Mechanism Design for Computationally Limited Agents

Kate Larson

August 2004

CMU-CS-04-152

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:
Tuomas Sandholm, Chair
Avrim Blum
Andrew Moore
Craig Boutilier, University of Toronto
Mark Satterthwaite, Northwestern University

Copyright © 2004 Kate Larson

This research was sponsored by the National Science Foundation under grant numbers IIS-0234694, IIS-0234693, and IIS-0121678. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the NSF, the U.S. government or any other entity.

Keywords: Artificial Intelligence, Multiagent Systems, Game Theory, Mechanism Design, Bounded Rationality, Resource Bounded Reasoning

First of all I would like to thank my adviser Tuomas Sandholm for all his support and patience during my years as a graduate student. He has provided me with a model of what a first class researcher should be. Second, I would like to thank the members of my thesis committee, Avrim Blum, Andrew Moore, Craig Boutilier, and Mark Satterthwaite, for giving me valuable criticism, insight and guidance, which helped shape the content and presentation of this dissertation.

There are many people who have played significant roles in my graduate career. I would particularly like to thank Sherry May for suggesting I try out both graduate school and computer science. Without her initial encouragement I would not be where I am today.

I have enjoyed many hours discussing research problems with a large group of people. These discussions introduced me to new ideas and helped clarify various technical points. In particular, discussions with Vince Conitzer, Andrew Gilpin, Anton Likhodedov, Benoit Hudson, Shuchi Chawla, Jason Hartline, Marty Zinkevich, Paolo Santi, Felix Brandt and David Parkes have proved to be invaluable.

There are many people from whom I have benefited beyond academics. I thank Mukesh, Rajesh and Mahim for their friendship over the last few years. They have provided distractions when I needed to be distracted, ears when I needed to talk, and technical advice when I was baffled by my computer. I would be remiss if I forgot to thank Jessica for understanding what it is like to be a grad student, and Emily and Jenn for reminding me that there is a world outside of school.

My time in graduate school would not have been possible without the support of my family. I thank Mom and Dad for always being there when I needed to talk, for boosting my morale when I needed it, and for celebrating my successes when they occurred. I would also like to thank Mike. His pride in my accomplishments has meant more to me than I can ever express.

Last, but not least, I thank Urs. He has attended numerous practice talks, proofread many papers, and acted as a sounding board for my ideas. More importantly, his belief in me has been unwavering, he has shared my frustration and joy, and the days are just more fun with him around.

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 both optimally determining their utilities from the game and determining 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 provide a theory of interaction for self-interested computationally bounded agents. This allows one to formally study the impact that bounded rationality has on agents' strategic behavior. It also provides a foundation for game-theory and mechanism design for computationally limited agents.

First, this thesis introduces a model of bounded rationality where agents must compute in order to determine their preferences. The computing resources of the agents are restricted so that the agents must carefully decide how to best use their computation. I present a fully normative model of deliberation control, the *performance profile tree*. Not only does this structure provide full normativity in theory, but I also show that in real-world applications it improves deliberation control compared to other methods.

This thesis proposes explicitly incorporating the deliberation actions of agents into a game-theoretic framework. I introduce a new game-theoretic solution concept, the *deliberation equilibrium*. This provides one with an approach for understanding and analyzing the strategic use of computation. Using this approach I analyze different negotiation protocols for computationally limited agents. I study two different bargaining settings where agents try to reach an agreement on whether to coordinate their actions or act independently. I provide algorithms that agents can use to determine their optimal strategies (including computing actions). I also study the impact that computing limitations have on bidding agents in auctions, where agents must compute or gather information in order to determine their valuations for the items being auctioned. I show that commonly used auction mechanisms all suffer from agents having incentive to strategically deliberate, that is use computing resources in order to (partially) determine their competitors' valuations. This means that mechanisms which had dominant strategy equilibria for rational agents are no longer strategy-proof for computationally limited agents.

Finally, this thesis studies the problem of designing mechanisms specifically for computationally-limited agents. My goal is to build mechanisms which have good deliberative properties as well as good economic properties. I propose a set of properties that I believe that mechanisms should exhibit, but then show that it is impossible to design interesting mechanisms which satisfy all the properties. While this result is negative, in that it is an impossibility result, it does provide direction for future research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are currently witnessing the growth of a new paradigm of open computer systems, as embodied by the Internet. The heterogeneous participants in these systems (which we will call agents) have their own private information and goals which may not coincide with the goals of the system designer, nor is it reasonable to assume that a system designer is able to force the agents to behave in some socially optimal way. In particular, these agents are often *self-interested*, and will act in their own self-interest, irrespective of the desires of others. Interesting applications with these properties include selfish routing scenarios [99], mobile ad-hoc networks [2, 33], as well as innumerable ecommerce applications. These scenarios can all be modeled as competitive multiagent systems.

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 stop them from optimally determining their utilities from the game and which strategies are best to follow. The game theoretic equilibria which describe how rational agents should behave do not, generally, remain the same for these (bounded-rational) agents. 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. The aim of this dissertation is to bridge this gap.

1.1 Thesis Statement

In this thesis we are trying to understand the impact that computational limitations have on the interactions of agents in multiagent environments. In particular, we are interested in the strategic behavior of computationally limited agents in different market mechanisms, in order to understand how these agents differ from the classic, fully-rational, agents and whether it is desirable and possible to design market mechanisms which take into account the limitations of the participating agents.

The thesis statement is:

> By using a *fully normative model of bounded rationality* it is possible to incorporate agents' deliberation actions into game theoretic settings.
>
> - This allows us to formally study the impact that limited deliberation resources has on agents' strategic behavior.
> - This provides a foundation for game theory and mechanism design for computationally limited agents.

1.2 Approach

This thesis has been heavily influenced by the ideas of Herbert Simon. As Simon pointed out, real economical players have limited time and powers of deliberation. He proposed the study of bounded rationality to investigate

> "... the shape of a system in which effectiveness in computation is one of the most important weapons of survival." [113]

Additionally, in correspondence with Ariel Rubinstein, Simon said

> "In my version of bounded rationality we look for answers to questions like:... What are the economic consequences of participants using certain procedures and not others? In what respects are current economic models deficient in the assumptions they make about reasoning procedures?" (February 7, 1997) [101].

Our approach develops and combines ideas from both a resource-bounded reasoning framework and a game theoretic framework. We model agents as being deliberative, in that at the meta-level they have to carefully reason about their preferences and goals. We assume that the resources used in this reasoning process are limited, forcing the agents to carefully weigh their alternatives and make tradeoffs concerning how they will deliberate.

We explicitly model the agents' deliberation decisions in a game-theoretic framework. This allows us to rigorously study how computational limitations affect the strategic behavior of agents as they interact with others. In particular, it provides us with a systematic way to compare computationally limited agents with fully rational agents, as well as providing us with a foundation for designing negotiation protocols explicitly for these agents.

1.2.1 Motivating Example

To make the presentation more concrete, we now discuss an example domain where our methods are needed. We use this example throughout the thesis.

Consider a distributed vehicle routing problem with two geographically dispersed dispatch centers that are self-interested companies (Figure 1.1) [106] [109]. 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 problem is to minimize transportation costs (driven mileage) while still making all of its deliveries while honoring the following constraints:

- 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 and maximum load volume constraint,
- Each vehicle has a maximum route length (prescribed by law).
- Each delivery has to be included in the route of some vehicle.

This problem is $\mathcal{NP}$-complete.

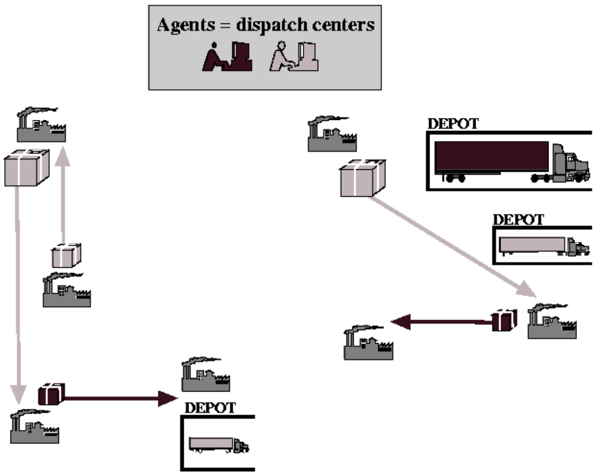


Figure 1.1: *Small example problem instance of the distributed vehicle routing problem.*

Assume that an additional task is to be allocated to a dispatch center via some auction mechanism. Before agents' can formulate and submit bids, they must first know how much they value the new task. This requires determining the cost of incorporating the new task into the current delivery schedule which potentially requires solving two $\mathcal{NP}$-complete problems (one without the new task, and one with the new task). The resources available to the agents to solve these problems may be limited. For example, the agents may have deadlines by which they require a solution or computing may be costly. Each agent must carefully consider the tradeoff they are willing to make on solution quality given the restrictions on their computing resources, as well as accounting for the fact that their computed solutions will also influence what sort of bids they can submit to the auctioneer.

1.3 Contributions

The key contributions of this thesis are

A normative model of bounded rationality. We present a model for a computationally-limited agent, endowing it with a fully normative deliberation control tool, the performance profile tree. This performance profile representation allows an agent to condition its deliberation decisions on any and all information deemed to be important. We show that this approach can be used in practice, leading to superior deliberation control decisions.

A formal game theoretic model for computationally limited agents. We propose incorporating the deliberation actions of agents in a game theoretic setting. We introduce the deliberation equilibrium solution concept and are able to understand and analyze the strategic use of computation.

Analysis of different negotiation mechanisms. Using the deliberation equilibrium solution concept, we analyze different standard negotiation mechanisms in order to understand the impact that computational limitations have on agents' strategies in such settings.

