

# Mechanism Design for Computationally Bounded Agents

Thesis Proposal

Kate Larson

Computer Science Department  
Carnegie Mellon University  
Pittsburgh, PA 15213

**Thesis Committee:** Tuomas Sandholm: Carnegie Mellon University  
Avrim Blum: Carnegie Mellon University  
Andrew Moore: Carnegie Mellon University  
Craig Boutilier: University of Toronto  
Mark Satterthwaite: Northwestern University

### **Abstract**

The frameworks of game theory and mechanism design have exerted significant influence on formal models of multiagent systems by providing tools for designing and analyzing systems in order to guarantee certain desirable outcomes. However, many game theoretic models assume idealized rational decision makers interacting in prescribed ways. In particular, the models often ignore the fact that in many multiagent systems, the agents are not fully rational. Instead they are computational agents who have time and cost constraints that hinder them from optimally determining their utilities from the game and which strategies are best to follow. Because of this, the game theoretic equilibrium for rational agents does not generally remain the same for agents with bounds on their computational capabilities. This creates a potentially hazardous gap in game theory and automated negotiation since computationally bounded agents are not motivated to behave in the desired way.

My thesis statement is that it is possible to bridge this gap. By incorporating computational actions into the strategies of agents, I plan to provide a theory of interaction for self-interested computationally-bounded agents. I will present a fully normative model of computation control which allows agents to make online decisions based on results of their computation and the restrictions placed on their computational capabilities. Using this model, I propose a new game theoretic solution concept, the deliberation equilibrium. This solution concept will allow me to investigate the impact computational restrictions have on agents' strategies in different mechanisms and will provide a foundation for developing mechanism design tools for computationally bounded agents. I finally propose to show experimentally that the model of computation control is feasible and general, and that it is possible to design robust multiagent systems for computationally bounded agents.

# 1 Introduction

It is an unavoidable fact that most systems will have bounded resources. There are restrictions on memory, space and computing cycles; Tasks have to be completed by deadlines; The costs of obtaining solutions may be prohibitively expensive. In such settings, perfect rationality does not always make sense. Perfect rationality, as used by economists, is focused on what actions agents make as opposed to how the agents decided to choose these actions. For example, the perfect rationality paradigm does not make a distinction between an agent exhibiting intelligent behavior that does the right thing by some means, and intelligent behavior that is the result of intelligent reasoning. When there are bounded resources, it is not fair to judge an agent irrational if it fails to take the best action. It well may have been behaving rationally *given the resources available to it*.

In multiagent systems the impact of bounded resources has large influences. 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); no agent is motivated to deviate from its strategy given that the others do not deviate [26, 23]. The problem is that the equilibrium for rational agents does not generally remain an equilibrium for computationally bounded 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 artificial intelligence 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. He claimed that "the global optimization problem is to find the least-cost, or best-return decision, net of computational costs" [43]. Considerable work has focused on developing *normative* models that prescribe how a computationally limited agent *should* behave (see, for example [10, 33, 5]). This is a highly nontrivial undertaking, encompassing numerous fundamental and technical difficulties. As a result most of those methods resort to simplifying assumptions such as myopic deliberation control [35, 36, 2], conditioning the deliberation control on hand-picked features [35, 36], assuming that an algorithm's future performance can be deterministically predicted using a performance profile [11, 13], assuming that an anytime algorithm's future performance does not depend on the run on that instance so far [3, 45, 44, 10] or that performance is conditioned on quality so far but not the path [9], or resorting to asymptotic notions of bounded optimality [34].

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 deliberation 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. Therefore, a fully normative deliberation control method is required as a basis for analyzing each agent's best strategy. This paper introduces such a fully normative deliberation control method. It takes into account that each agent may use all the information it has available to control its computation, including conditioning on the problem instance and the path of solutions found on the run so far. This paper will discuss how this deliberation control method can be used as a basis for deciding on an agent's best-response strategy: what deliberation actions and negotiation actions (offers, acceptances, and rejections) the agent should execute at any point in the game.

Game theorists have also realized the significance of computational limitations (see, for example [32]), but the models that address this issue have mostly analyzed how complex it is to compute the rational strategies [16] (rather than the computation impacting the strategies), memory limitations in keeping track of history in repeated games via deterministic finite automata or Turing machines (see, for example, [1, 29, 7]), limited uniform-depth lookahead capability in repeated games [14], or showing that allowing the choice between taking one computation action or not undoes the dominant strategy property in a Vickrey auction [38]. 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.

In auctions there has been work on both bounded–rational bidding agents and mechanisms. For bounded–rational bidding agents, Sandholm noted that under a model of costly computation, the dominant strategy property of Vickrey auctions fails to hold [38]. Instead, an agent’s best computation action can depend on the other agents. In recent work, auction settings where agents have hard valuation problems have been studied [31]. Auction design is presented as a way to simplify the meta–deliberation problems of the agents’, with the goal of providing incentives for the “right” agents to compute for the “right” amount of time. A costly computation model, with simple computation control where agents compute to refine their valuations, is used. However, situations where agents may compute on each others’ problems in order to refine their bids are not considered, nor are combinatorial auctions, where agents have to select which of their own valuation problems and which of their opponents’ valuation problems to compute on, studied.

There has also been recent work on computationally limited mechanisms. In particular, research has focused on the generalized Vickrey auction and investigates ways of introducing approximate algorithms to compute outcomes without loosing the incentive compatibility property [28, 15, 22]. These methods still require that the bidding agents compute and submit their valuations.

In my thesis I propose to focus on settings where the *computational capabilities* of agents are restricted. I propose to first study different mechanisms to see how restricted computation impacts the strategies agents use, and then to develop mechanism design tools for resource bounded agents.

**I propose that ;**

- 1. It is possible to incorporate agents computational actions into a game theoretic model,**
- 2. Agents’ computational restrictions have a strong impact the choice of strategies agents’ will make,**
- 3. It is possible to design mechanisms and mechanism design tools for computationally limited agents that take into account agents computational restrictions. Designing systems using these tools leads to more robust multiagent systems.**

## **1.1 Example Application**

To make the motivation more concrete, I now discuss an example domain where these methods are needed. Consider a multiagent vehicle routing problem [39] with two geographically dispersed dispatch centers that are self-interested companies (Figure 1). 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 while honoring the following constraints [39]:

- Each vehicle has to begin and end its tour at the depot of its center (but neither the pickup nor the drop–off locations of the orders need to be at the depot).
- Each vehicle has a maximum load weight constraint. These may differ among vehicles.
- Each vehicle has a maximum load volume constraint. These may differ among vehicles.
- Each vehicle has a maximum route length (prescribed by law).
- Each delivery has to be included in the route of some vehicle.

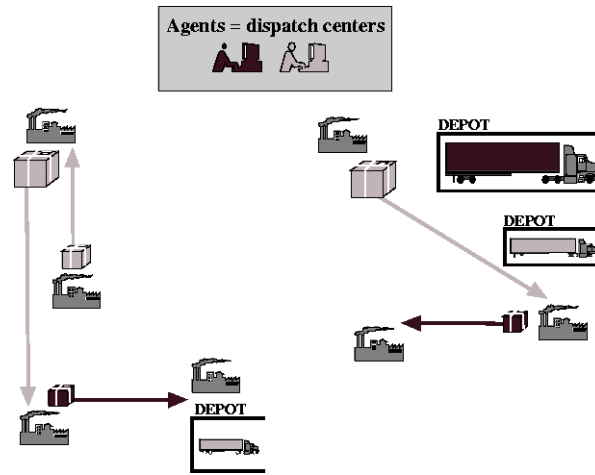


Figure 1: *Small example problem instance of the multiagent vehicle routing problem. This instance has two dispatch centers represented in the figure by computer operators. They receive the delivery orders and route the vehicles. The light dispatch center has light tasks and trucks while the dark dispatch center has darker tasks and trucks. The dispatch centers receive all of their delivery orders at once, and then have some time to compute a routing solution before the trucks need to be dispatched. 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 [39].*

An agent's individual problem is  $\mathcal{NP}$ -hard since  $\Delta\text{TSP}^1$  can be trivially reduced to it. The problem is in  $\mathcal{NP}$  because the cost and feasibility of a solution can be checked in polynomial time. Therefore, the problem is  $\mathcal{NP}$ -complete.

The geographical operation areas of the centers overlap. This creates the potential for either center to handle a delivery. There is a potential for savings in driven mileage by pooling the agents' tasks and resources since 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.

Whether the agents actually decide to coordinate their deliveries is determined by the costs associated with the different solutions. The agents must negotiate, or bargain, about whether to independently deliver their own packages, or whether to share their delivery tasks and resources in order to reduce costs. They must also negotiate as to how they will split the costs and benefits of the joint solution, if they agree to carry out a joint solution. However, before agents can decide whether to carry out a joint solution or the two individual solutions, they must have solutions (possible routes) for the three problems.

## 1.2 Outline

The rest of this document is organized as follows. In the next section I provide some background in game theory and mechanism design. I then describe the roles of computation in my setting and how I propose to model the bounded computational capabilities of the agents. This is followed by a description of how I propose to incorporate the computational actions of agents into a game theoretic framework. I then discuss work I plan to do on the effects of restricted computation on different mechanisms, followed with a proposal of mechanism design for computationally

<sup>1</sup>The  $\Delta\text{TSP}$  is a Traveling Salesman Problem where the distances between cities satisfy the triangle inequality.

bounded agents. I describe some experiment I plan to run and conclude with a summary of what I propose for the thesis work along with a schedule for completing the work.

