrium agreement may or may not be reached before the deadline.*

Case 2*: Deadline T is even. Therefore agent β gets to make the final proposal. The equilibrium strategy for agent α is the same as the equilibrium strategy for agent β is Case 1. The equilibrium strategy for agent β is the same as that of agent α in Case 1.*

Proof: We will only provide a proof of Case 1. The other cases follow a similar argument. Since T is odd, if the game reaches time T then agent α will get to make the final proposal. At this moment, the agents are engaged in a take–it or leave–it game. Agent α will try to make the lowest acceptable offer to agent β and agent β will want to be in its strongest bargaining position by having a high fallback value. That is, $v^\beta(T)$. Since $v^{\text{joint}}(T) - v^\beta(T) \geq v^\alpha(T)$, and both agents knew this before beginning the game, agent α will offer $x = v^\beta(T)$ and agent β will accept.

Assume that agent β had made an offer, x, at time $t < T$. Since we assume that neither agent would make or accept an offer that would result in a utility that is less than their expected utility from making no offer, it must be the case that at time t, $v^{\text{joint}}(t) - x \geq v^\beta(T - t)$. Since $v^\beta(T - t) \leq v^\beta(T)$, upon seeing an offer of x, agent α would update its beliefs about what possible solution agent β is able to compute for its own problem, reject the offer and propose $v^\beta(T - t)$ at time T. This results in a lower utility for agent β. If agent α makes an offer y such that $y \geq v^\beta(T)$ then agent β will accept it as this offer is greater than what it expects it can receive if it stays longer in the game. Agent α has a best response strategy to agent β's strategy. It knows that by computing only on the joint problem and offering agent β an amount of $v^\beta(T)$ at time T, then its utility will be $v^{\text{joint}}(T) - v^\beta(T)$. If it computes t time steps on its own problem anytime in the game then its utility would be $\max[v^{\text{joint}}(T-t) - v^\beta(T), v^\alpha(t)] \leq v^{\text{joint}}(T) - v^\beta(T)$. Assume that at time $t < T$ agent α makes an offer of x. If the offer is accepted by agent β then agent α must update its beliefs about what solution agent β has obtained and must conclude than it has made an offer which is too high. Therefore, agent α would deviate from this strategy, and would want to make a lower offer. In the limit, the offer agent α would want to make (after all belief updates) is $\emptyset$. □

3.1.2 Stochastic Performance Profiles

In the stochastic setting there is uncertainty associated with the values being computed. Agents no longer generally have dominant strategies when it comes to computing and bargaining. A proposal becomes a signal which leaks information to the other agent about what values the agent has computed. Each agent would like the other to believe that it is in a strong bargaining position, i.e., it has

a high fallback value and is willing to bargain hard in order to get the best possible deal. We obtain the following result.

THEOREM 2. *Assume that the deadline T is odd and that there exists at least one path in each performance profile that does not flatten out before T. Agent β has a dominant strategy $(\sigma_\beta(t))_{t=1}^{T}$ where*

$$\sigma_\beta(t) = \begin{cases} (a^\beta, \text{no}) & \textit{for odd } t,\ t < T \\ (a^\beta, \text{yes}) & \textit{for } t = T \textit{ and } x \geq v^\beta(T) \\ (a^\beta, \text{no}) & \textit{for } t = T \textit{ and } x < n^\beta(T) \\ (a^\beta, \emptyset) & \textit{for even } t \end{cases}$$

That is, agent β computes only on its individual problem, makes no offers and only accepts an offer if it is made at time T and is greater than, or equal to the value it can obtain from the solution of its individual solution.

Agent α has possibly multiple equilibrium strategies which all have the same form. At odd t, $t < T$, it offers $\emptyset$, for even t, $t < T$ it accepts any offer greater than $\max[\max v^\alpha(T), \max v^{\text{joint}}(T)]$. It follows a computing policy and makes an offer at the deadline such that its expected utility, $E[u_\alpha]$ is maximized, i.e.,

$$E[u_\alpha] = \max_{x,\left(t^\alpha, t^\beta, t^{\text{joint}}\right)} \{P(x \geq v^\beta(T))(v^{\text{joint}}(t^{\text{joint}}) - x) + (1 - P(x \geq v^\beta(T)))v^\alpha(t^\alpha)\}$$

where $(t^\alpha, t^\beta, t^{\text{joint}}) \in \text{Part(T)}$. If T is even, the equilibrium strategies of the agents are reversed.