Mechanism design principles for computationally limited agents. We propose a set of desiderata for mechanisms designed for computationally limited agents. In particular, we argue that mechanisms should have good economic properties and good deliberative properties. We show that these desiderata are orthogonal, and that tradeoffs in design must be tolerated.

1.4 Guide to the Thesis

At a high level this thesis can be divided into two sections. In the first section we discuss bounded rationality and present our model of bounded rationality - in the form of computationally limited agents. This work is pertinent to both single-agent settings and multiagent settings. In the second section we move to multiagent settings. We show how we are able to model computationally limited agents in a game theoretic setting, and use this approach to study and design negotiation mechanisms.

Here we outline the chapters in the rest of thesis.

Section I: Computationally Limited Agents

Chapter 2 - Modeling Computationally Limited Agents. In this chapter we present our model of a computationally limited agent. We describe the role of computation, limitations on an agent's computing resources, and provide policies an agent can follow in order to effectively use its computing resources in the best possible way. In particular, we introduce the performance profile tree; a fully normative deliberation control method.

Chapter 3 - Improving Deliberation Control: Experimental Results. In this chapter we present a series of experimental results to show that the performance profile tree, introduced in Chapter 2, is a feasible approach for deliberation control for computationally limited agents, and that it outperforms other commonly used deliberation control methods.

Section II: Negotiating and Computationally Limited Agents

Chapter 4 - Game Theory and Mechanism Design. In this chapter we provide an overview of important game theory and mechanism design concepts.

Chapter 5 - Game Theory for Computationally Limited Agents. In this chapter we show how a model for a computationally limited agent can be placed in a game theoretic setting. We define strategies so that they include the deliberation actions of the agents as well as any other (i.e. negotiation) actions. We introduce the deliberation equilibrium solution concept and discuss new strategic behavior which arises with computationally limited agents.

Chapter 6 - One-to-One Negotiation: Bargaining. In this chapter we study bargaining protocols where two computationally limited agents must try to reach agreement on whether to coordinate their actions to execute a joint plan, or whether to act independently. We study the equilibria of different scenarios, and present algorithms that agents can use to determine their best strategies.

Chapter 7 - One-to-Many Negotiation: Auctions. In this chapter we study common auction protocols in order to understand the the implications that computational limitations have on bidding agents. We show that computationally-limited agents exhibit new forms of strategic behavior, and that auction mechanisms, which in classical settings have desirable game theoretic properties, lose these properties.

Chapter 8 - Mechanism Design for Computationally Limited Agents. In this chapter we look at the problem of designing allocation mechanisms for computationally limited agents in order to obtain desirable strategic properties, as well as desirable deliberative properties. We propose a set of reasonable desiderata which we believe that mechanisms should have, but show that in many situations these requirements are too strong.

Chapter 9 - The Social Cost of Selfish Computing. In this chapter we investigate what happens at a system wide level when the participants in a system are computationally limited. We introduce the *miscomputing ratio*, a way of measuring the system-wide impact of selfish computing. We show that by allowing agents to freely choose their computing strategies, the social welfare can be adversely affected.

Chapter 10 - Related Work. In this chapter we discuss other work that has been done in the intersection of artificial intelligence, theoretical computer science, and economic theory, and comment on how it is similar and how it differs from the work presented in this dissertation.

Chapter 11 - Conclusions. We conclude our work with a review of our contributions along with a discussion of future work in the area of mechanisms for computationally limited agents.

Much of the work in this thesis has appeared in the following papers:

- Kate Larson and Tuomas Sandholm, *Bargaining with limited computation: Deliberation equilibrium*, Artificial Intelligence, 132(2): 183-217. A short early version appeared in AAAI-2000 [59].

- Kate Larson and Tuomas Sandholm, *Costly valuation computation in auctions*, In the proceedings of the Eighth Conference of Theoretical Aspects of Knowledge and Rationality (TARK VIII), July 2001 [60].

- Kate Larson and Tuomas Sandholm, *Bidders with hard valuation problems*, In the proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), July 2002. (Poster paper) [62].

- Kate Larson and Tuomas Sandholm, *An alternating offers bargaining model for computationally limited agents*, In the proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002), July 2002 [61].

- Kate Larson and Tuomas Sandholm, *Miscomputing Ratio: The social cost of selfish computing*, In the proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), July 2003 [63].

- Kate Larson and Tuomas Sandholm, *Strategic deliberation and truthful revelation: An impossibility result*, In the proceedings of ACM Conference on Electronic Commerce (EC 04), May 2004 (short paper) [65].
- Kate Larson and Tuomas Sandholm, *Using performance profile trees to improve deliberation control*, In the proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-2004), July 2004 [66].
- Kate Larson and Tuomas Sandholm, *Experiments on deliberation equilibria in auctions*, In the proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), July 2004 [64].

Section I: Computationally Limited Agents

Chapter 2

Modeling Computationally-Limited Agents

> "Life is not long, and too much of it must not pass in idle deliberation how it shall be spent."
>
> **Samuel Johnson**

In many AI applications, bounded rationality is simply a feature that has to be dealt with. The realities of limited computational resources and time pressures caused by real-time environments mean that agents are not always able to optimally determine their best decisions and actions. The field of artificial intelligence has long searched for useful techniques for coping with this problem. Herbert Simon advocated that agents should forgo perfect rationality in favor of limited, economical reasoning. His thesis was that "the global optimization problem is to find the least-cost, or best-return decision, net of computational costs" [113]. In our work we follow the Simon thesis. We assume that agents are *deliberative* in that they determine how to best use their computational resources by "careful consideration with a view to decision" (Oxford English Dictionary).

In this chapter we present a model for computationally limited, deliberative agents, and show how these agents can effectively use their limited resources. The rest of this chapter is organized as follows. We first provide an overview of the AI literature which study the idea of agent rationality and address the problem of how agents should behave when they are bounded-rational. We then present our model of a computationally limited agent. We describe the role that computing plays in settings where agents are not endowed with full information about all possible states of the world (Section 2.2). We then describe the properties which define a computationally limited agent and explain the set of tools available to the agent so that it can effectively use its resources (Section 2.3). We finally introduce the *performance profile tree*, a fully normative deliberation control method (Section 2.4).

2.1 AI and Bounded Rationality

As a field, AI has long studied the problem of what does it mean to be *rational*. The earliest models of rationality in artificial intelligence had a logical definition [72, 81]. In this "logicist" approach to rationality it was assumed that an agent could be completely defined in terms of its beliefs and goals, and that an agent was rational if it satisfied one of its goals entailed by its beliefs. In particular, the emphasis of the research was on normative reasoning, while ignoring the possible complexity of implementing such an approach. In particular, McCarthy believed that it was important to first focus on "logical adequacy" before "heuristic adequacy" [103], and the goal of the agent was to reason using a consistent set of axioms, and generate provably correct plans

The decision sciences have heavily influenced models of rationality in artificial intelligence. Economic rationality provides formal tools for understanding reasoning, and gives a rigorous framework for analyzing utility and choice, in that it is possible to define an agent as being rational if it chooses the action with the maximum expected utility. A possible weakness of this approach is that it focuses entirely on *what* action was taken, and ignores the process of *how* an action was chosen. Good was the first to discuss the idea of explicitly integrating the cost of inference into the classical framework of rationality [38]. He distinguished between two types of rationality: *Type I* and *Type II*. His *Type I* rationality refers to the classical axiomatic approach to decision theory. It ignores any costs associated with the inference process. However, *Type II* rationality takes into the account the costs of reasoning. An agent is considered to be *Type I* rational if, in the end, the results satisfy the agent's preferences, while an agent is *Type II* rational if it maximizes its expected utility, taking into account the cost of deliberation. Herbert Simon was also very interested in the effects that deliberation costs had on rationality. He differentiated between *substantive* and *procedural* rationality. He defined substantive rationality to be the situation where the agent *does the right thing*, according to the agent's preferences. Procedural rationality, on the other hand, is similar to Good's *Type II* notion of rationality, in that a behavior is considered to be procedurally rational when it is the outcome of some strategy of reasoning.

Russell and his coauthors outlined four candidates for the formal definition of intelligence [102, 103, 105]:

1. Perfect rationality: the classic notion of rationality in economics, where an agent acts in such a way as to maximize its utility.

2. Calculative rationality: the notion of rationality where it is possible to achieve perfect rationality if the decision process is perfect and has infinite resources.

3. Meta-level rationality: the notion of rationality where the system optimizes over the meta-level decision making process. This is also *Type II* rationality.

4. Bounded optimality: the notion of rationality where an agent behaves as well as possible given its computational resources [47].

It has been argued that the goal of artificial intelligence is to design agents which are bounded optimal. However, we (as well as other researchers) believe that by focusing on meta-level rationality it is possible to design intelligent agents. Meta-level rationality has been well studied by artificial intelligence researchers, and considerable work has focused on developing *normative* models that prescribe how a bounded-rational agent *should* behave (see, for example [36, 46, 102]) as well as applying these approaches in different applications (see, for example [39, 44]). This is a highly non-trivial undertaking, encompassing numerous fundamental and technical difficulties. As a result, most of those methods resort to simplifying assumptions such as myopic deliberation control [6, 104, 105], conditioning the deliberation control on hand-picked features [104, 105], or resorting to asymptotic notions of bounded optimality [103].

Our work, as presented in the rest of this chapter, is also focused on techniques for achieving meta-level rationality. In particular, we assume that the agents have computational limitations which the meta-level accounts for when determining what the best action to take is. However, our work differs from previous research in this area, as we require a fully normative deliberation control procedure. This desire for full normativity arises since we wish to include meta-level rationality into multiagent systems, where the meta-level actions of one agent can influence the choice of actions of other agents.

2.2 The Role of Computation

In this section we describe the role that computation plays in settings where agents are not endowed with full information about all possible states of the world. In particular, we are interested in settings where agents must actively go out and gather information or solve nontrivial problems in order to decide how to best act.