## 2 Game Theory and Mechanism Design

Game theory is a method of studying systems of agents in settings where there are strategic interactions. This section is a short introduction to some important game theoretic concepts. A game has a set of agents and a set of outcomes  $O$ . Each agent has a set of strategies from which it chooses a strategy to use. A strategy is a contingency plan that determines what action the agent will take at any given point in the game. A *strategy profile*,  $s = (s_1, \dots, s_n)$ , is a vector specifying one strategy for each player  $i$  in the game. We use the notation  $s = (s_i, s_{-i})$  to denote a strategy profile where agent  $i$ 's strategy is  $s_i$  and  $s_{-i} = (s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$ . The strategies in the profile determine how the game is played out, and thus determine the outcome  $o(s) \in O$ . Each agent  $i$  tries to choose its strategy,  $s_i$ , to as to maximize its utility, which is given by a utility function  $u_i : O \mapsto \mathbb{R}$ . Unless otherwise specified, I will assume in the rest of my thesis, as is commonly done, that agents all have *quasi-linear* utility functions. That is,  $u_i(o) = v_i(x) - p_i$  where the outcome  $o$  defines some choice  $x$ ,  $v_i(x)$  is agent  $i$ 's valuation function, and  $p_i$  is a payment made by the agent.

Noncooperative game theory is interested in finding stable points in the space of strategy profiles. These stable points are the *equilibria* of the game. There are many types of equilibria but in this paper we focus on the two most common ones: *dominant strategy equilibria* and *Nash 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.

**Definition 1** Agent  $i$ 's strategy  $s_i^*$  is a dominant strategy if

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

The strategy is weakly dominant if the inequality is not strict.

If each agent's strategy in a strategy profile is the agent's dominant strategy, then the strategy profile is a *dominant strategy equilibrium*.

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 2** A strategy profile  $s^*$  is a Nash equilibrium if no agent has incentive to deviate from his strategy given that the other players do not deviate. Formally,

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

The Nash equilibrium is strict if the inequality is strict for each agent.

For Nash equilibrium it is assumed that all agents have complete information about each other, including the payoffs that each receives from the various outcomes of the game. However, this is a very strong assumption. Instead, agents may sometimes only have beliefs about other agents' preferences, beliefs about other agents' beliefs about their own preferences and so on. This idea can be modelled as something called a Bayesian game. It is assumed that a nonstrategic player, called Nature, makes a first move, and chooses realizations of random variables that determine each agent's preference type,  $\theta_i \in \Theta_i$  where  $\Theta_i$  is the set of all possible types for agent  $i$ . Each player observes their own type, and shares a common prior over the distribution of agents' types,  $F(\theta)$  where  $\theta = (\theta_1, \theta_2, \dots, \theta_n)$ . A solution concept called the Bayesian-Nash equilibrium is defined for Bayesian games.

**Definition 3** A strategy profile  $s^*$  is a Bayesian Nash equilibrium if for every agent  $i$  and all preferences  $\theta_i \in \theta$

$$\overline{u}_i(o(s_i^*, s_{-i}^*)) \geq \overline{u}_i(o(s_i', s_{-i}^*))$$

where  $\overline{u}_i$  denotes the expected utility over distribution  $F(\theta)$  of types.

A further refinement of the Bayes Nash equilibrium is the *perfect Bayesian equilibrium* [23]. 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 Bayesian Nash equilibrium. A perfect Bayesian equilibrium is a profile of complete strategies and a profile of complete beliefs such that:

1. Given the beliefs, the strategies are unilaterally unimprovable at each potential decision node that might be reached, and
2. The beliefs are consistent with the actual evolution of play as prescribed by the equilibrium strategies.

## 2.1 Concepts from Mechanism Design

Mechanism design is a subfield of game theory and microeconomics that studies how to implement good system wide solutions to problems that involve multiple self-interested agents. An important concept in mechanism design is the *social choice function*.

**Definition 4** A social choice function  $f : \Theta_1 \times \dots \times \Theta_n \mapsto O$  chooses an outcome  $f(\theta) \in O$ , given types  $\theta = (\theta_1, \dots, \theta_n)$ .

That is, given agents' types  $\theta$ , we want to choose an outcome  $f(\theta)$ . The goal of mechanism design is to implement the "rules of the game" by defining the strategies available to agents and the function used to select an outcome. For example, in an auction the mechanism specifies the strategies (bid at least the ask price) and the methods used to select the final outcome based on the agents' strategies (the item is sold to the agent with the highest bid). Formally, a mechanism is defined as follows.

**Definition 5** A mechanism  $M = (S_1, \dots, S_n, g(\cdot))$  is a set of strategies  $S_i$  available to each agent, and an outcome function  $g : S_1 \times \dots \times S_n \mapsto O$  such that  $g(s)$  is the outcome implemented by the mechanism for strategy profile  $s = (s_1, \dots, s_n)$ .

Game theory is used to analyze the outcome of the mechanism. Given some mechanism  $M$  and an outcome function  $g(\cdot)$ , the mechanism *implements* social choice function  $f(\theta)$  if the outcome computed with equilibrium strategies is a solution to the social choice function for all possible preferences of the agents.

**Definition 6** A mechanism  $M = (S_1, \dots, S_n, g(\cdot))$  implements social choice function  $f(\theta)$  if  $g(s^*(\theta)) = f(\theta)$  for all  $(\theta_1, \dots, \theta_n) \in \Theta_1 \times \dots \times \Theta_n$ , where  $s^*$  is an equilibrium solution to the game induced by  $M$ .

Ignoring any computational issues, there are several properties that are desirable for mechanisms. These include

- **Efficiency:** The outcome maximizes the total value over all agents.
- **Individual rationality:** No agent who participates in the mechanism makes an expected loss in utility.
- **Incentive compatibility:** It is best for all agents to truthfully report their preferences.

In truth, incentive compatibility is only used when talking about *direct mechanisms*. In a direct mechanism the only actions available to agents is to directly report things about their preferences. Therefore, in an incentive compatible direct mechanism, agents *truthfully* report their preferences in equilibrium. An example of an incentive compatible direct mechanism is the Vickrey auction.

At first, direct mechanisms would seem to be very restrictive. However, the *revelation principle* states that one can take any mechanism and convert it into an equivalent incentive compatible direct mechanism. However, as has been noted before, in complex distributed systems, it is not clear that the revelation principle still holds [30].

### 3 Modelling Computationally Bounded Agents

In this section I discuss computation. In particular I discuss the roles of computation in my model, how computation is restricted, and techniques for making decisions about how computation should be controlled.

#### 3.1 Roles of Computing

To participate in a market setting, agents need to determine valuations for the items being sold or bought. The question becomes: how are these valuations derived? In this paper we focus on situations where agents do not simply know their own valuations. Rather, they have to allocate computational resources to compute the valuations.

I classify problems in two settings. The first setting is where agents compute or acquire information in order to improve their valuations. For example, agents may be solving optimization problems as to how they will use the item being auctioned once they obtain it. An example is a procurement auction where agents are companies bidding on a task which consists of delivering a parcel from one location to another. The agent who submits the lowest bid wins and its utility is the bid amount,  $b_i$ , that it gets paid minus its cost,  $c_i$ , of performing the delivery. In order to determine the marginal cost of delivering the parcel, an agent must first determine how much it would cost to not deliver the parcel and then how much it would cost to include the parcel in its schedule. This results in the agent trying to solve to NP-complete problems. It may not be possible to solve optimally, thus resulting in the agent using some form of approximation [37]. As the agent computes on the problem of how to deliver the parcel, it can obtain better (less costly) vehicle routing solutions. The agent might then want to modify its bid, decreasing it so as to increase the likelihood of winning the auction.

The second setting is one where computation refines the agent's belief as to what the true valuation for the item is. Agents start by having a probability distribution over the true valuation of the items being auctioned. By acquiring information, the agents refine the probability distribution. The final valuation of the item may be higher than initially expected or it may be lower. Once the item is obtained, no further computation is required as it is assumed that its true value can be determined upon receipt. An example of such a setting could be an art auction where a bidder is not sure whether the art work is a forgery, and therefore be worth little, or an original and worth a lot. However, the bidder can do research (acquire information) and thus use this information to determine the true value of the painting.

A subtle difference between these two settings is that in the valuation improving setting the agent's valuation is what it has computed or determined, while in the refining setting the true valuation will be revealed to the agent when it obtains the good.

#### 3.2 Restrictions on Resources

In my thesis we consider two different limitations on an agent's computation resources. First, we investigate settings where agents have access to free computational resources, but are restricted by deadlines. At some point in time, the agents must stop computing on the valuations.

**Definition 7** An agent  $i$  is computing with deadlines if at time  $T_i$  it must stop computing on its valuation problems. Agent  $i$ 's computational deadline may occur before or after the close of the auction.

The second model is one where agents have no explicit restrictions on the amount of time they can compute. However, each computation step incurs a cost which becomes a limiting factor in the amount of computation available to an agent.

**Definition 8** Assume there are  $n$  possible problems on which agent  $i$  can use its computational resources in order to determine the valuations. Agent  $i$  has costly computation if there exists a cost function,  $c_i$ , where

$$c_i : T^n \rightarrow \mathbb{R}$$

where  $T^n$  is a time vector that specifies how much time agent  $i$  has spent computing on each of the  $n$  valuation problems. The function allows for an agent to have different costs for computing on different problems.

### 3.3 Anytime Algorithms

*Anytime algorithms* are a model that allows for the trading off of computational resources for quality of results [3]. The defining property of an anytime algorithm is that it can be stopped at anytime to provide a solution and the quality of the solution increases as more time is allocated to the problem. This makes it possible to compute approximate solutions to problems when there are time constraints. Anytime algorithms have been successfully used in many settings including planning and scheduling [3], information gathering [8], and belief network and influence diagram evaluation [12].

Anytime algorithms can be broadly classified into two groups. The first group includes iterative improvement algorithms such as iterative refinement. These algorithms always have a solution in hand and make small changes to the solution to try and improve it. Iterative improvement algorithms have been very successful in finding near optimal solutions to hard problems with very small amounts of time [24]. The second group of anytime algorithms contains complete algorithms such as many tree search algorithms. Branch and bound, for example, improves the bounds on the solution as more time is allocated to the algorithm [40].

### 3.4 Meta-Level Control: Performance Profiles

Alone, anytime algorithms do not provide a complete solution for finding “the least-cost, or best-return decision, net of computational costs” [43]. In order to do this, anytime algorithms have been 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. In this section, I describe *performance profiles* which is a model of how quality of solution produced by an anytime algorithm improves with computation time.