Proof Sketch: The proof follows a similar argument to that of Theorem 1. The difference is that the agents no longer know with certainty what final solutions their opponent may have obtained from computing as there is uncertainty in the performance profiles. Therefore, the agent who gets to make the final offer at time T may use some of its own computational resources in order to refine its beliefs about what its opponent's fallback value is. □

In other words, no proposals are made before the deadline. At the deadline, an agreement may or may not be reached. There are some special cases where there are multiple equilibria. If T is odd and every path in agent β's individual problem's performance profile flattens out before T, then agent β will compute until it has obtained the maximum fallback value possible after which it may compute on any of the three solutions. Agreement still may or may not be reached at time T. If every single path in all performance profiles flatten out before T, then it is possible that the game will end before T.

The beauty of this result is that it takes a complex situation and reduces it to a simple one. While the space of possible actions for both agents is extremely large, in equilibrium agents behave as if they were in a restricted action space where only one agent is allowed to propose at only one time point (i.e., at the deadline). This simplified problem was discussed in an earlier paper where an algorithm was presented for determining the agents' equilibrium strategies (Algorithm 1) [7]. Given Theorem 2, this algorithm can also be used in the alternating offers setting. Then, using the equilibrium strategies, and depending on the algorithms' performance profiles and the problem instance, agreement may or may not be reached at the deadline.

3.2 Different Deadlines

Another situation which may arise is the case where agents have different deadlines. Let the agents' deadlines be noted by d_α and d_β. Since a deadline, d_i, means that at time $t = d_i$ agent i must begin executing a solution to a problem, the final deadline for the joint solution is $d = \min[d_\alpha, d_\beta]$. If $d < d_i$ for $i = \alpha$ or β, and no agreement is reached by d, then it is in the best interest for agent i to continue computing on the solution to its individual problems.

What is interesting in this setting is that the agent who has the most *power* in the negotiation is not necessarily the one with the latest deadline (unlike in, e.g., [18]). Instead, which agent gets to make the offer at the earliest deadline also plays a vital role in which strategies the agents will want to execute.

If both deadlines are common knowledge, the game is similar to the settings described in the previous section except that agreement must be reached by $\min[d_\alpha, d_\beta]$. However, the agents have to take into account that the agent with the later deadline can compute on its own problem also *after* the first deadline (and before its own deadline). The results from the previous section still hold with the following slight modifications to account for this. Due to space constraints, we omit the proofs.

THEOREM 3. *With different deadlines that are common knowledge, Theorems 1 and 2 hold if $v^\alpha(T)$ is replaced by $v^\alpha(d_\alpha)$, $v^\beta(T)$ is replaced by $v^\beta(d_\beta)$ and $v^{\text{joint}}(T)$ is replaced by $v^{\text{joint}}(\min[d_\alpha, d_\beta])$.*

If the deadlines are private information, but there is common knowledge of the joint distribution from which the deadlines are drawn, $f(d_\alpha, d_\beta)$, then the situation is more complicated. Agents must maintain a second set of beliefs about when its opponent's deadline will occur. As time passes, the beliefs are updated using Bayes rule and are used in speculating what the opponents fallback values might be.

In both the deterministic and stochastic settings a similar problem arises. The only equilibria that exist have the agents waiting for the first deadline to arrive. Early proposals leak too much information to the opposing agent and thus reduces the proposing agent's expected utility.

THEOREM 4. *If the agents' private deadlines are drawn from a joint distribution, $f(d_\alpha, d_\beta)$, then there will be no proposing and counter-proposing until the earliest deadline is reached. At that time an agreement may or may not occur.*

4. DISCUSSION

Whether or not agreement is reached depends on the agents' equilibrium strategies, performance profiles and the problem instance. Determining the equilibrium strategies for the agents involves Bayesian updating as agents update their beliefs as to what the other agent has obtained as solution values. For a computationally limited agent, the cost of performing such belief revision may be prohibitive. However, it has been shown that there exist strategies which are equivalent to the ones played in take-it-or-leave-it bargaining games. For some settings (depending on the performance profiles and knowledge about deadlines) there are algorithms that can compute offline, the online computing and bargaining policies for agents participating in take-it-or-leave-it bargaining games [7]. For example, Algorithm 1, Strategy Finder, based on dynamic programming, can be used to determine the optimal computing policy in situations where agents know each other's deadlines and have stochastic performance profiles [7].

In equilibrium, agent β will follow a computing policy which has it obtain a value of $v^\beta(T)$. The algorithm computes an optimal offer for agent α, for each possible state of computation, given that agent β has computed so as to maximize its fallback value. The algorithm then works backwards, determining the optimal computation and bargaining actions.