We assume that an agent has some set of resources, T, that it can use to solve various problems. By allocating these resources on different problems an agent is returned information that it can use. We will assume that agents are using their computational resources to get *solutions* to various problems. Consider the following example. Assume that an agent is responsible for finding a route for a delivery truck, given a set of deliveries. While the agent can easily come up with a feasible delivery route (for example, by just making the deliveries in any random order), by allocating additional computing time to the problem, the agent may be able to come up with a shorter route. That is, by computing the agent changes its knowledge about the best route available.

We use a *feature tuple* to denote the information that the agent has about a problem after devoting t steps of computation to it.

Definition 1 (Feature Tuple). *The* feature tuple *for an agent i after allocating t computing steps on problem j is*

$$f_i^j(t) = (\mathrm{sol}(t), \mathrm{inst}) \in \mathcal{S} \times \mathcal{I}$$

where $\mathcal{S}$ is the set of features which describe the problem solutions, and $\mathcal{I}$ is the set of features which describe the different problem instances with which the agent may be working with. We denote the set of all possible feature tuples for problem j after agent i has allocated t computing steps on it by $\mathcal{F}_i^j(t)$. This set may contain more than a single feature tuple due to different problem instances as well as nondeterminism in the algorithms.

The feature tuple differentiates between features of the solution and features of the problem instance. To illustrate the difference, consider again, a traveling salesman

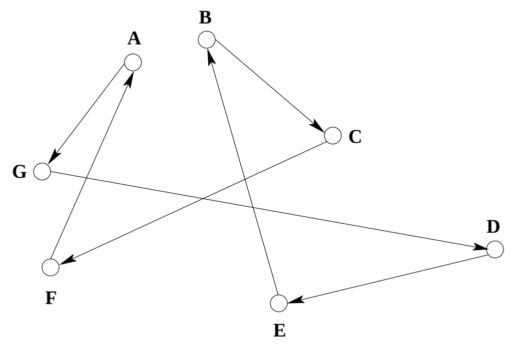


Figure 2.1: *An example traveling-salesman problem, with current route* **A-G-D-E-B-C-F-A**. *A solution feature is the length of the route. A problem instance feature is the location of the cities.*

problem (Figure 2.1). The set of solution features for a traveling-salesman problem contains such information as the length of the computed tour and the order in which the cities are visited. The set of problem instance features contains such information as the location of each city, and any constraints on the order in which they must be visited.

Given a feature tuple, an agent is able to extract information to determine the *quality* of having a solution with that feature vector.

Definition 2 (Quality Function). *A* quality function*, q_i^j for agent i and problem j is a mapping from the space of feature tuples $\mathcal{F}_i^j(t)$ to the real numbers. That is*

$$q_i^j : \mathcal{F}_i^j(t) \mapsto \mathbb{R} \quad \forall t.$$

We will often refer to the quality of the solution as the *value* of the solution and will sometime use the notation $v_i^j(f(t))$. Additionally, we will often use the notation $q_i^j(t)$ when it is clear from the context which feature tuple is meant.

In our model an agent uses its resources to change (its knowledge about) solution quality. We differentiate between an agent *refining* its information about the solution solution and *improving* the quality of its solution.

In the refining model an agent's computing does not change the solution. Instead, it is assumed that the problem or instance has some real $\hat{s}$ which is initially unknown to the agent. As the agent devotes resources to the problem, its knowledge about the solution changes, while the real solution $\hat{s}$ remains unchanged.

Definition 3 (Refining). *Let $\mathcal{F}_j$ be the set of all possible feature tuples for problem j. Let Pr_j be the probability distribution over $\mathcal{F}_j$. Assume that*

- *problem j has some true, fixed, unchangeable solution $\hat{s}_j$,*
- *agent i initially knows probability distribution Pr_j,*
- *as agent i allocates t resources to problem j it collects information* $\text{info}(t)$*, where* $\text{info}(t-1) \subseteq \text{info}(t)$*, and*
- *let $Pr_j^{\text{info}(t)}$ denote the probability distribution over $\mathcal{F}_j$ defined such that*
$$Pr_j^{\text{info}(t)}(\hat{s}_j = s_j) = Pr_j^{\text{info}(t-1)}(\hat{s}_j = s_j|\text{info}(t))$$
for $s_j \in \mathcal{F}_j$, where $Pr_j^{\text{info}(0)} = Pr_j$.

Computing refines *for agent i if there exists some t such that*

1. $\text{info}(t-1) \subset \text{info}(t)$*, and*
2. $Pr_j^{\text{info}(t-1)} \neq Pr_j^{\text{info}(t)}$.

We draw the readers attention to the fact that it is possible that an agent may gather information in a time step which does not change its knowledge about the true solution of the problem. For refining to occur, we merely insist that at some time step an agent collects information which causes it to update its probability distribution over $\mathcal{F}_j$.

Refining a solution is a general process so we provide an example in order to make it more concrete. Assume that an agent is only interested in learning how much it must pay in order to get a certain quality of service. Assume that the agent knows that the true price, $\hat{p}$, is an integer drawn uniformly from the interval $[p_1, p_2]$. That is, the probability that price of the service is p is $\frac{1}{p_2-p_1+1}$. By allocating resources to the problem, the agent may be able to change the bounds on the interval. For example, it may collect information indicating that the real price actually lies in the interval $[p_1, p_2']$ where $p_2' \leq p_2$. The agent would then update its probability distribution so that it is consistent with the new information, thus deducing that the probability that the real price is p is $\frac{1}{p_2'-p_1+1}$ if $p \in [p_1, p_2']$ and 0 otherwise. It should be noted that the agent did not change the price of the service by computing.

In the improving model, the agent actively changes the solution quality by allocating resources to the problem. In particular, the solution quality improves.

Definition 4 (Improving). *Computing* improves *the solution quality for agent i on problem j, if after computing for t time steps the solution vector was $f_i^j(t)$ and for any $\Delta t > 0$,*

$$q_i^j(f_i^j(t + \Delta t)) \geq q_i^j(f_i^j(t)).$$

When there are no restrictions on computing resources, an agent will always wish to allocate more resources to a problem since the solution quality is guaranteed to improve.

To summarize, there is a subtle difference between computing to improve solution quality and computing to refine a solution. If computing is used to improve solution quality then the solution actually changes. If computing is used to refine the solution, then the actual solution does not change, instead the agent's knowledge about the solution changes.

2.2.1 Examples

In this section we present two examples where an agent uses computing resources in order to change the quality of its solution. The first example involves a scenario where computing improves solution quality, while the second example illustrates refining of solution quality.

Consider the network problem in Figure 2.2. The goal of the agent in charge of the network is to find a route from the source to the destination such that the latency is minimized. An agent can start with some initial path, and then search to find paths with lower latency. As the agent computes (searches) on the problem, it has the potential of finding better routes, thus improving the solution.

In some applications agents can *gather* or *refine* information. Consider an agent who is responsible for organizing a holiday for a user. Initially, the agent has an estimate as to what airfare, food and lodging will cost. As the agent gathers information it improves its estimate as to what the total cost may be, as well as what days of the week the trip can be taken. The information refining done by the agent does not change the actual lowest cost of the airfare and hotel. However, the information gathering actions do change the information the agent has available to it about the cost of the trip.

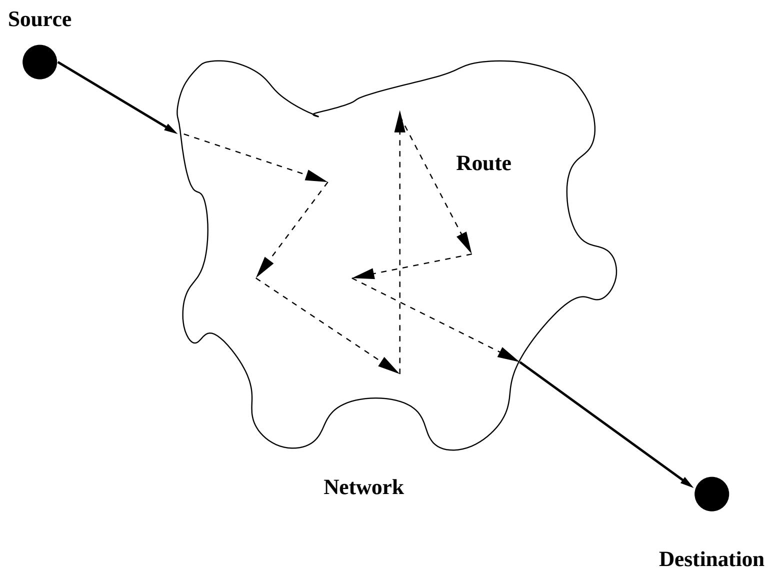


Figure 2.2: *An agent is responsible for routing packets from a source to a destination. Its goal is to minimize latency.*

2.3 Computationally Limited Agents

In the previous section (Section 2.2) we described the roles that computation can play in different agent settings. In this section, we define what we mean by a computationally limited agent, and describe the tools that agents can use to effectively use their computational resources.

A computationally limited agent i is defined by

$$\langle T_i, \mathrm{cost}_i(), \mathcal{A}_i, \mathcal{PP}_i \rangle$$

where

- T_i denotes the agent's computing resources,
- $\mathrm{cost}_i()$ is a cost function which represents the agent's computational limitations,
- $\mathcal{A}_i$ is the set of algorithms used by the agent, and
- $\mathcal{PP}_i$ is the set of performance profiles, that is, tools used by the agent in order to make effective decisions on how to use its computing resources.