There has been much work on performance profile based control of computation [45, 9, 3, 11]. Boddy and Dean first coined the phrase to refer to a model of the performance of an anytime algorithm that specified the expected quality as a function of computational time [3]. Strictly speaking, such profiles are defined only for a particular input, and only for deterministic algorithms. However, performance profiles can be generalized to take into account differences in input quality and runs of randomized algorithms.

**Definition 9** A conditional performance profile of an anytime algorithm,  $P(q_j|x, t)$ , denotes the probability of obtaining a solution of quality  $g_j$ , given time  $t$  and inputs  $x$ .

Performance profiles have often been represented using a table of discrete values. Computation is discretized into a finite number of steps and solution quality is discretized into a finite number of levels. This approach allowed for conditioning algorithms' runs based on the problem instance but no information was used about the path an algorithm was following. Hansen and Zilberstein extended the idea of the conditional performance profile to allow for conditioning on solution quality so far [9].

Most work on anytime algorithms and performance profile based control of computation has focused on single agent settings. This has been a difficult task with numerous fundamental and technical difficulties. As a result, many of the methods have had to resort to simplifying assumptions such as myopic computation control [2, 35, 36], conditioning the control of computation on hand picked features [35, 36], assuming that an algorithm's future performance can be deterministically predicted using a performance profile [11], assuming that an anytime algorithm's future performance does not depend on the run on that instance so far [3, 10, 44, 45], conditioning the performance on quality but not path [9], or resorting to asymptotic notions of bounded optimality [34]

While these simplifications are 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. Since the strategy choice of one agent can influence what strategy is optimal for another agent, a small deviation by an agent can lead to social outcomes that are from expected and possibly far from desirable. Therefore, a fully normative method for controlling computation is required as a basis for analyzing each agent's best strategy. In the next section I will propose and present the performance profile tree which is such a method.

### 3.4.1 Performance Profile Trees

To represent the performance profiles I propose to use a different approach, a tree structure [18]. The advantage of this approach is that it allows optimal conditioning on results of execution so far, and allows conditioning on the actual problem instance.

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

The general tree form represents uncertainty that comes from both randomized algorithms and variation of performance on different problem instances. There are two different types of nodes in the performance profile tree, random number nodes and value nodes. A random node occurs whenever a random number is needed to chart the path of the algorithm run. For example, an anytime algorithm for the Traveling Salesman Problem (TSP), randomly chooses cities and swaps them in the route. In this situation, a random number would be a city label. The edges from a random node represent the possible random numbers (bits), and are labeled with the probability of a certain random number occurring. Each edge emanating from a random node has a weight of zero. Value nodes store the value of the solution that the algorithm has computed at a certain point in time. The children of a value node may be either more value nodes or random nodes. Each edge emanating from a value node has a weight of one time step. The edges are labeled with the probability of reaching the child, given that the parent was reached. This allows one to compute the probability of reaching any particular future node in the tree given any node, by multiplying the probabilities on the path between nodes. If the only uncertainty in the algorithm's run is from the variation of performance on different problem instances, then there are no random nodes in the performance profile tree.

We specify two different types of performance profiles, *stochastic* and *deterministic*. In a *stochastic performance profile* there is uncertainty as to what results future computation will bring. At least one node in the tree has multiple children. The uncertainty can come from variation in performance on different problem instances or from the use of a randomized algorithm. In a *deterministic performance profile* the algorithm's performance can be projected with

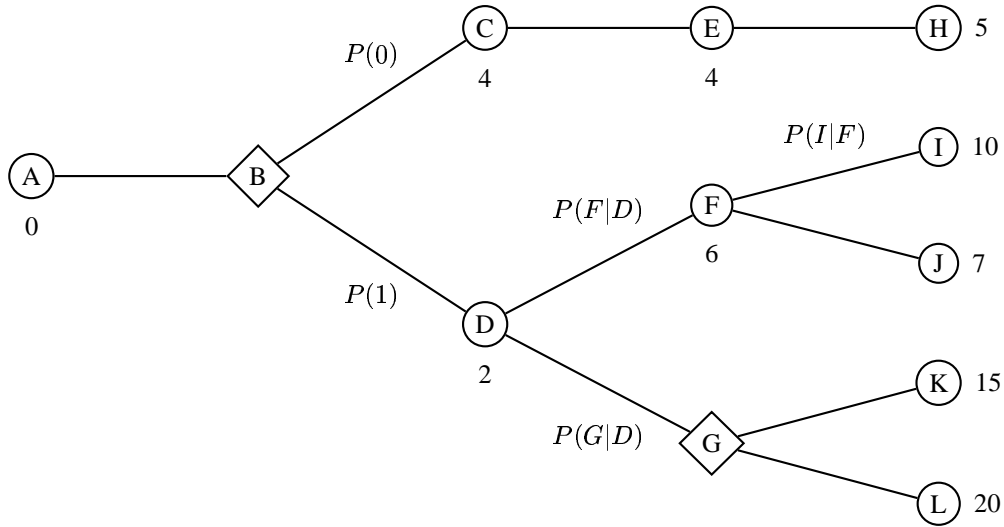


Figure 2: An agent's performance profile tree for one problem. The diamond shaped nodes are random nodes and the round nodes are value nodes. At random node A, the probability that the random number will be 0 is  $P(0)$ , and the probability that the random number will be 1 is  $P(1)$ . The edges from any value node have a weight of one time step, while the edges from any random node have a weight of zero time steps. This means it takes 1 time step to reach node D, and 2 time steps to reach node F. In general, an agent would have several trees, one for its own problem, one for the other agent's problem and one for the joint problem.

certainty (that is the tree is a branch). Before doing any computation, an agent can determine what the solution will be after any number of computation steps devoted to the problem.

If the only uncertainty comes from the variation in the performance of the algorithm on different problem instances, then agents can compute on others problems, and are able to obtain identical results. This information can be used when it comes to the bid formation process. If an agent can determine that another agent has a high valuation for a good, then it might decide that the likelihood of obtaining the good in the auction is small, and thus can concentrate its resources on obtaining valuations for other items. However, if agents are running randomized algorithms, then, since the path is dependent on random numbers, one agent's results from computing on a certain valuation problem may be different from the valuation obtained by another agent computing on the same problem. Therefore, we allow one agent to *emulate* the possible algorithm runs another agent may be following. While emulating an algorithm run, whenever an agent reaches a point where a random number is used, the agent can select which random number to run the algorithm with. At a later point in time, the agent may revisit the same random node, and select a different random number with which to run the algorithm. While the emulation does not remove all uncertainty as to what valuations other agents have computed (as the actual random numbers are not known), it does allow an agent to remove some uncertainty as it provides information about the possible paths the algorithm might have taken.

Agents can use their computation resources in two different ways. Agents can *compute* in order to determine what their solutions to the various problems will be. However, if, for example, agent  $\alpha$  computes on agent  $\beta$ 's individual problem, then there is no guarantee that the random number generator will produce the same numbers. Therefore, the results obtained by both agents may be quite different. Therefore, each agent is allowed to use their computation resources to either run an algorithm or *emulate* the run of a random algorithm. When an agent is running an algorithm,

whenever a random node is reached, the random number produced by the random number generator is used to choose the path of computation. The solution obtained from running the algorithm is the solution that can be used for solving the actual problem. Emulation of an algorithm is slightly different. With emulation, an agent uses its computation resources in order to determine what the solution would have been if the algorithm had followed a certain path. Whenever a random node is reached, the agent *chooses* a random number instead of using the number generated by the random number generator. This allows the agent doing the emulation to learn what solution *would* have been obtained if the random numbers generated were the same as the ones it chose. This provides the agent doing the emulation information about what values the other agent may have obtained for its solutions. Agents are allowed to emulate any algorithm, including the algorithms for the joint problems and the agents' own problems.

Agents keep track of the amount of time spent deliberating or emulating. The amount of time deliberating to reach a node  $n$  is denoted by  $\text{time}(n)^d$ . This is equal to the sum of the weights of the edges on the path from the root of the performance profile tree to the node  $n$ . Determining the emulation time spent to reach a node  $n$ ,  $\text{time}(n)^e$ , is more complex. It is not just the sum of the edges along the path to the node, as in the computation case. This would lead to over counting the time as an agent does not have to start at the root of the tree each time it emulates a run. Instead, it can back up to a random node, choose a new random number and continue emulation from there. Therefore, when determining the emulation time for each node in the emulation component the following approach is used. For any random node  $nr$ ,  $\text{time}(nr)^e$  is equal to the sum of the edges of the path to  $nr$ . Let  $n_1$  and  $n_2$  be two value nodes in the emulation component which are in the same performance profile tree. Let  $n'$  be the least common random node ancestor of  $n_1$  and  $n_2$ . Then,  $\text{time}(n_1)^e$  is equal to  $\text{time}(nr)^e$  plus the sum of the edges on the path from  $n'$  to  $n_1$ . Since the path from the tree root to random node  $n'$  has already been traversed when emulating the algorithm and reaching node  $n_1$ , it should not be counted again when determining the time to reach node  $n_2$ . Therefore,  $\text{time}(n_2)^e$  is equal to the sum of the weights on the edges along the path from node  $n'$  to node  $n_2$ . Then for the emulation component,  $t^e$  is equal to the sum of the times of the nodes in the component.

In practice it is unlikely that an agent knows the valuation for every time allocation without actually doing the computation. Rather, there is uncertainty about how the valuation improves over time. A performance profile tree allows one to capture this uncertainty. The tree can be used to determine  $P(v_i^g | t)$  denoting the probability that running the algorithm for  $t$  time steps produces a solution of value  $v_i^g$ .

Unlike earlier performance profile models for controlling anytime algorithms (for example, [45, 3, 11, 9]), the performance profile tree supports conditioning on the path of valuation quality so far. 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  $\mathcal{T}_i^g(n)$ . If an agent is at a node  $n$  with value  $v$ , then when estimating how much additional computation would increase the valuation, the agent need only consider paths that emanate from node  $n$ . The probability,  $P_n(n')$ , of reaching a particular future node  $n'$  in  $\mathcal{T}_i^g(n)$  is simply the product of the probabilities on the path from  $n$  to  $n'$ . The expected valuation after allocating  $t$  more time steps to the problem, if the current node is  $n$ , is

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

where the sum is over the set  $\{n' | n' \text{ is a node in } \mathcal{T}_i^g(n) \text{ which is reachable in } t \text{ time steps}\}$ .