In certain settings, such as when agents' deadlines are drawn from some joint distribution, determining the optimal strategies is

Algorithm 1 Strategy Finder

Require: $T \geq 0$
  for each computation state $\theta_\alpha(T)$ at time T **do**
    $x^o_\alpha(\theta_\alpha(T)) \leftarrow \arg\max_x [P_a(x)[V(n^{\text{joint}}_\alpha) - x] + (1 - P_a(x))V(n^\alpha_\alpha)]$
  end for
  for $t = T - 1$ down to 0 **do**
    for each computation state $\theta_\alpha(t)$ **do**
      $a^z(\theta_\alpha(t)) \leftarrow \arg\max_{a^z} E[u_\alpha((a^z, \theta_\alpha(t)), S_\beta)]$
    end for
  end for
  return $((a^z(\theta_\alpha(t)))^{T-1}_{t=0}, (x^o_\alpha(\theta_\alpha(T), V(n^\alpha_\alpha))_{\theta_\alpha(T)})$

difficult because each agent's best-response strategy depends on the strategies of the other. It might be possible to use new game representations to solve for equilibrium in such games [5].

5. RELATED RESEARCH

There is a substantial body of work in both economics and AI that has investigated different bargaining settings. In this section we discuss some of the work that is closely related to this paper.

Rubinstein developed a bargaining model of alternating offers that took time into consideration [12]. He studied situations where time was finite, where both agents had the same deadline, and also situations where time was infinite but there was a cost function which reduced the value of agreement as time passed. In his model, both bargainers knew what the value of the item being negotiated over was, and what the fallback values were. While we use the alternating offers model, we include another dimension to the problem. Agents must invest resources in order to determine what is being bargained over and what the fallback values are.

In AI, Kraus, Wilkenfeld and Zlotkin looked at the alternating offers model of bargaining where agents take the passage of time into account [6]. They investigated the effects of time preferences on bargaining strategies. Other work that studied the effects of time in bargaining has been done by Sandholm and Vulkan [18]. They investigated a bargaining setting where agents were free to make and respond to offers at any point in time, but were constrained by deadlines. Our work is different from both of these since time is considered to be a resource that is used by agents to determine what is actually being negotiated over. The use of time for computation is an integral part of agents' strategies.

6. CONCLUSIONS AND FUTURE WORK

The contribution of this work is that it incorporates the computing actions of computationally limited agents into a complex, but standard bargaining model. In this paper we presented an alternating offers computing and bargaining model for agents negotiating in an open system. At each time step agents are allowed to take one computing step for computing solutions to either an agent's individual problem, its opponent's individual problem or the joint problem. Agents take turns making a proposal or responding to the other's proposal. We presented two main results. First, in a deterministic setting we characterized all possible equilibrium outcomes. The second result is that even in a stochastic setting, even with the rich bargaining model with a large strategy space, the game is equivalent to a much simpler bargaining setting with a small strategy space. If an agreement is reached, it occurs at the (earlier) deadline (for which there exist algorithms to determine off–line the optimal online computing strategies).

Future research includes generalizing this work to more than two agents. Using different models of bounded rationality (e.g., costly but unlimited computation [8]) would be another interesting direction of further exploration.

Acknowledgments

This material is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678.

7. REFERENCES

[1] M. Boddy and T. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[2] E. Hansen and S. Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.

[3] E. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *3rd Workshop on Uncertainty in AI*, pages 429–444, Seattle, 1987.

[4] P. Jehiel. Limited horizon forecast in repeated alternate games. *Journal of Economic Theory*, 67:497–519, 1995.

[5] D. Koller, N. Megiddo, and B. Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

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

[7] K. Larson and T. Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001.

[8] K. Larson and T. Sandholm. Costly valuation computation in auctions. In *TARK VIII*, pages 169–182, Sienna, July 2001.

[9] A. Mas-Colell, M. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[10] M. J. Osborne and A. Rubinstein. *Bargaining and Markets*. Academic Press, Inc., 1990.

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

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

[13] A. Rubinstein. *Modeling Bounded Rationality*. MIT Press, 1998.

[14] S. Russell. Rationality and intelligence. *Artificial Intelligence*, 94(1):57–77, 1997.

[15] S. Russell and E. Wefald. *Do the right thing: Studies in Limited Rationality*. The MIT Press, 1991.

[16] T. Sandholm. Issues in computational Vickrey auctions. *International Journal of Electronic Commerce*, 4(3):107–129, 2000.

[17] T. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[18] T. Sandholm and N. Vulkan. Bargaining with deadlines. In *AAAI'99*, pages 44–51, Orlando, FL, 1999.

[19] H. Simon. *Models of bounded rationality*, vol. 2. MIT Press, 1982.

[20] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.