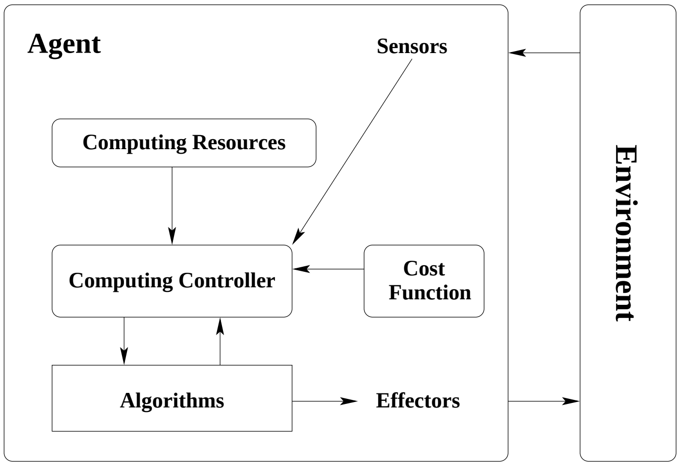


Figure 2.3: *A computationally limited agent.*

Figure 2.3 illustrates a computationally limited agent. In the rest of this section we describe each aspect of a computationally limited agent.

2.3.1 Computing Resources and Cost Functions

We represent the computing resources of an agent i by T_i. As mentioned in Section 2.2, an agent uses these resources to change its knowledge about the solution quality of different problems. This may involve physically changing the solution of a specific problem (improving solution quality) or it may involve learning more about some solution without changing it (refining solution quality).

If the agent had unlimited computing resources then it would be able to fully discover the optimal solution to any problem. However, this is rarely the case. Limitations on the resources restrict the choices available to agents, who must carefully decide how to best use the resources available to them.

We assume that it is possible to allocate computing resources across m different problems, and let $\bar{t}_i = (t_1, \dots, t_m) \in T_i^m$ denote the situation where the agent has allocated t_j computing steps to problem j, for $1 \leq j \leq m$. To model restrictions on an agent's computing power, we use cost functions.

Definition 5 (Costly Computing). *An agent i has* costly computing *if there exists a cost function,* cost_i*, such that*

$$\text{cost}_i : T_i^m \mapsto \mathbb{R}^+ \cup \{0\}.$$

A fully rational agent, one who in theory has no computational restrictions, has a cost function of the form $\text{cost}(\bar{t}) = 0\ \forall \bar{t}$. In general we place no restrictions on the cost functions of an agent. However, there is one class of functions which are of particular interest. These *limited computing functions* allow an agent to compute for free up to a certain limit, after which all computing must stop.

Definition 6 (Limited Computing). *An agent i has* limited computing *if it has a cost function* cost_i*, and a limit D such that for all $(t_1, \ldots, t_m) \in T_i^m$*

$$\text{cost}_i((t_1, \ldots, t_m)) = \begin{cases} 0 & \textit{if } \sum_{j=0}^{m} t_j \leq D \\ \infty & \textit{otherwise} \end{cases}$$

Situations where agents have hard deadlines naturally fit into the limited computing model. The more general costly computing model can naturally cover situations where there is some cost associated with each step of computing. For example, an agent may have to pay for CPU cycles, and can incur a fixed cost, K, for each cycle used. The cost function for such a scenario would be

$$\text{cost}_i((t_1, \ldots, t_m)) = K \sum_{j=1}^{m} t_j.$$

As a side note, when there is only one problem upon which an agent can compute, we abuse notation and use $\text{cost}(t)$ to denote the situation where an agent has allocated t resources on the one problem.

2.3.2 Anytime Algorithms

We assume that agents have algorithms which allow them to make an explicit tradeoff between the agents' computing resources and the quality of the solution returned by the algorithm. In particular, we assume that the agents' algorithms are *anytime*. There are two defining properties of an anytime algorithm:

1. An anytime algorithm can be stopped at any point and will return a solution, and

2. The more computing resources allocated to an anytime algorithm, the better the returned solution quality.

While anytime algorithms naturally capture the improving model of computation, they also model the refining scenario in that they can be viewed as returning information to the agent at any point in time.

All anytime algorithms fall into one of two categories; interruptible algorithms or contract algorithms.

Definition 7 (Interruptible Algorithm). *[124] Let $\mathcal{A}$ be an anytime algorithm and let $q_{\mathcal{A}}(t)$ be the quality of the solution returned by the algorithm at time t. Algorithm $\mathcal{A}$ is an* interruptible algorithm *if for each problem instance, for all time t, t', if $t' \geq t$ then*

$$q_{\mathcal{A}}(t') \geq q_{\mathcal{A}}(t).$$

Clearly, interruptible algorithms have the anytime property since by definition they can be stopped at any time step and will return a usable solution, and the quality of the solution improves if additional time is allocated. What makes interruptible algorithms particularly desirable is that the running time need not be specified in advance and can still return valid solutions when interrupted unexpectedly. Examples of interruptible algorithms include most iterative refinement algorithms such as simulated annealing and heuristic repair [75].

Contract algorithms also have the anytime property but do not have the same flexibility of interruptible algorithms.

Definition 8 (Contract algorithm). *[124] Let $\mathcal{A}$ be an anytime algorithm and let $q_{\mathcal{A}}(t)$ be the quality of the solution returned by the algorithm at time t. Algorithm $\mathcal{A}$ is a* contract algorithm *if*

1. *For any time t, t', $t' \geq t$ and t, t' were announced prior to running algorithm $\mathcal{A}$,*

   $$q_{\mathcal{A}}(t') \geq q_{\mathcal{A}}(t)$$

   and,

2. *For any time $t^* \neq t, t'$, if algorithm $\mathcal{A}$ is stopped at time t^* there is no guarantee as to the quality of $q_{\mathcal{A}}(t^*)$.*

While contract algorithms have the anytime property, the time allocation must be done *before* the algorithm begins. Given a time allocation t, the algorithm will return the best solution possible. However, if the algorithm is stopped before t then there is no guarantee that even a usable solution will be returned. Tree search algorithms can often be modeled as contract algorithms. For example, RTA* uses a predetermined search horizon that is determined from a given time allocation [54]. While the algorithm will return a result for any time allocation, if it is interrupted early then it is not guaranteed to have found a solution. Trivially, interruptible algorithms can be converted into contract algorithms. It is also possible to convert a contract algorithm into an interruptible algorithm [124].

2.3.3 Performance Profile Based Deliberation Control

We assume that agents are deliberative with respect to how they use their computing resources in order to get different solutions. The goal of an agent is to choose how to allocate its computing resources so as to maximize its utility. We define utility of agent i after allocating t resources to algorithm $\mathcal{A}$ to be

$$U(q_{\mathcal{A}}(t), t) = q_{\mathcal{A}}(t) - \mathrm{cost}(t)$$

where $q_{\mathcal{A}}$ is the solution quality returned by algorithm $\mathcal{A}$ after t resources have been allocated, and $\mathrm{cost}(t)$ is the associated cost. In particular, an agent wants to choose the allocation t^* such that

$$t^* = \arg\max_t [q_{\mathcal{A}}(t) - \mathrm{cost}(t)].$$

This problem is illustrated in Figure 2.4. While solution quality increases, at some point the cost associated with improving the solution quality becomes too high compared to the possible gain in solution quality, so that further computation has little value. The optimal computing allocation depends on several factors: the quality of the solution, the prospect of further improvement in solution quality, the current state of the environment, and the cost of computing. While anytime algorithms are models that allow for the trading off of computational resources for solution quality, they do not provide a complete solution for an agent. Instead, anytime algorithms are paired with a meta-level control procedure which helps in determining how long to run an algorithm, and when to stop and act with the solution obtained. There are two components to the meta-level control procedure:

1. The *performance profile* which describes how computation changes the solution quality, and

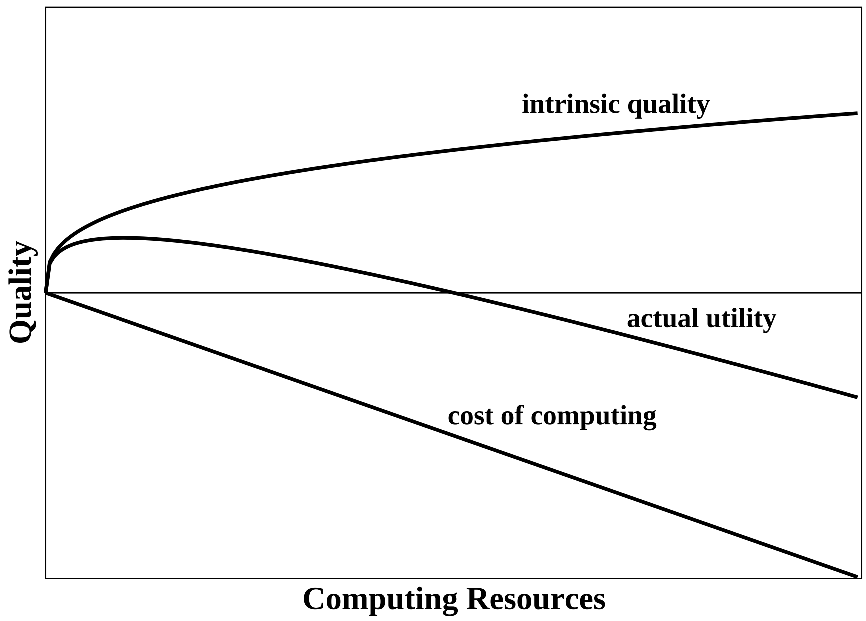


Figure 2.4: *A typical view of anytime algorithms [32, 43, 45, 104].*

2. The *deliberation control policy* which determines how to use the information from the performance profile to make decisions about how to allocate computing resources.

In the rest of this thesis we will use the term performance profile to refer to both the descriptive and prescriptive parts of the meta-level control procedure. While the performance profiles for some algorithms can be determined by analysis of the algorithm itself (for example, Newton's method where the error in the result is bounded by the number of iterations can be considered to be a performance profile as it provides information about the solution for allocations of computing resources), for most algorithms structural analysis is too difficult, if not impossible, for practical purposes. Instead, performance profiles are generally created by collecting statistics from previous runs of the algorithm on different problem instances or simulations.