Agents store the results of their computation actions at each time step in a *computation state*. The state of computation contains a list of nodes that the agents has reached in each performance profile, and whether the node was reached by deliberating or emulating.

**Definition 10** *The state of computation for agent  $i$  at time  $t$  is*

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

where  $t = t^d + t^e$ .

The computation component,  $\theta_i^d(t^d)$ , is a vector of nodes, one in each performance profile tree, that agent  $i$  has reached by taking computation steps,

$$\theta_i^d(t^d) = \langle n_j^g \rangle_{j \in A}^{g \in G}$$

where  $\sum_{j,g} \text{time}(n_j^g) = t^d$ .

The emulation component,  $\theta_i^e(t^e)$ , is a vector of lists. There is one list for each performance profile tree. Each list consists of the nodes that agent  $i$  has reached by emulating in the tree. The lists are indexed by the performance profile trees ( $j$  and  $g$ ) and the nodes in the lists are labeled with the random numbers chosen on the path to the node ( $R$ ),

$$\theta_i^e(t^e) = \langle (n(r))_j^{r \in R, g} \rangle_{j \in A}^{g \in G}$$

where  $\sum_{j,g} \sum_{r \in R} \text{time}(n(r)) = t^e$ .

If the agent's computation is costly, then the state of computation additionally maintains information on the cost incurred. This is done by additionally keeping a record of  $c_i(\bar{t})$  where

$$\bar{t} = \left( \sum_{r \in R} \text{time}(n(r))_j^{r \in R, g} + \text{time}(n_j^g) \right)_{j \in A}^{g \in G}.$$

A *computation set* is the set of computation states that an agent can reach in exactly  $t$  computation actions.

**Definition 11** The *computation set* of agent  $i$  at time  $t$  is

$$\Theta_i(t) = \{ \theta_i(t) | t^d + t^e = t \}$$

## 4 Game Theory for Computationally Bounded Agents

In order to analyze strategic interactions between computationally bounded agents, the computational actions of the agents must be incorporated into the agents' strategies. Once this is done it is possible to talk about equilibria in a computationally bounded setting.

**Definition 12** A (Nash, dominant, Bayesian etc) deliberation equilibrium for computationally bounded agents is a (Nash, dominant, Bayesian, etc) equilibrium where the agents' computing strategies for a (Nash, dominant, Bayesian etc) equilibrium, and the agents' noncomputing strategies are in (Nash, dominant, Bayesian, etc) equilibrium.

Agents use their computational resources in different ways. They can compute on their own problems in order to obtain better valuations. They can also compute on their opponents' problems in an attempt to gather information about the actions that the opponents; may be taking. We make a distinction between these two types of computation. The first we call *weak strategic computation*. The second is called *strong strategic computation*.

**Definition 13** If an agent  $i$  uses part of its computation resources to compute on another agent's valuation problems, then agent  $i$  is performing **strong strategic computation**.

**Definition 14** If an agent  $i$  does not actually use its computational resources to compute on another agent's valuation problems, but does use information from the opponent's performance profile to devise a strategy, then agent  $i$  is performing **weak strategic computation**.

## 5 The Impact of Bounded Computation on Standard Mechanisms

It is important to understand how agents' computational limitations affect the strategies that they choose to follow in different settings. In this setting I describe both work I have completed and work that I plan to do on the impact of bounded computation on standard mechanisms. I focus on three different negotiation settings. The first is one-to-one negotiation where agents bargain over what problem should be solved and how any rewards from solving a problem should be shared. The second is a one-to-many negotiation setting. In particular, I study auctions, both single-item and multiple-item, where agents must compute in order to determine their valuations for the items up for bid. The third setting I study is many-to-many negotiation, where I plan to study the impact of bounded resources on the equilibrium outcomes of exchanges.

### 5.1 One-to-One Negotiation: Bargaining

The term *bargaining* is used to refer to a situation where

1. Agents have to 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 individual without its approval.

In particular, I am interested in settings where agents bargain over how and which problem to solve. Each agent has some individual goals they wish to achieve. However, by collaborating it may be possible to achieve all goals in a more efficient or cheaper manner. The agents' preferences are opposed in that each agent prefers to receive more rather than less if a joint solution is implemented, and each agent has the possibility to opt out and work solely on their individual problems. If agents reach agreement then they implement the joint solution using the solution of the agent who made the acceptable proposal. The accepting agent's utility from the bargaining is the amount,  $x$ , that it agreed to accept. The agent who made the accepted proposal has utility equal to the value of its joint solution minus the amount paid to the other agent. If no agreement is reached then both agents independently implement the solutions to their individual problems, and obtain a utility equal to the value of the solution for their own problem.

There are many different bargaining mechanisms but I focus on two in particular. The first is a single-shot bargaining mechanism while the second is an alternating offers mechanism.

#### 5.1.1 Single Shot Bargaining Mechanism

In this section I describe the single shot bargaining mechanism, and some results for agents' that have limited computational resources, that is, agents who have deadlines that restrict their computational capabilities. For a more detailed description of the limited computation bargaining model see [19].

Each agent has a deadline, these deadlines may be different. The deadline of agent  $i$  is denoted by  $T_i$ . Each agent can compute up to their own deadlines, but must stop all computation after that. One agent is the proposer. It gets to make one proposal that the other agent can accept or reject. If the proposal is accepted then the joint solution is implemented, the accepting agent receives what was offered and the proposing agent receives the difference between its computed joint solution and what was offered.

Say that agent  $\alpha$  is the proposer. It makes a take-it-or-leave-it offer,  $x_\alpha^o$ , to the other agent,  $\beta$ , about how much agent  $\beta$ 's payoff will be if they pool.<sup>2</sup> Agent  $\beta$  can then accept or reject. If agent  $\beta$  accepts the offer, the agents pool and use agent  $\alpha$ 's solution to the joint problem. Agent  $\beta$ 's payoff is  $x_\alpha^o$  as proposed and agent  $\alpha$  gets the rest of the

---

<sup>2</sup>We allow an agent to make a negative "unacceptable" offer which signals that it does not want to coordinate or implement the joint solution.

value of the solution:  $v_\alpha^{\text{joint}} - x_\alpha^o$ . If agent  $\beta$  rejects, both agents implement their own computed solutions to their own individual problems, in which case agent  $\alpha$ 's payoff is  $v_\alpha^\alpha$  and agent  $\beta$ 's payoff is  $v_\beta^\beta$ . The payoffs are presented in Table 1.

Agent	Payoff if the offer is accepted	Payoff if the offer is rejected
$\alpha$	$v_\alpha^{\text{joint}} - x_\alpha^o$	$v_\alpha^\alpha$
$\beta$	$x_\alpha^o$	$v_\beta^\beta$

Table 1: If agent  $\alpha$  makes an offer,  $x_\alpha^o$ , to agent  $\beta$ , then agent  $\beta$  has the choice of either accepting or rejecting it. The payoffs for the agents in either situation are listed in the table above.

Before the deadline, the agents may or may not know which one of them is the proposer. In any case, if the agents agree to implement the joint solution, the joint solution computed by the proposer is used. In our model the probability that agent  $\alpha$  will be the proposer is  $P_{prop}$ , and this is common knowledge. When agents reach the bargaining stage, each agent's strategy is captured by an *offer-accept vector*. An offer-accept vector for agent  $\alpha$  is  $OA_\alpha = (x_\alpha^o, x_\alpha^a) \in \mathbb{R}^2$ , where  $x_\alpha^o$  is the amount that agent  $\alpha$  would offer if it were the proposer, and  $x_\alpha^a$  is the minimum value it would accept if agent  $\beta$  made the proposal. The offer-accept vector for agent  $\beta$  is defined similarly.

The agents' strategies incorporate actions from both the computation part and the bargaining part of the game. For the computation part of the game, an agent's strategy is a mapping from the state of computation to the next computation action (that is, selecting which solution  $z$ ,  $z \in \{\alpha, \beta, \text{joint}\}$  to compute another time step on—in words, whether to compute on the agent's own problem, the other agent's problem, or the joint problem).

**Definition 15** A computation strategy for agent  $\alpha$  with deadline  $T$  is

$$S_\alpha^D = (S_\alpha^{D,t})_{t=0}^{T-1}$$

where

$$S_\alpha^{D,t} : \Theta_\alpha(t) \rightarrow \{a^\alpha, a^\beta, a^{\text{joint}}\}$$

is a mapping from a computation state at time  $t$ ,  $\theta_\alpha(t) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$ , to a computation action  $a^z$  where  $a^z$  is the action of computing one time step on the solution for problem  $z \in \{\alpha, \beta, \text{joint}\}$ . The computation strategy of agent  $\beta$ ,  $S_\beta^D$ , is defined analogously.

In a deterministic setting, taking a computation action causes the agent to move into a specific state of computation. However, in the stochastic setting, if an agent is in a certain state of computation at time  $t$ , then the action of computing on a problem will cause the agent to be in any one of several states of computation at time  $t + 1$ . The probability with which an agent will enter into a specific state of computation is determined by the performance profiles.

At the deadline,  $T$ , each agent has to decide on its offer-accept vector. Therefore, the strategy at time  $T$  is a mapping from the state of computation at time  $T$  to an offer-accept vector.

**Definition 16** A bargaining strategy for agent  $\alpha$  with deadline  $T$ ,

$$S_\alpha^B : \Theta_\alpha(T) \rightarrow \mathbb{R}^2$$

is a mapping from a state of computation at time  $T$ , to an offer-accept vector,  $(x_\alpha^o, x_\alpha^a)$ . The bargaining strategy of agent  $\beta$ ,  $S_\beta^B$ , is defined analogously.

An agent's strategy consists of a computation strategy and a bargaining strategy.

**Definition 17** A strategy for agent  $\alpha$  with deadline  $T$  is

$$S_\alpha = (S_\alpha^D, S_\alpha^B).$$

A strategy for agent  $\beta$ ,  $S_\beta$  is defined analogously.