There are different ways of representing performance profiles. At a high level, performance profiles can be classified as either *static* or *dynamic*. Static performance profiles only allow the agent to make computing decisions before running the anytime algorithm. Dynamic performance profiles allow the agent to make online decisions, using information from the progress of the algorithm. In the rest of this section we describe commonly used static and dynamic performance profiles. In the next sec-

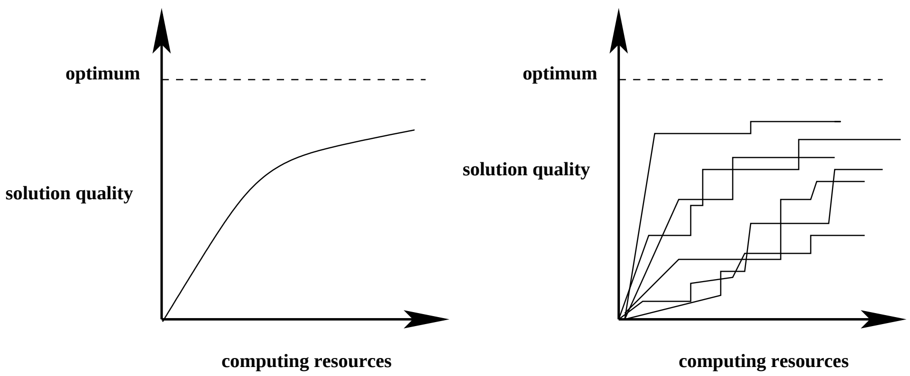


Figure 2.5: *Performance profiles. The performance profile on the left is an idealized performance profile. The performance profile on the right illustrates a more realistic performance profile for an anytime algorithm.*

tion we introduce the performance profile tree, a fully normative deliberation control procedure.

Static Performance Profiles

At a high level, a static performance profile of an algorithm $\mathcal{A}$ is simply a function which maps computing resources to solution quality:

$$PP_{\mathcal{A}} : T \mapsto \mathcal{Q}$$

where $\mathcal{Q}$ is the space of solution quality.

In an ideal world, the performance profile of an anytime algorithm can be represented as a single curve, showing the solution quality as a function of allocated computing resources (Figure 2.5 left). However, in reality, there is often variance in each run of the algorithm, caused by differences in different inputs as well as randomization in the algorithm itself (Figure 2.5 right). Instead of using the idealized performance profile, an *expected performance profile* is often used. Statistics are collected from previous runs of the algorithm on different input. This produces a probability distribution over solution quality for each allocation of computing resources, which allows the agent to determine the expected solution quality for every allocation t.

Definition 9 (Expected performance profile). *[124] An* expected performance profile*, $E_{\mathcal{A}}$, for algorithm $\mathcal{A}$ is a function from computing to the expected quality of* *solution. That is*

$$E_{\mathcal{A}} : T \mapsto \mathcal{Q}$$

where

$$E_{\mathcal{A}}(t) = \sum_{q(t)} q(t) Pr_{\mathcal{A},t}(q(t))$$

where $q(t)$ is solution quality at t and $Pr_{\mathcal{A},t}(q)$ is the probability that algorithm $\mathcal{A}$ produces solution quality q after t computing steps.

Although it is sometimes possible to represent a performance profile by a parameterized function [12, 48], performance profiles are commonly represented as tables of discrete values, where computing resources are discretized into a finite number of time steps, $t_0, \ldots, t_n$ where t_0 is the start time of the algorithm and t_n is the maximum running time of the algorithm. Solution quality is also discretized into a finite number of levels $q_0, \ldots, q_m$, where q_0 is the lowest quality level and q_m is the highest quality level. Figure 2.6 is an example of such a table. Each table entry (q_i, t_j) contains the probability of getting solution quality q_i given that t_j computing resources was allocated to the algorithm. These probabilities are derived from statistics collected from earlier runs of the algorithm on different problem instances.

An agent uses a static performance profile to determine how to best allocate its computing resources. In particular, the agent wishes to solve the optimization problem

$$\max_t [q_{\mathcal{A}}(t) - \text{cost}(t)].$$

Since the agent must work in expectation, it solves

$$\max_t [\sum_{q(t)} q(t) Pr_{\mathcal{A},t}(q(t)) - \text{cost}(t)].$$

This decision is made before the algorithm is started, as static performance profiles do not have enough flexibility to allow for online deliberation control decisions. They are ideally suited for contract algorithms.

Dynamic Performance Profiles

While static performance profiles help agents decide how to allocate their computing resources before starting an anytime algorithm, better stopping policies may be

solution quality		t0	t1	t2	t3	t4	t5
	q7	0.0	0.0	0.0	0.0	0.0	0.3
	q6	0.0	0.0	0.0	0.1	0.2	0.5
	q5	0.0	0.0	0.1	0.25	0.3	0.2
	q4	0.0	0.0	0.2	0.4	0.4	0.0
	q3	0.0	0.1	0.2	0.2	0.09	0.0
	q2	0.01	0.3	0.3	0.05	0.01	0.0
	q1	0.98	0.5	0.2	0.0	0.0	0.0
	q0	0.01	0.1	0.0	0.0	0.0	0.0

computing resources

Figure 2.6: *An example performance profile. The table lists discrete probability distributions over solution quality levels, for each discrete time step. An agent uses the table to compute the expected solution quality for each time allocation.*

achievable by using information about the current run of the algorithm on the current problem instance. This leads to the introduction of *dynamic performance profiles.*

Definition 10 (Dynamic performance profile). *Let $\mathcal{Q}(t)$ be the set of all qualities possible at time t, i.e.$\mathcal{Q}(t) = \{q(t)\}$. A* dynamic performance profile *of anytime algorithm $\mathcal{A}$ is*

$$PP_{\mathcal{A}} : \mathcal{Q}(t) \times \{\Delta t\} \mapsto \{Pr(q(t+\Delta t)|q(t), \Delta t) : \; q(t+\Delta t) \in \mathcal{Q}(t+\Delta t)\}$$

where $Pr(q(t+\Delta t)|q(t), \Delta t)$ is the probability of having solution quality $q(t+\Delta t)$ after allocating $t+\Delta t$ computing resources, given that at resource usage t the solution quality was $q(t)$.

That is, given a current solution quality at t and some predefined amount of computing resources Δt, a dynamic performance profile produces a probability distribution over solution quality at $t + \Delta t$. The power of the dynamic performance profile is that it allows an agent to base its deliberation decision on current information about the algorithm's performance. As the agent monitors the progress of the algorithm, it can use this information to make better informed decisions. Figure 2.7 represents a dynamic performance profile. The table on the left shows the agents probability distributions over solution quality at different time steps. As always, this probability distribution is based on historic data from previous runs of the algorithm.

solution quality	t0	t1	t2	t3	t4	t5
q7	0.0	0.0	0.0	0.0	0.0	0.3
q6	0.0	0.0	0.0	0.1	0.2	0.5
q5	0.0	0.0	0.1	0.25	0.3	0.2
q4	0.0	0.0	0.2	0.4	0.4	0.0
q3	0.0	0.1	0.2	0.2	0.09	0.0
q2	0.01	0.3	0.3	0.05	0.01	0.0
q1	0.98	0.5	0.2	0.0	0.0	0.0
q0	0.01	0.1	0.0	0.0	0.0	0.0

computing time

solution quality	t0	t1	t2	t3	t4	t5
q7	0.0	0.0	0.0	0.0	0.0	0.3
q6	0.0	0.0	0.0	0.13	0.22	0.5
q5	0.0	0.0	0.1	0.33	0.33	0.2
q4	0.0	0.0	0.2	0.54	0.45	0.0
q3	0.0	0.1	0.2	0.0	0.0	0.0
q2	0.01	0.3	0.3	0.0	0.0	0.0
q1	0.98	0.5	0.2	0.0	0.0	0.0
q0	0.01	0.1	0.0	0.0	0.0	0.0

computing time

Figure 2.7: *A dynamic, table-based performance profile. By monitoring the solution quality of the algorithm as it executes can improve the deliberation control policy of an agent.*

Assume that at time t_2 the anytime algorithm that the agent was using had solution quality q_4, as indicated by the circled entry in the table. Since the agent is using an anytime algorithm, it knows that if more time is allocated to the algorithm, the solution quality *will not decrease*. Therefore, as shown in the table to the right in Figure 2.7, the agent can update (using Bayes Rule) the probability distributions over solution quality at future time steps given that it has solution quality q_4 at time t_2, and then make a new deliberation decision about whether to allocate more computing resources or to stop and act with the solution quality at hand. There are a collection of techniques which agents use to develop deliberation policies coupled with dynamic performance profiles. These range from a simple myopic approach where, after updating the probability distributions over future solution quality given current solution quality, an agent makes a decision based on one-step of lookahead, to more complex techniques involving dynamic programming [43]. To illustrate how these performance profiles work, we describe dynamic programming deliberation control [43]. The goal is to determine a deliberation policy $\pi(q_i(t_k), t_k)$ which specifies what deliberation action an agent should take for every resource usage t_k and solution quality $q_i(t_k)$). The deliberation actions available to the agent are {stop, continue for one step}. We denote a deliberation action by $d \in$ {stop, continue for one step}. A deliberation policy is found by optimizing the following value function

$$V(q_i(t), t) = \max_d \begin{cases} q_i(t) - \text{cost}(t) & \text{if } d = \text{stop} \\ \sum_j Pr(q_j(t+\Delta t)|q_i(t), \Delta t) V(q_j(t+\Delta t), t+\Delta t) & \text{if } d = \text{continue} \end{cases}$$

quality	t0	t1	t2	t3	t4	t5
q7	0.0	0.0	0.0	0.0	0.0	0.3
q6	0.0	0.0	0.0	0.1	0.2	0.5
q5	0.0	0.0	0.1	0.25	0.3	0.2
q4	0.0	0.0	0.2	0.4	0.4	0.0
q3	0.0	0.1	0.2	0.2	0.09	0.0
q2	0.01	0.3	0.3	0.05	0.01	0.0
q1	0.98	0.5	0.2	0.0	0.0	0.0
q0	0.01	0.1	0.0	0.0	0.0	0.0

time

Figure 2.8: *A table-based dynamic performance profile. While it is possible to condition deliberation decisions on solution quality, information about the path is lost. For example, to reach the shaded square, an algorithm could have followed path A or path B. If it followed path B then it appears more likely that the solution quality would continue to improve at a faster rate than it path A had been followed. This is not captured in the table-based representation.*

to determine the policy

$$\pi(q_i(t), t) = \arg\max_d \begin{cases} q_i(t) - \text{cost}(t) & \text{if } d = \text{stop} \\ \sum_j Pr(q_j(t+\Delta t)|q_i(t), \Delta t) V(q_j(t+\Delta t), t+\Delta t) & \text{if } d = \text{continue} \end{cases}$$

This dynamic programming approach, coupled with appropriately discretized table-based performance profiles, leads to optimal deliberation policies, assuming that the solution quality completely satisfies the Markov property (Theorem 2 in [43]). In particular, as long as the probability distribution of future quality depends only on the current "state" of the anytime algorithm then it is optimal, where in the table-based performance profile, the "state" of the anytime algorithm is assumed to be current solution quality. However, there are circumstances when this assumption does not hold due to limitations with the table-based representation. For example, the table-based representation can not capture information such as the path of the algorithm (Figure 2.8), and thus this information can not be used by the agent when creating its deliberation control policies.

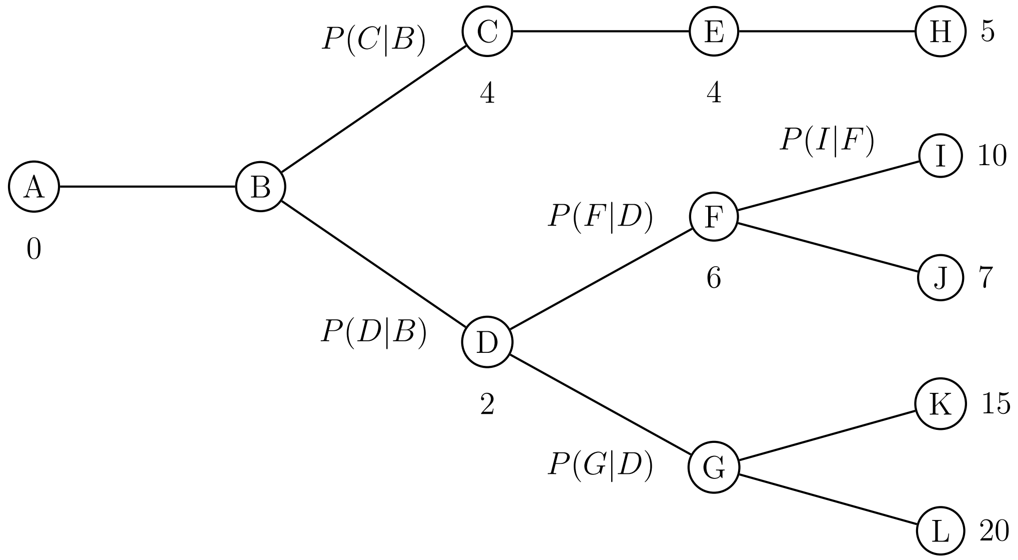


Figure 2.9: *An agent's performance profile tree. The numbers represent the solution quality associated with each node. The edges from any node has a weight equal to the probability of reaching the child, given that the parent was reached. Each edge corresponds to the act of taking one computing step. For example, it take 3 computing steps to reach node F from node A.*

2.4 Performance Profile Trees

While the performance profile deliberation control methods described in this chapter so far have been to shown to work well in practice, none of them are fully normative. Instead, they rely on simplifying assumptions such as assuming that an algorithm's performance can be deterministically predicted, or assuming that the path of the algorithm does not depend on the run of the instance so far. This lack of full normativity arises due to the performance profile representations. In this section, we propose using a tree structure which allows an agent to condition its deliberation decisions on all information available (such as solution quality, problem instances, path of the algorithm). We call this performance profile a performance profile tree (Figure 2.9)

We start by describing a performance profile tree created using only solution quality information. In following sections we expand the definition of the performance profile tree to include conditioning on additional information stored in feature vectors, as well as describing how performance profile trees can handle uncertainty arising from different sources.

Each node in the tree (solution node) stores solution quality obtained for a certain time allocation. We use the notation $V(n)$ to denote the value or solution quality of node n. Each edge in the tree has a weight. The weight of edge e connecting nodes n and n' is the probability of getting the solution quality stored in node n' after one more computing step, given that the current solution quality is stored in node n. That is, the weight of edge e, is defined to be $weight(e) = P(n'|n)$. Each edge in the tree represents the act of taking one computing step. This is useful in determining how much computational effort is required to obtain a certain solution quality. For example, to obtain solution quality stored at node n, an agent must compute for the number of steps equal to the number of edges in the path to reach node n. We denote this by time(n). It should be noted that the information stored at node n can occur multiple times in the tree. However, each occurrence is represented by a different node, which represents a unique path. Because of this, an agent may be able to expend different amounts of computational effort to get the same solution quality.

The tree is constructed by collecting statistical data from previous runs of the algorithm on different problem instances. The more finely solution quality and time are discretized, the more accurate deliberation control is possible. However, with more refined discretization, the number of possible runs increases (it is $O(m^D)$ where m is the number of levels of solution quality and D is depth of the tree), so more runs need to be seen to populate the space. A tighter bound can be obtained once the observation is made that the values of the solutions are always increasing and can be represented as step functions. The bound is $O(Nd)$ where N is the number of leaves in the tree and d is the average depth [6].[1] Furthermore, the space should be populated densely to get good probability estimates on the edges of the performance profile trees. Each run is represented as a path in the tree. As a run proceeds along a path in the tree, the frequency of each edge of that path is incremented, and the frequencies at the nodes on the path are normalized to obtain probabilities. If the run leads to a value for which there is no node in the tree, the node is generated and an edge is inserted from the previous node to it.

One advantage of the performance profile tree is that it automatically supports conditioning deliberation decisions on the path of the algorithm. Each node in tree corresponds to a *unique path* that the algorithm may have followed. The performance

[1]The number N can be quite large, however, it is always bounded by the number of instances used to populate the performance profile tree since each algorithm run can insert at most one path into the tree.

profile tree that applies given a path of computing is the subtree rooted at the current node n. This subtree is denoted by $\mathcal{T}_i(n)$. If an agent is at a node n with quality $q(n)$, then when estimating how much additional deliberation 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 $\mathcal{T}_i(n)$ is simply the product of the probabilities on the path from n to n'. The expected solution after allocating t more time steps to the problem, if the current node is n, is

$$\sum P(n'|n) \cdot q(n')$$

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

Using a performance profile tree, an agent can determine an online deliberation policy which specifies whether an agent should take an additional computing step, given the current solution quality. We denote the deliberation action of agent as $d \in \{$stop, continue for one step$\}$. The deliberation policy for an agent using a performance profile tree is constructed by optimizing the following rule:

$$V(n,t) = \max_d \begin{cases} q(n) - \text{cost}(t) & \text{if } d = \text{stop} \\ \sum_{n'} P(n'|n)V(n', t+1) & \text{if } d = \text{continue} \end{cases}$$

where $n' \in \{$nodes in $\mathcal{T}(n)$ reachable in 1 computing step$\}$, to determine the deliberation policy

$$\pi(n,t) = \arg\max_d \begin{cases} q(n) - \text{cost}(t) & \text{if } d = \text{stop} \\ \sum_{n'} P(n'|n)V(n', t+1) & \text{if } d = \text{continue} \end{cases}$$

This policy is computed offline, but produces a policy that is used online. The agent monitors the solution quality as it deliberates, and then uses the deliberation policy to determine what action should be taken in the next step.

2.4.1 Conditioning Deliberation on Additional Features

The performance profile tree additionally allows an agent to condition its deliberation decisions on any and all features of the solution and problem that the agent deems to be important. In particular, the nodes in the tree can store the quality of the solution along with the feature tuple which describes the solution and problem instance in

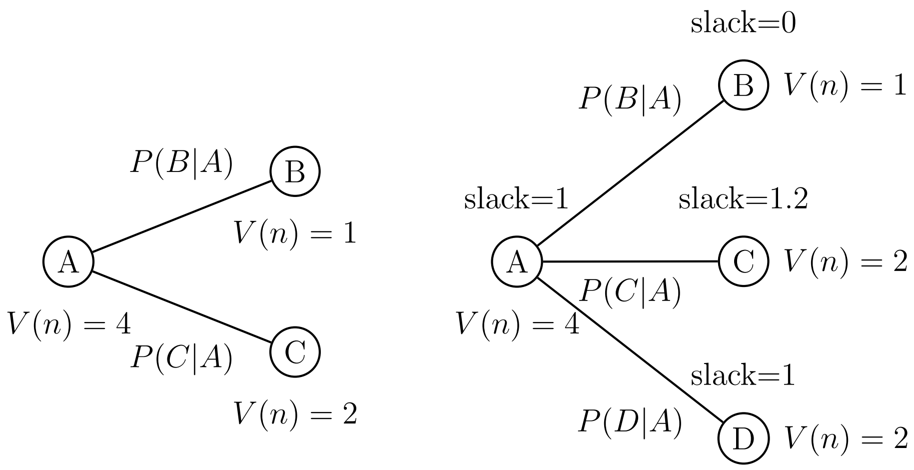


Figure 2.10: *Two example performance profile trees for a scheduling application. The performance profile tree on the left was created using only data on the length of the schedule (the optimization problem is to find the shortest schedule, thus lower numbers signal better solution quality). The performance profile tree on the right was created using solution quality data along with information on the slackness in the schedule. This can lead to larger performance profile trees. For example, if the agent initially has a solution associated with node A, then in one computing step an agent may obtain a schedule of length 2. However, there are two solutions of length 2 which have different slackness in them. Thus, they are represented by two distinct nodes in the tree (C and D).*

more detail. For example, in a scheduling domain both the length of the current schedule (quality) and the slackness in the schedule can serve as indicators as to whether further computing on the problem will lead to improvements. These two sources of information can both be stored in a node. Each node n with quality $V(n)$ and slackness $\mathrm{sl}(n)$ uniquely defines a path the algorithm followed on a problem instance. The agent can use this information in formulating its deliberation policies as it need only consider the subtree rooted at the current node, n, when estimating how further computing will change the solution quality and slackness.