Our analysis will also allow *mixed strategies*. A mixed strategy for agent  $\alpha$  is  $\tilde{S}_\alpha = (\tilde{S}_\alpha^D, \tilde{S}_\alpha^B)$  where  $\tilde{S}_\alpha^D$  is a mapping from a computation state  $\theta_\alpha(t)$  to a probability distribution over the set of computation actions  $\{a^\alpha, a^\beta, a^{\text{joint}}\}$ . We let  $p^\alpha$  be the probability that an agent takes action  $a^\alpha$ ,  $p^\beta$  be the probability that an agent takes action  $a^\beta$ , and therefore,  $1 - p^\alpha - p^\beta$  is the probability that an agent takes action  $a^{\text{joint}}$ . The mixed bargaining strategy,  $\tilde{S}_\alpha^B$ , is a mapping from a computation state  $\theta_\alpha(T)$  to a probability distribution over offer–accept vectors.

There are many parameters to the mechanism that determine what strategies agents will choose to follow in equilibrium. These parameters include

- Which agent is the proposer,
- Whether the proposer is known in advance,
- Whether the deadlines are common knowledge, and
- Whether the performance profiles are stochastic or deterministic.

For example, if an agent knows that it will never be able to make an offer, then it has a dominant strategy.

**Theorem 1** *If an agent  $i$  with deadline  $T_i$  knows that it will not make an offer then it has a dominant strategy of computing only on its own problem, and accepting any offer  $x$  such that  $x \geq E[v_i(T)]$ . If the performance profile does not flatten before the deadline, then this is the unique dominant strategy.*

Since one agent has a dominant strategy, the other agent must find a best response strategy. The best response strategy depends on the performance profiles and the knowledge about the deadlines.

**Theorem 2** *Assume that all the agents' performance profiles are deterministic, agent  $\alpha$  is the proposer and agent  $\alpha$  knows agent  $\beta$ 's performance profiles. Then, there exists a Nash equilibrium where agent  $\beta$  follows its dominant strategy, and agent  $\alpha$  computes solely on its own problem or solely on the joint problem.*

If at least one performance profile is stochastic, or the deadlines are not known with certainty then the strategy of the proposing agent can be more complex. It may compute on any of the problems and may switch from one problem to another as it gathers results.

We do not assume that the performance profiles are common knowledge. However, it is required that at least agent  $\alpha$  can observe the performance profiles for agent  $\beta$ . Agent  $\beta$  does not need to know that agent  $\alpha$  can view its performance profiles. This knowledge does not change agent  $\beta$ 's behavior as it has a dominant strategy.

We use a dynamic programming algorithm to determine agent  $\alpha$ 's best response to agent  $\beta$ 's strategy. The base case involves looping through all possible computation states  $\theta_\alpha(T)$  for agent  $\alpha$  at the deadline  $T$ . Each  $\theta_\alpha(T)$  determines a probability distribution over the set of nodes agent  $\beta$  reached by computing  $T$  time steps. For any offer  $x$  that agent  $\alpha$  may make, the probability that agent  $\beta$  will accept is

$$P_\alpha(x) = \sum_{\{n^\beta | n^\beta \text{ in subtree } \mathcal{T}^\beta(n_\alpha^\beta) \text{ at depth } T - \text{time}(n_\alpha^\beta) \text{ s.t. } V(n^\beta) \leq x\}} P(n^\beta)$$

The best offer,  $x_\alpha^o$ , that agent  $\alpha$  can make to agent  $\beta$ , given that  $\alpha$  is in the state of computation  $\theta_\alpha(T) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$  is

$$x_\alpha^o(\theta_\alpha(T)) = \arg \max_x [P_\alpha(x)(V(n_\alpha^{\text{joint}}) - x) + (1 - P_\alpha(x))V(n_\alpha^\alpha)].$$

We denote the optimal bargaining strategy, given the computation state  $\theta_\alpha(T)$  by

$$S_\alpha^{B*}(\theta_\alpha^T) = (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha)).$$

The expected utility to agent  $\alpha$  from following such a bargaining strategy while in computation state  $\theta_\alpha(T)$  is

$$E[\pi_\alpha((S_\alpha^D, S_\alpha^{B*}(\theta_\alpha(T))), S_\beta)] = P_\alpha(x_\alpha^o)(V(n_\alpha^{\text{joint}}) - x_\alpha^o) + (1 - P_\alpha(x_\alpha^o))V(n_\alpha^\alpha).$$

It is possible to work backwards and to compute which computation action,  $a^z$ , is optimal if agent  $\alpha$  finds itself in computation state  $\theta_\alpha(t)$  at time  $t$  once the optimal offers have been computed for each final computation state. Let  $\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)$  denote the utility to agent  $\alpha$  of computing on problem  $z$  at time  $t + 1$ , given that at time  $t$  it is in computation state  $\theta_\alpha(t)$  and agent  $\beta$  is following strategy  $S_\beta$ . The expected value is

$$E[\pi_\alpha^D((a^z, \theta_\alpha(T-1)), S_\beta)] = \sum_{\theta_\alpha(T) \in \Theta_\alpha(T)} P(\theta_\alpha(T) | \theta_\alpha(T-1), a^z) E[\pi_\alpha((S_\alpha^D, S_\alpha^{B*}(\theta_\alpha(T))))]$$

and

$$E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)] = \sum_{\theta_\alpha(t+1) \in \Theta_\alpha(t+1)} P(\theta_\alpha(t+1) | \theta_\alpha(t), a^z) \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t+1)), S_\beta)]$$

for  $t < T - 1$  where  $P(\theta_\alpha(t) | \theta_\alpha(t-1), a^z)$  is the probability of reaching computation state  $\theta_\alpha(t)$  given that upon reaching computation state  $\theta_\alpha(t-1)$  the agent computes one step on problem  $z$ .

The optimal action in computation state  $\theta_\alpha(t)$  is

$$a^z(\theta_\alpha(t)) = \arg \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)].$$

The sequence of actions  $(a^z(\theta_\alpha(t)))_{t=0}^{T-1}$  is agent  $\alpha$ 's best-response computation strategy to agent  $\beta$ , and the offer-accept vectors  $(x_\alpha^o(\theta_\alpha(T)), 0)_{\theta_\alpha(T)}$  define its best-response bargaining strategy. Therefore,

$$S_\alpha = ((a^z(\theta_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha))_{\theta_\alpha(T)}).$$

The following algorithm computes the best-response strategy for agent  $\alpha$ .

**Theorem 3** *Algorithm 1 correctly computes a PBE strategy for agent  $\alpha$ .<sup>3</sup> Assume that the number of children of any node in  $\mathcal{T}_\alpha^\alpha$ ,  $\mathcal{T}_\alpha^\beta$  and  $\mathcal{T}_\alpha^{\text{joint}}$  is at most  $k$ . Algorithm 1 runs in  $O(k^T T^3)$  time.*

If the deadlines are only known with some probability, a similar algorithm can be used to determine the agents' strategies, where the probability of the game ending is included in the expected payoff calculation.

If the agents do not know who will be the proposer then neither agent may have a dominant strategy. In fact there exist instances where neither agent has a pure strategy.

**Theorem 4** *There exist instances (defined by the performance profiles) of the game that have a unique mixed strategy Nash equilibrium not no pure strategy Nash equilibrium.*

Lack of knowledge has another cost. The equilibrium outcome may not be Pareto efficient. Both agents would be better off from following strategies that were not in equilibrium.

**Theorem 5** *There exist instances (defined by the performance profiles) where the outcome of the Nash equilibrium is not Pareto efficient.*

<sup>3</sup>By keeping track of equally good actions at every step, Algorithm 1 can return all PBE strategies for agent  $\alpha$ . Again, the dominant strategy of agent  $\beta$  is to compute on its own problem (unless the performance profile flattens out after which it does not matter what agent  $\beta$  computes on.

---

**Algorithm 1**

---

**Require:**  $T \geq 0$

**for** each computation state  $\theta_\alpha(T)$  at time  $T$  **do**  
     $x_\alpha^o(\theta_\alpha(T)) \leftarrow \arg \max_x [P_a(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_a(x))V(n_\alpha^\alpha)]$   
**end for**  
**for** time  $t = T - 1$  down to 0 **do**  
    **for** each computation state  $\theta_\alpha(t)$  **do**  
         $a^z(\theta_\alpha(t)) \leftarrow \arg \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)]$   
    **end for**  
**end for**  
**return**  $((a^z(\theta_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha))_{\theta_\alpha(T)})$

---

### 5.1.2 Alternating Offers Bargaining Mechanism

In the alternating offers bargaining mechanism, the game is divided into multiple stages. In each stage both agents can make one computational action followed by one round of bargaining. A bargaining round consists of one agent making an offer  $x$  to the other agent. IF the offer is accepted, the game ends. The joint solution is used, the agent that accepted the offer has utility of  $x$  and the agent that made the offer has utility of the value of its joint solution minus  $x$ . If the proposal is rejected the game continues to the next stage. Both agents take parallel computing steps, but switch roles for the bargaining portion.

The actions and strategies in such a game can be defined formally.

**Definition 18** 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 computation 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 computation action and  $\text{res} \in \{\text{yes}, \text{no}\}$  is agent  $i$ 's response to the opposing agent's offer.

A history describes the computation state of both agents at time  $t$  and the offers and responses of both agents.

**Definition 19** 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 computation 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 20** A strategy for agent  $i$ ,  $S_i$ , is

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

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

In the previous section I described a setting where agents were restricted to one round of communication at the end of the computation phase. This restricted form had the advantage that agents did not leak information about their computation strategies while they computed. On the other hand, in our alternating offers model, each proposal and response provides information as to what problems the opponent has computed on and what values it has obtained for them. 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 computation actions, to guide its future computation actions, proposals and responses.