While including both solution quality and feature tuples in the solution nodes of the performance profile trees provides an agent with more information that it can use when making deliberation decisions, it does come at a cost to the size of the performance profile trees. Including additional information only increases the number of nodes in the tree as illustrated in Figure 2.10.

2.4.2 Modeling Sources of Uncertainty

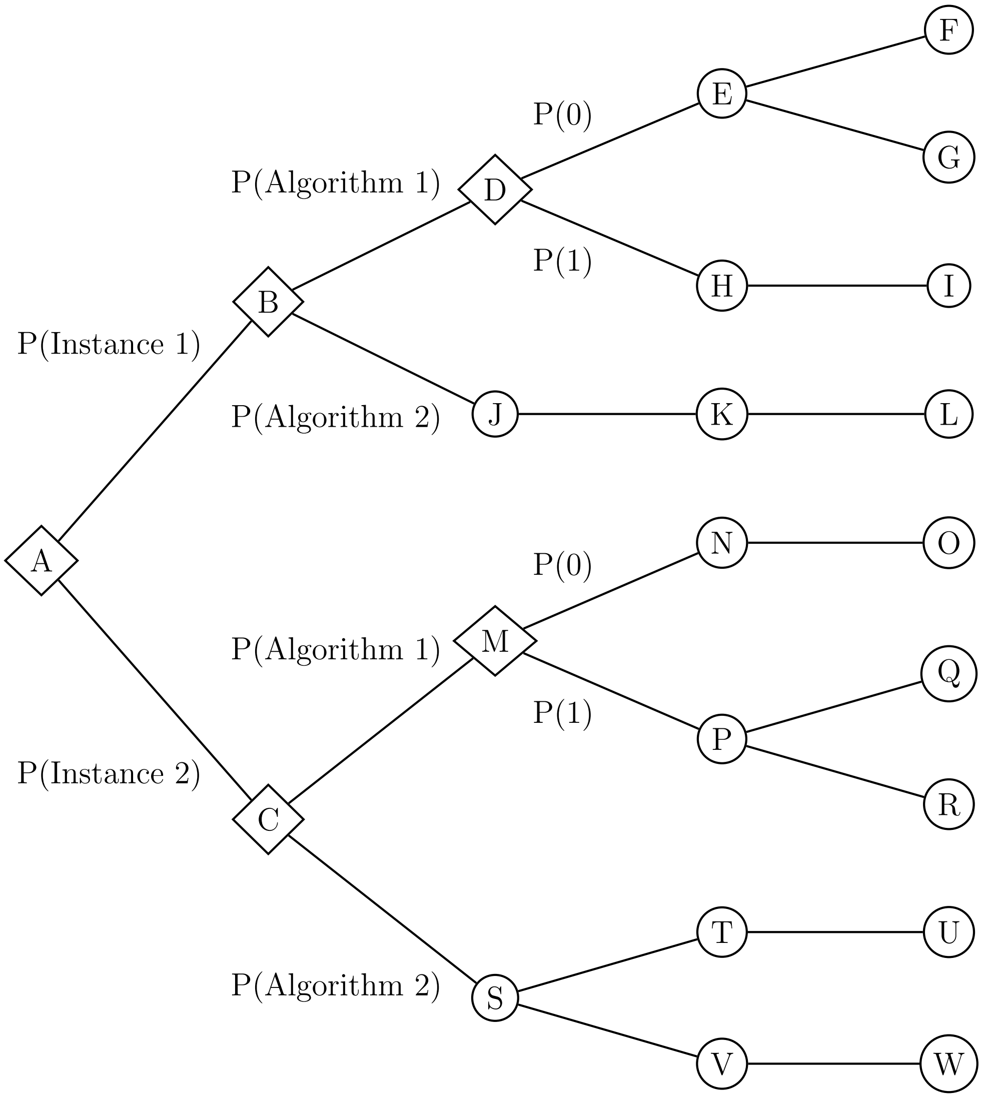


Figure 2.11: *An augmented performance profile. This performance profile captures the situation where there is uncertainty in which problem instance is being computed on (random node A), in which algorithm is being used (random nodes B and C), and when algorithm 1 is used, whether the random number drawn is 1 or 0 (random nodes D and M). To reach node F the agent must compute 1 time step, while to reach node L the agent must compute for 2 time steps.*

The performance profile tree discussed so far in this dissertation is ideally suited for single-agent settings. It captures uncertainty that an agent running a single algorithm on a single problem might encounter, such as randomness in both the algorithm run and the problem instance, and allows the agent to make optimal deliberation decisions using this information. However, in multiagent settings, as will be introduced in Chapter 5, agents may need explicit models of where uncertainty arises in the problems that other agents are computing on, since the deliberation decisions of one agent can determine the best deliberation decisions of another agent. A problem that an agent can face is that it might not know how the uncertainty in the computing problem of a competitor agent resolved. Instead, if uncertainty is explicitly captured in the performance profile, the agent can emulate possible runs of a competitor's algorithm, and thus make better deliberation control decisions. While we delay the description of the details of how an agent might use performance profile information of another agent until Chapter 5, we now describe how uncertainty information can be explicitly modeled in the performance profile tree.

We can explicitly capture uncertainty in our deliberation control framework by using an *augmented performance profile tree*. An example of an augmented performance profile tree is presented in Figure 2.11. An augmented performance profile tree is simply a performance profile tree together with *random nodes* which represent the occurrence of randomness which make it impossible for an agent to predict which particular path its computation will follow, even if the problem instance is known completely. We use random nodes to explicitly model uncertainty for multiagent settings in the following situations:

1. when randomized algorithms are used,
2. when there there is uncertainty as to which algorithm is being used, and
3. when there is uncertainty about the actual problem instance being computed on.

Random nodes are inserted into the performance profile whenever there is only probabilistic information about some event which is unrelated to the actual problem instance, and which are outside of the agent's control. The edges emanating from a random node each represent the occurrence of a specific event, and are labeled with the probability that the event occurred. The weight of each edge is zero since we assume that the randomization or lack of knowledge does not correspond to a computing or deliberating action taken by the agent.

If a randomized algorithm is being used, then there is uncertainty as to what the path of the algorithm on any given problem instance will be, since it is not known in advance what random numbers will be provided by the random number generator. Random nodes can be used whenever randomization occurs, and the edges emanating from a random node each correspond with a particular occurrence of a random number. Each edge is labeled with the probability that its corresponding number was generated by the random number generator. An agent is unable to predict in advance which number will be drawn, but it can *emulate* the occurrence of different draws in order to help in deciding whether to continue deliberating on a problem or not. That is, whenever it encounters a random node in the tree when it is determining its deliberation policy, the agent can actively choose a "random" number and thus learn what sort of solution quality it could obtain if the random number chosen occurred in practice. By doing so, the agent can determine optimal deliberation control policies for each possible random number. As mentioned earlier, this can be a useful strategic tool when there are multiple strategic agents. We refer the reader to Chapter 5 for a detailed description.

Random nodes are also used whenever an agent needs to collect information about the computing results of another agent, but is unsure which algorithm is being used, or which problem instance is being computed on. Each edge emanating from a random node corresponds to a specific algorithm (or problem instance) and are labeled with the probability that the algorithm (problem instance) is the one of interest. An agent can use this representation exactly as in the random number setting, by first assuming a particular algorithm or problem instance and determining the deliberation policy for that instance, and then afterwords, combining all deliberation policies into a larger one, conditional on whether the uncertainty is eventually resolved.

2.4.3 Stochastic and Deterministic Performance Profile Trees

Performance profiles trees can have different structures, depending on the data used to generate them. At times the structure of the tree can be used in order to simplify the agents' deliberation control problems. We make a distinction between *stochastic* performance profiles and *deterministic* performance profiles. Stochastic performance profiles are trees where there is at least one node with a branching factor greater than one. This means that before an agent does any computing there is some uncertainty as to the path of the algorithm. This can make determining the optimal deliberation

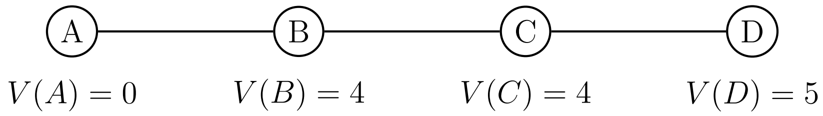


Figure 2.12: *A deterministic performance profile tree.*

control policy difficult as the uncertainty must be taken into account. Stochastic performance profiles can occur both when an agent is computing to improve its solution quality or refine its solution quality due to either randomization in the algorithm itself, lack of knowledge about some aspect of the problem instance, *etc.* Performance profile trees in Figures 2.9, 2.10 and 2.11 are all stochastic.