Each agent's beliefs consist of two parts which are interrelated. The first part is belief about *where* the other agent has devoted its computational resources. At time  $t$  there are  $\frac{(t+2)(t+1)}{2}$  ways that an agent could have divided its computation between the three problems.<sup>4</sup> The set of possible computation partitions at time  $t$  is denoted by

$$\text{Part}(t) = \{(t^\alpha, t^\beta, t^{\text{joint}}) \mid \sum_{z \in \{\alpha, \beta, \text{joint}\}} t^z = t\}$$

Each agent maintains a probability distribution over  $\text{Part}(t)$  which represents its beliefs about how the other agent has used its computational resources at time  $t$ . We let  $b_i(\bar{t})$  be the probability with which agent  $i$  believes the other agent has partitioned its computational resources as  $\bar{t} \in \text{Part}(t)$ . For the rest of this section we will discuss the beliefs that agent  $\alpha$  has. The beliefs for agent  $\beta$  are analogous.

Agent  $\alpha$  updates its belief  $b_\alpha(\bar{t})$  in two ways. The beliefs at time  $t$  must be consistent with those that were held at time  $t - 1$ . Therefore, we require

$$\begin{aligned} b_\alpha((t^\alpha, t^\beta, t^{\text{joint}})) &= P(a^\alpha) b_\alpha((t^\alpha - 1, t^\beta, t^{\text{joint}})) \\ &\quad + P(a^\beta) b_\alpha((t^\alpha, t^\beta - 1, t^{\text{joint}})) \\ &\quad + P(a^{\text{joint}}) b_\alpha((t^\alpha, t^\beta, t^{\text{joint}})) \end{aligned}$$

where  $P(a^z)$  is the probability that computation action  $z$  was performed by the opponent  $\beta$ .

The second way of updating  $b_\alpha(\bar{t})$  depends on the signal from agent  $\beta$ , i.e., agent  $\beta$ 's proposals and responses. Assume that agent  $\beta$  has made a proposal  $x$  at time  $t$ . We also assume that agent  $\beta$  is individually rational *in the sense that it would never knowingly make a proposal that, if accepted, would make it worse off than under no agreement*. Let  $t_\beta^{\text{joint}}$  be the number of time steps agent  $\beta$  has devoted to the joint problem, and let  $t_\beta^\alpha$  be the number of steps it has devoted to agent  $\alpha$ 's individual problem. Then, if agent  $\beta$  proposes an amount  $x$  it must be the case that

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

Once agent  $\alpha$  has observed  $x$  then it can update its beliefs,  $b'_\alpha(\bar{t})$ , by using Bayes Rule,

$$b'_\alpha(\bar{t}) = b_\alpha(\bar{t}|x) = \frac{P(x|\bar{t}) b_\alpha(\bar{t})}{\sum_{t' \in \text{Part}(t)} P(x|t') b_\alpha(t')}$$

The probability,  $P(x|t^*)$  for  $t^* \in \text{Part}(t)$  is determined from the performance profiles and agent  $\alpha$ 's own results from computing.

The second part of an agent's belief is over *what* solutions (of what value) the other agent has obtained for the different problems. Let  $P_\alpha^z(v^z|t)$  be agent  $\alpha$ 's belief that if  $t$  computation steps are devoted to problem  $z$  then the solution would have value  $v^z$ . In a deterministic setting  $P_\alpha(v^z|t)$  is either 0 or 1. However, in a stochastic setting, as agents compute on a problem, some of the uncertainty is resolved and the beliefs can be updated. What agents are

<sup>4</sup>This is the number of integer solutions to the problem  $x_1 + x_2 + x_3 = t$  for  $x_i \geq 0$ .

really interested in is the computation states in which the other agent may be. For example, agent  $\alpha$ 's belief that  $\beta$  is in computation state  $\theta_\beta(t) = \langle n_\beta^\alpha, n_\beta^\beta, n_\beta^{\text{joint}} \rangle$  is

$$B_\alpha(\theta_\alpha(t)) = b_\alpha(\bar{t}) \prod_z P_\alpha^z(V(n_\alpha^z) | t^z)$$

where  $\bar{t} = (t^\alpha, t^\beta, t^{\text{joint}})$ ,  $\text{time}(n_\beta^\alpha) = t^\alpha$ ,  $\text{time}(n_\beta^\beta) = t^\beta$ , and  $\text{time}(n_\beta^{\text{joint}}) = t^{\text{joint}}$ .

Surprisingly, even though the alternating offers bargaining mechanism has a much larger action space than the single shot bargaining mechanisms, the equilibria are in some sense similar to the equilibria that occur in the single shot bargaining mechanism. In particular, no agent is willing to make a proposal until they have to, that is until they have reached their deadline. An early proposal signals too much information to the opposing agent about where an agent has computed and what values it has obtained. The bargaining game “reduces” to a single-shot game in that the agent with the earliest deadline at time  $T_i$  makes an offer only then. For the offer to be accepted, it must be larger than the value that the opposing agent can compute for itself by its own deadline [20].

### 5.1.3 Future Work

Both the single-shot and alternating offers bargaining mechanism have been studied using only a limited computation model. In the future I plan to investigate both settings using a costly computation model. I plan to ask the same questions;

- Do agents have dominant strategies?
- Are there pure strategy Nash equilibria?
- Is it possible to design special purpose algorithms to compute equilibrium strategies?
- When is the outcome Pareto efficient?
- Do the results from a costly computation model differ from the limited computation model?

## 5.2 One-to-Many Negotiation: Auctions

Auctions are commonly used mechanisms for situations where there is one buyer and multiple sellers or one seller and multiple buyers. In this section I will focus only on settings where there is one seller and multiple buyers. There are many different auction mechanisms, but we will discuss the standard ones. We make the usual assumptions that the bidders have quasilinear preferences. That is, a loser's utility is zero ( $u_i = 0$ ) and a winner's utility is the item's value minus what he has to pay in the auction (that is  $u_i = v_i - p_i$ ).

In an *English auction* each bidder is free to raise its bid. When no bidder is willing to increase the bid further, the auction ends with the item being allocated to the agent with the highest bid. That agent pays the amount of its bid. For rational agents there is a dominant bidding strategy. Agents keep bidding some small amount  $\epsilon$  more than the previous high bid until they reach their valuation. They stop bidding at that point.

In a *first-price sealed-bid auction* each agent submits one bid without knowing the other agents' bids. The highest bidder wins the item and pays the amount of her bid. There is no dominant bidding strategy for rational agents. An optimal strategy depends on the bids of the other agents.

In a *Dutch auction* the auctioneer lowers the price until some bidder takes the item at the current price. Again, there is no dominant bidding strategy. In fact, the Dutch auction is strategically equivalent to the first-price sealed-bid auction.

The fourth auction type is the *Vickrey auction* or *second-price sealed-bid auction*. Each bidder submits one bid without knowing what the others' bid. The highest bidder wins the item but pays the amount of the second highest bid. For rational agents there is a dominant strategy which is for each agent to bid its true valuation.

The Dutch and first-price sealed-bid auctions are strategically equivalent. If the bidders are risk neutral as we assume, then the English and Vickrey auctions are strategically equivalent [17].

In auctions where multiple distinguishable items are sold, bidding strategies for agents can become much more complicated. A bidder's valuation for a combination of items might not be the sum of the individual items' valuations. It may be greater, smaller, or the same.

In traditional auction formats where items are auctioned separately, in order to decide how much to bid on an item, an agent needs to estimate which other items it will receive in the other auctions. This can lead to inefficient allocations where bidders do not get the combinations they want or else get combinations that they do not want [38].

*Combinatorial auctions* can be used to overcome these deficiencies. In a combinatorial auction, bidders may submit bids on combinations of items which allows the bidders to express complementarities between items. Based on the bids on the combinations of items, or *bundles*, the goods are *allocated* to the agents. Let  $X = \{x_1, \dots, x_n\}$  be a set of items. A bundle is a subset of the items, for example,  $\{x_1\}$  or  $\{x_1, x_n\}$ . An allocation of items among  $A$  agents is  $Y = (y_1, \dots, y_A)$  where  $y_i \subseteq X$ ,  $\cup_{i=1}^A y_i \subseteq X$  and  $y_i \cap y_j = \emptyset$  for  $i \neq j$ . The *generalized Vickrey auction* (GVA) is a combinatorial auction where the payments are structured so that each bidder's dominant strategy is to bid truthfully. It is an application of the Clarke tax mechanism to auctions [4].

### 5.2.1 The Generalized Vickrey Auction (GVA) Mechanism

The generalized Vickrey auction (GVA) works in the following manner.

1. Each agent declares a valuation function. So  $v_i(Y)$  is agent  $i$ 's valuation for allocation  $Y$ .
2. The GVA chooses an optimal allocation  $Y^*$  that maximizes the sum of all the agents' declared valuations.
3. The GVA announces the winners and their payment  $p_i$ :

$$p_i = \sum_{j \neq i} v_j(Y_{\sim i}^*) - \sum_{j \neq i} v_j(Y^*)$$

where  $Y_{\sim i}^*$  is the allocation that maximizes the sum of all agents' valuations assuming that agent  $i$  did not participate.

Under the usual assumption that each agent has quasilinear preferences  $u_i(Y) = v_i(Y) - p_i$ , the utility of bidder  $i$  in the GVA is

$$u_i(Y^*) = v_i(Y^*) - p_i = v_i(Y^*) + \sum_{j \neq i} v_j(Y^*) - \sum_{j \neq i} v_j(Y_{\sim i}^*).$$

The GVA has several nice properties for rational agents. First, if the agents have quasilinear preferences, the GVA is incentive compatible. The dominant strategy for rational agents is to bid their true valuations for the bundles of items. Second, the GVA is Pareto efficient. There is no other way to allocate the items (and compute payments) that would make some agent better off without making some other agent worse off. Finally, it is individually rational for agents to participate. An agent's utility obtained from participating in the GVA is never lower than if it had not participated (that is, the agent will never end up paying more for its bundle of items than its true valuation for the bundle).

## 5.2.2 Strategies for Agents

A strategy for an agent is composed of two interrelated components – the computation component and the bidding component. What an agent bids depends on the solutions it has obtained for its valuation problems, and what an agent computes on depends partially on what bids it has observed.

Let  $A$  be the set of agents participating in the auction and  $G$  be the set of bundles. Let  $\text{Act}(A, G)$  be the set of computation actions where  $\text{act}_i^g \in \text{Act}(A, G)$  is the action of (possibly choosing a random number and then) taking one computation step on agent  $i$ 's problem for bundle  $g$ . Let  $D$  be the deadline at which point the agent must stop computing. In the costly computation model  $D = \infty$  while in the free but limited computation model  $D$  is finite. There is also a deadline  $T$  for single shot auctions. The deadline  $T$  specifies the time when the auction closes (that is, by when the agents must submit their final bid). An agent's computing deadline,  $D$ , may occur before, after, or at the auction closing deadline  $T$ . We do not require that agents compute every step. Instead, they may decide to pass on computing. In the costly computation model agents do not incur any cost when they decide not to compute on a problem, however, in the free but limited computation model, a skipped step is still counted as though a computation step was used. We denote by  $\emptyset$  the act of taking no computation step. While bidding, agents may either send a message stating how much their bid is, or that they no longer wish to participate in the auction. We require agents to send "withdraw" messages in sequential auctions in order to differentiate between not increasing a bid, and leaving the auction.

The auction mechanism determines the agents' strategies.

**Definition 21** A strategy for agent  $i$  in a sequential auction is

$$S_i = ((\sigma_i^d(t), \sigma_i^b(t)))_{t=0}^D$$

with computation component

$$\sigma_i^d(t+1) : \theta_i(t) \times B_i(t) \rightarrow \text{Act}(A, G)$$

where  $B_i(t)$  is the list of bids observed by agent  $i$  up to time  $t$ . The bidding component is

$$\sigma_i^b(t+1) : \theta_i(t+1) \times B_i(t) \rightarrow \mathbb{R}^{2^M-1} \cup \{\text{withdraw}\}.$$

In a single-shot setting the strategy is slightly different. The auction ends at some time  $T$  at which point agents submit a single bid. An agent never has the opportunity to observe the other agents' bids.

**Definition 22** A strategy for agent  $i$  in a single-shot auction which closes at time  $T$  is

where the computation component is

$$S_i = ((\sigma_i^d(t))_{t=0}^D, \sigma_i^b)$$

$$\sigma_i^d(t+1) : \theta_i(t+1) \rightarrow \text{Act}(A, G)$$

and the bidding component

$$\sigma_i^b : \theta_i(T) \rightarrow \mathbb{R}^{2^M-1}.$$

Table 2 lists the results of the analysis of the four single item auctions and the one multiple item auction. In auctions where fully rational agents would counterspeculate in equilibrium, it is not surprising that for computationally bounded agents, strong strategic computation occurs. The performance profile model provides the agents with a method to counterspeculate. What is surprising, however, is that the English and Vickrey auctions lose their dominant strategy properties if agents have costly computation but not when agents have limited computation. This illustrates how important it is to explicitly model agents computational capabilities and to model how the agents are bounded.

	Auction mechanism	Counterspeculation by rational agents?	Strategic computation?	
			Limited computation	Costly computation
Single item	<b>English</b>			
	deterministic performance profiles	no	<b>no</b>	<b>no</b>
	stochastic performance profiles	no	<b>no</b>	<b>yes</b>
	<b>Vickrey</b>			
	deterministic performance profiles	no	<b>no</b>	<b>no</b>
	stochastic performance profiles	no	<b>no</b>	<b>yes</b>
	<b>First Price</b>			
	deterministic performance profiles	yes	<b>weak only</b>	<b>weak only</b>
	stochastic performance profiles	yes	<b>yes</b>	<b>yes</b>
	<b>Dutch</b>			
	deterministic performance profiles	yes	<b>weak only</b>	<b>weak only</b>
	stochastic performance profiles	yes	<b>yes</b>	<b>yes</b>
Multiple items	GVA (deterministic performance profiles)	no	<b>weak only</b>	<b>weak only</b>
	GVA (stochastic performance profiles)	no	<b>yes</b>	<b>yes</b>

Table 2: *When does strategic computation occur? The answer to this question depends on whether computation is limited or costly and on the topology of the agents' performance profiles. It is independent of other dimensions such as whether agents are computing to improve or refine their valuations, whether agents are allowed to compute past the close of the auction or not, and whether agents can skip steps while computing or not.*

### 5.2.3 Future Work

So far I have focused on auctions where there is one seller and multiple buyers. I also plan to investigate what happens in reverse, or procurement auctions. In these settings there is one buyer (the auctioneer) and multiple sellers (the bidders). The buyer wishes to buy a set of items, with the constraint that it must get at least all the items in the set. If there is free disposal then the buyer may be ambivalent about acquiring more items. The sellers submit bids on how cheaply they are willing to sell a bundle of items to the buyer.

On the surface, auctions and reverse auctions appear to be similar. However, there are some noticeable differences. For example, while winner determination for both combinatorial auctions and reverse combinatorial auctions is *NP*-complete, reverse auctions with free disposal can be approximated while standard combinatorial auctions is nonapproximable [41]. Also, the communication complexity of auctions and reverse auction differs [27].

While it is possible that bidding strategies for computationally bounded agents in reverse auctions may be identical to those in standard auctions, given that there do exist difference in other aspects between the two types of auctions, I plan on investigating the problem in further detail. In particular I plan to ask and answer the questions;

- How does one model reverse auctions so as to incorporate the computational actions?
- Under what circumstances do agents have dominant strategies?
- Under what circumstances will strong and weak strategic computation occur?

### 5.3 Many-To-Many Negotiation: Exchanges

Exchanges can be viewed as generalizations of auctions in that there can be multiple buyers and multiple sellers. Like in auctions, I plan to study exchanges where agents have both limited and costly computation. I plan to try to determine under what circumstances strategic computation will occur and what impact that has on the efficiency of the outcome/final allocation.

## 6 Mechanism Design for Computationally Bounded Agents

So far I have proposed to work with standard mechanisms and determine their strengths and limitations when agents have bounds on their computation. However, it is known that many standard mechanisms lose some of their desirable properties in a limited computational setting. For example, both the English and Vickrey auctions lose their dominant strategy proofness if agents have a cost associated with computing. More generally, it has been shown that classical results such as the Revenue Equivalence Theorem and the Revelation Principle no longer necessarily hold in computational settings [21, 30].

I propose that what is needed is a set of *new design principles* for mechanisms, designed for computationally bounded agents. As it has been shown, the model of bounded rationality is an important component in determining the optimal strategies for agents. Therefore, I believe that it should be part of any mechanism that is being designed for bounded rational agents.

I plan to concentrate on one-to-many mechanisms and focus on the following topics.

### 6.1 What is a mechanism?

The first question that must be addressed is “What is a mechanism for resource bounded agents?” In particular, what must each agent tell the mechanism. Must each agent reveal its results from computing so far, its performance profile databases, its cost functions? This must still be determined. Another question to be addressed is “What should the mechanism specify?”. Based on what agents report should it only specify an allocation and payments, or should it also be involved in specifying computation policies.

Another important question is “Is there a similar formulation to the Revelation Principle for computationally bounded agents?” The Revelation Principle is an important tool in that it says that one need only restrict one's attention to very simple types of mechanisms, those in which all agents are asked to reveal their type. The type in an auction setting is the agent's valuation. For computationally limited agents it is not clear what a type is. Is it the valuation it has obtained, the valuation it could obtain if following a specific computation policy, or something else instead? It may be the case that for computationally limited agents, it is better not to use a direct mechanism, that iterative mechanisms provide better outcomes. This must be investigated further.

### 6.2 Properties of Mechanisms

There are several desirable properties of mechanisms that designers often want to capture. These include

- incentive compatibility
- individual rationality
- Pareto efficiency
- budget balance

Some questions that I plan to ask and investigate are:

- Is it possible to design mechanisms for computationally bounded agents that exhibit some subset of the properties listed above?
- Do the bounds on the computational resources of the agents mean that it is not possible to obtain certain properties?
- Are there other properties that a mechanism for computationally bounded agents should exhibit? For example, it may be desirable to have mechanisms where the waste of computational resources is kept to a minimum.
- There are impossibility results that state what is achievable or not achievable for noncomputationally bounded agents in different settings (for example, Gibbard-Satterthwaite Impossibility Theorem [42, 6] and the Myerson-Satterthwaite theorem [25]). These impossibility results will still hold in the computationally bounded setting. However, computation may introduce further complications. Are there tradeoffs that can be made so as to be able to achieve most properties most of the time?

### 6.3 Measuring Mechanisms

A natural question to ask is whether the cost or limit on computing results in a loss in efficiency. However, efficiency is hard to compare in such settings. The Vickrey auction is efficient in the sense that it always allocates the item to the bidder with the highest valuation. However, an agent who *might* have been able to obtain the highest valuation via computing, may have used its limited computing on a different problem, thus causing a different agent to have the highest valuation and win the auctions. This outcome is still efficient *given how agents computed*. This overlooks the computational issues in an unsatisfying way. This suggests that Pareto efficiency may not be the right measure to use in the context of computationally limited agents. Is there a better measure?

#### 6.3.1 The Social Cost of Selfish Computing

Instead of looking at efficiency, I propose to use social welfare as the measure. I want to know how letting agents freely choose their own computing strategies impacts the social welfare of the set of all bidders. In particular, I compare the highest achievable social welfare to the lowest social welfare achievable in any Nash equilibrium.

When we determine the highest achievable social welfare we optimistically assume that there is a global controller who imposes each agent’s computing strategy (so as to maximize social welfare). The controller has full information about all performance profiles, deadlines, cost functions, and intermediate results of computing. In the bidding stage agents are free to bid as they wish, but their goal is still to maximize their own utility, and so they bid truthfully in the Vickrey auction, given the valuations they have obtained under the enforced computing policy.

**Definition 23** *Let  $o^*$  be the outcome that is reached if the global controller dictates computing policies to all agents, and agents are free to bid in the Vickrey auction.*

On the other extreme, we are interested in what happens when agents are free to choose to follow any computing and bidding strategy. Let  $\text{NashEq}$  be the set of Nash equilibria in that game. We now define what is meant by the worst-case Nash equilibrium.

**Definition 24** *The worst case Nash equilibrium is*

$$NE = \arg \min_{s \in \text{NashEq}} SW(o(s)).$$

We use the following ratio to see how much letting agents choose their own computing strategies reduces the social welfare.

**Definition 25** *The miscomputing ratio is*

$$R = \frac{SW(o^*)}{SW(o(NE))}.$$

This ratio isolates the impact of selfish computing from the traditional strategic bidding behavior in auctions. This is because in both the coordinated and uncoordinated scenario, the agents bid based on self-interest.

Using this measure it is possible to compare different settings.

**Theorem 6** *Assume there are  $n$  bidders in a Vickrey auction, and that each bidder has free but limited computing. Then, the miscomputing ratio  $R$  can be infinity.*

Similarly, in settings where agents have cost associated with computing, the miscomputing ratio can be arbitrarily large.

**Theorem 7** *Assume there are  $n$  bidders in a Vickrey auction. Assume that each bidder  $i$  has costly, unlimited computing. Then, the miscomputing ratio  $R$  can be infinity.*

These results are quite pessimistic. They state that permitting agents to freely select their own computing strategies can result in large reductions in social welfare. However, there are certain restrictions that a system designer can use in order to motivate agents to follow desirable strategies. In particular, by carefully tailoring the cost functions of agents, it is possible to motivate agents to select strategies that maximize social welfare.

There are several directions in which this work can lead. A further investigation of the design of cost functions is warranted, as it may be possible to develop general purpose cost functions for agents. A second direction that might be fruitful is to use information from the performance profile trees to decide which mechanism should be used. For example, there are many ways to set up an auction for a single item. However, given the performance profiles of the agents and their deadlines or cost functions, one mechanism might result in a lower miscomputing ratio than others. This idea must be investigated further.

## 7 Experiments

So far I have proposed to theoretically analyze different settings where agents with bounded computational resources interact. In this section I will describe some experiments I plan to conduct to see how my approaches work in a more practical setting.

In particular I wish to show:

- The performance profile tree is a feasible model. For a theoretical model the performance profile tree can be used to model all information needed by an agent to make fully normative decisions. However, in real-world settings it is not practical to have a model where computation can be conditioned on all aspects of the problem. I plan to do some studies to try and determine what are the most important aspects to condition on for certain problems such as scheduling and TSP. I also plan to show that the performance profile tree can be easily used by agents.
- I plan to show that my approach is general. That is, agents with any type of bounded computation and problems can use the performance profile trees to aid in negotiation. In particular, I hope to show that “off the shelf tools” such as general purpose schedulers can be easily incorporated into negotiation settings with little modification required.

- I plan to run some experiments to see what impact (if any) strategic computation will have on outcomes for the agents. In particular, I want to see if strategic computation is an artifice of the analysis or something that will occur in the “real world”.
- To aid in the experiments I am thinking of designing and building tools for visualizing the negotiations and computations of the agents in a distributed manner.

I have three different experiments that I plan to conduct.

1. A traveling salesman problem where agents place bids on bundles of cities they wish to service. I will make the following assumptions:
  - The seller does not need to have all points delivered to.
  - The price of a bundle of items is simply the cost of travelling to each city in the bundle.
  - The bidders each have a finite number of possible starting positions (discrete). Each bidder knows where it will start its route, the other bidders have a probability distribution over where their opponents will start.
  - There will be prebundling to cut down on the number of problems each agent needs to compute. Cities will be placed into bundles in a random fashion. Only a small number of bundles will be put up for auction. That is, agents are limited in the number of things they can place bids on.

A combinatorial auction will be run, to auction off the deliveries. This experiment will (hopefully) show that it is possible to incorporate computational actions with bidding actions in a reasonable way.
2. A single item English auction where the valuation of the item is equal to the value of a schedule that each agent organizes.
  - (a) I plan to use schedulers developed by other researchers to generate data for the performance profile trees.
  - (b) The goal of this experiment is to demonstrate that it is possible to incorporate off-the-shelf tools (eg. schedulers) into the computational/negotiation model developed in my thesis.
3. A reverse auction where the problem to be solved is a multiagent vehicle routing problem. In particular, using real world data, I plan to implement a system where agents are participating in a multi-item reverse auction.
  - (a) Prebundling will be used to simplify the winner determination problem.
  - (b) The goal of this experiment is to see whether strategic computing is an artifice of the analysis or something that will occur in real world settings.

## 8 Summary and Status

To summarize, I propose to make the following main contributions in my thesis.

1. Provide a fully normative model of computation control for computationally bounded agents.
2. Provide a model for incorporating agents’ computational actions into their strategies so that strategic interactions between agents can be modeled game theoretically. Define and use deliberation equilibria and strategic computation.

3. Study different mechanisms to see how the equilibrium differs if agents are fully rational or are computationally bounded.
4. Provide tools and measures for designing mechanisms for agents that specifically take into account the restrictions on their computational capabilities, including techniques for measuring and comparing mechanisms for bounded agents.
5. Provide experimental results that show that this approach is feasible to use in realistic settings.

Table 3 shows the status of the dissertation and the schedule that I plan to follow.

Task	Completed?	Date to be Completed
Performance Profile Tress	√	
<b>Mechanisms</b>		
Bargaining - Limited	√	
Bargaining - Costly		May 2002
Auctions - Limited	√	
Auctions - Costly	√	
Auctions - Reverse		
Exchanges		
<b>Mechanism Design</b>		
Definitions and Properties		Fall 2002/Spring 2003
Measuring Mechanisms		Fall 2002
<b>Experiments</b>		Summer 2002 - Spring 2003
<b>Writing Up</b>		Winter 2003

Table 3: Status

## References

- [1] Dilip Abreu and Ariel Rubinstein. The structure of Nash equilibrium in repeated games with finite automata. *Econometrica*, 56(6):1259–1281, 1988.
- [2] Eric B Baum and Warren D Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.
- [3] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.
- [4] E H Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.
- [5] Alan Garvey and Victor Lesser. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems*, 6:317–347, 1994.

- [6] A Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.
- [7] Itzhak Gilboa and Dov Samet. Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, pages 213–221, 1989.
- [8] Joshua Grass and Shlomo Zilberstein. A value-driven system for autonomous information gathering. *Journal of Intelligent Information Systems*, 14:5–27, 2000.
- [9] Eric Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.
- [10] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.
- [11] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–444, Seattle, Washington, July 1987. American Association for Artificial Intelligence. Also in L. Kanal, T. Levitt, and J. Lemmer, ed., *Uncertainty in Artificial Intelligence 3*, Elsevier, 1989, pps. 301–324.
- [12] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, 1987.
- [13] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 111–116, St. Paul, MN, August 1988. Morgan Kaufmann, San Mateo, CA.
- [14] Philippe Jehiel. Limited horizon forecast in repeated alternate games. *Journal of Economic Theory*, 67:497–519, 1995.
- [15] Noa Kfir-Dahav, Dov Monderer, and Moshe Tennenholtz. Mechanism design for resource bounded agents. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS)*, pages 309–315, Boston, MA, 2000.
- [16] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.
- [17] David M Kreps. *A Course in Microeconomic Theory*. Princeton University Press, 1990.
- [18] Kate Larson and Tuomas Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001.
- [19] Kate Larson and Tuomas Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001. Short early version appeared in the Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 48–55, Austin, TX, 2000.
- [20] Kate Larson and Tuomas Sandholm. An alternating offers bargaining model for computationally limited agents. In *First International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 2002.
- [21] John Ledyard. Incentive compatibility. In John Eatwell, Murray Milgate, and Peter Newman, editors, *Allocation, Information and Markets*, pages 141–151. W W Norton, 1989.

- [22] Daniel Lehmann, Lidian Ita O’Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 96–102, Denver, CO, November 1999.
- [23] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.
- [24] 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(1–3):161–205, 1992.
- [25] Roger Myerson and Mark Satterthwaite. Efficient mechanisms for bilateral exchange. *Journal of Economic Theory*, 28:265–281, 1983.
- [26] John Nash. Equilibrium points in n-person games. *Proc. of the National Academy of Sciences*, 36:48–49, 1950.
- [27] Noam Nisan. The communication complexity of combinatorial auctions, 2001. Draft. Institute of Computer Science, The Hebrew University of Jerusalem. July 22nd.
- [28] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 242–252, Minneapolis, MN, 2000.
- [29] Christos H. Papadimitriou and Mihalis Yannakakis. On complexity as bounded rationality. In *STOC-94*, pages 726–733, 1994.
- [30] David Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, May 2001.
- [31] David C Parkes. Optimal auction design for agents with hard valuation problems. In *Agent-Mediated Electronic Commerce Workshop at the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
- [32] Ariel Rubinstein. *Modeling Bounded Rationality*. MIT Press, 1998.
- [33] Stuart Russell. Rationality and intelligence. *Artificial Intelligence*, 94(1):57–77, 1997.
- [34] Stuart Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 1:1–36, 1995.
- [35] Stuart Russell and Eric Wefald. *Do the right thing: Studies in Limited Rationality*. The MIT Press, 1991.
- [36] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.
- [37] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 256–262, Washington, D.C., July 1993.
- [38] Tuomas Sandholm. Issues in computational Vickrey auctions. *International Journal of Electronic Commerce*, 4(3):107–129, 2000. Special Issue on Applying Intelligent Agents for Electronic Commerce. A short, early version appeared at the Second International Conference on Multi-Agent Systems (ICMAS), pages 299–306, 1996.

- [39] Tuomas Sandholm and Victor R Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997. Special issue on Economic Principles of Multiagent Systems. Early version appeared at the International Joint Conference on Artificial Intelligence (IJCAI), pages 662–669, 1995.
- [40] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1102–1108, Seattle, WA, 2001.
- [41] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. Winner determination in combinatorial auction generalizations. In *AGENTS-01 Workshop on Agent-Based Approaches to B2B*, pages 35–41, Montreal, Canada, May 2001.
- [42] M A Satterthwaite. Strategy-proofness and Arrow’s conditions: existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–217, 1975.
- [43] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.
- [44] Shlomo Zilberstein, François Charpillet, and Philippe Chassaing. Real-time problem solving with contract algorithms. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1008–1013, Stockholm, Sweden, 1999.
- [45] Shlomo Zilberstein and Stuart Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.