Deterministic performance profile trees are branches (Figure 2.12). Each node has at most one child and there is no uncertainty as to what results computing will bring. This makes determining the optimal time allocation for a problem trivial as an agent need only solve the optimization problem

$$\arg\max_t [V(n(t)) - \text{cost}(t)]$$

where $n(t)$ is the (unique) node reachable by computing for t time steps. If agents compute to *refine* its solution then an agent with a deterministic performance profile has no incentive to compute at all on the problem as there is no uncertainty to resolve. However, if an agent computes to *improve* solution quality, then it still has incentive to compute if its performance profile is deterministic since computing changes the solution. Thus, even though an agent with a deterministic performance profile may know with certainty that in t time steps it will have a solution with quality $V(n(t))$, unless the agent actively computes it will not obtain this solution. For example, in a traveling-salesman domain, there is a difference between knowing that it is possible to find a route of length k, and actually knowing the route.

2.4.4 Full Normativity of the Performance Profile Tree

The performance profile tree is a fully normative deliberation control method. It can, in theory, capture all possible features that an agent may use to make deliberation control decisions. For example, the feature tuple stored in each solution node contains information about the problem instance and solution features which the agent can monitor and base deliberation decisions. Path information is maintained automatically due to the tree structure, and thus can also be used by agents when creating deliberation policies. Finally, the performance profile tree is also flexible enough to capture sources of uncertainty arising from randomization in the algorithms, as well as uncertainty from other sources.

In many single-agent applications, the full normativity of the performance profile might not be required, as reasonable performance might be achievable even when simplifying assumptions are made. However, in multiagent systems, any deviation from full normativity has the potential to be catastrophic. In self-interested multiagent systems, the system designer attempts to create incentives so that the agents will behave in a socially beneficial way. If the models of the agents are not fully normative, then the designer may not be able to predict all the factors which agents base their actions on, and may fail to provide the correct incentives for the agents to act as desired. Even if the deviation of the agent is slight, since the actions taken by one agent can influence the action choice of other agents in the system, the final outcome may be arbitrarily far from expected.

2.5 Summary

In this chapter we introduced a model of bounded rationality in the form of computationally limited agents. We assume that agents must actively expend computational resources in order to determine their knowledge and preferences in the world. However, the agents have limitations on their computational resources which restricts their abilities to optimally compute or gather all the information that they might need. Instead, agents are deliberative in that they carefully decide how to best use their computing resources in the best possible way. To help in this decision process, agents use anytime algorithms coupled with performance profiles – statistical models that show how solution quality changes with the allocation of computing resources. We described different types of performance profiles, and explained how they are used to create deliberation control policies. In particular, we introduced the performance profile tree - a fully normative deliberation control method.

Chapter 3

Improving Deliberation Control: Experimental Results

In the previous chapter we introduced our model of a computationally limited agent and described the set of deliberation tools the agents could use in order to effectively use their computing resources. In particular, we introduced the performance profile tree, a fully normative deliberation control method for making stopping decisions for anytime algorithms.

While the full normativity of the performance profile tree is of theoretic interest, it has been unclear as to whether it is of practical use. In particular, it has been unclear as to whether this more sophisticated deliberation control approach created substantially better deliberation policies compared to other, simpler, commonly used approaches.

In this chapter we experimentally study the use of performance profile trees to determine their practicality and usefulness for helping a single agent decide when to stop its anytime optimization algorithm. On data generated from black-box anytime problem solvers, we illustrate that it is feasible to use performance profile trees in hard real-world problems. We also show that this leads to more accurate deliberation control decisions than the use of performance profile representations presented in prior literature. Furthermore, we illustrate that the performance profile tree can easily handle conditioning its deliberation policies on (the path of) other solution features, in addition to solution quality.

The rest of the chapter is organized as follows. We first review the performance profile methods used in the experiments. We then provide a description of the setup of the experiments. We describe the problem domains which we use and explain how the performance profiles are created. This is followed by a presentation and discussion of the results obtained from the experiments.

3.1 Decision-theoretic Deliberation Control

We begin by providing a quick overview of the deliberation control methods tested in this chapter. We refer the reader to Chapter 2 for a more detailed description of computationally limited agents and performance profile deliberation control.

We assume that agents have anytime algorithms and *time-dependent* utility functions. That is, the utility of an agent depends on both the solution quality obtained, and the amount of time spent getting it,

$$U(q(t), t) = q(t) - \mathrm{cost}(t)$$

where $q(t)$ is the utility to the agent of getting a solution with quality $q(t)$ and $\mathrm{cost}(t)$ is the cost incurred of computing for t time steps.

Anytime algorithms are paired with a meta-level deliberation controller that determines how long to run the anytime algorithm, that is, when to stop computing and act with the solution obtained. The deliberation controller's stopping policy is based on a performance profile: statistical information about the anytime algorithm's performance on prior problem instances. This helps the deliberation controller project how much (and how quickly) the solution quality would improve if further computation were allowed. Performance profiles are usually generated by prior runs of the anytime algorithm on different problem instances.

One of the performance profiles we study is what we call the *performance profile curve (PPCurve)* [12, 48]. Figure 3.1(a) is an example of a PPCurve. It is created by averaging, at each predefined time point, the solution quality obtained on prior runs (on different problem instances). The PPCurve is a static performance profile in that the amount of time allocated to the algorithm is determined before any computation takes place. Given the curve, along with a cost function, $\mathrm{cost}(t)$ the deliberation policy of the agent is to allocate time t^* to the algorithm, where

$$t^* = \arg\max_t \left[q(t) - \mathrm{cost}(t)\right].$$

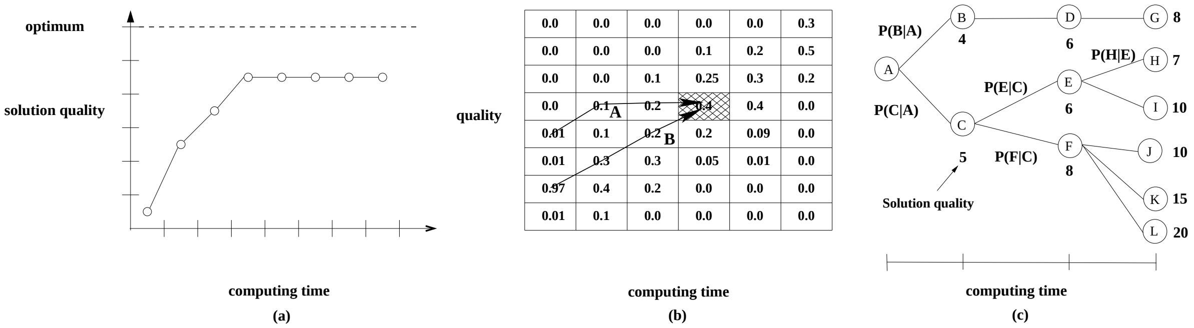


Figure 3.1: *Three performance profile representations: a) performance profile curve (PPCurve), b) performance profile table (PPTable), and c) a performance profile tree (PPTree).*

The second performance profile deliberation control procedure we study is the *performance profile table (PPTable)* (Figure 3.1(b)). A performance profile table is a table of discrete values which specify a discrete probability distribution over solution quality for each (predefined) time step. This is coupled with dynamic programming to produce (online) deliberation control policies. This is described in detail in the previous chapter.

The *performance profile tree, (PPTree)*, is a way to capture all information available for making stopping decisions [59]. In a PPTree, the nodes represent solutions at given time points, while each edge carries the probability that the child node is reached given that the parent was reached. Figure 3.1(c) exemplifies one such tree. A PPTree can support conditioning on any and all features that are deemed to be of importance for making stopping decisions since the nodes can hold information about solution quality and any other solution feature that may be important. A key feature of the PPTree is that it automatically supports conditioning on the *path* so far, which we believe is an important predictor of solution quality improvement.

3.2 Experimental Setup

The goal of the experimental work presented in this chapter was to determine;

1. Whether the performance profile tree based deliberation control method is feasible in practice, and

2. Whether in practice such sophisticated deliberation control is better than earlier decision-theoretic deliberation control methods that relied on simpler performance profile representations.

In the first set of experiments, we demonstrate both the feasibility and improved decision-making of the PPTree. In these experiments we use solution quality as the only feature stored in a tree node In the second set of experiments, we show that it is feasible to use additional problem features to make deliberation decisions.

Our deliberation control method is domain independent and domain problem solver independent—yielding a clean separation between the domain problem solver (a black box) and the deliberation controller. This separation allows one to develop deliberation control methodology that can be leveraged across applications. To demonstrate this we conduct experiments in two different application domains using software which was developed independently from the deliberation controllers.

3.2.1 Example Domain Problem Solving Environments

We conducted our experiments in two different domain environments – vehicle routing and single-machine manufacturing scheduling.

Vehicle Routing

In the real-world vehicle routing problem (VRP) in question, a dispatch center is responsible for a certain set of tasks (deliveries) and has a certain set of resources (trucks) to take care of them [106, 109]. Each truck has a depot, and each delivery has a pickup location and a drop-off location. The dispatch center's problem is to minimize transportation cost (driven distance) while still making all of its deliveries and honoring the following constraints:

- each vehicle has to begin and end its tour at its depot,
- each vehicle has a maximum load weight and maximum load volume constraint, and
- each delivery has to be included in the route of some vehicle.

To generate data for our experiments, an iterative improvement algorithm was used for solving the VRP. The center initially assigned deliveries to trucks in round-robin order. The algorithm then iteratively improved the solution by selecting a delivery at random, removing it from the solution, and then reinserting it into the least expensive place in the solution (potentially to a different truck, and with pickup potentially added into a different leg of the truck's route than the drop-off) without violating any of the constraints. Each addition-removal is considered one iteration. We let the algorithm run until there was no improvement in the solution for some predefined number, k, of steps. Figure 3.2 shows the results of several runs of this iterative improvement algorithm on different instances used in the experiments, with $k = 250$. The algorithm clearly displays diminishing returns to scale, as is expected from anytime algorithms.

A problem instance was defined to be a set of deliveries. The problem instances were generated using real-world data collected from a dispatch center that was responsible for 15 trucks and 300 deliveries. We generated training and testing sets by randomly dividing the deliveries into a set of 210 training deliveries and 90 testing deliveries. To generate a training (testing) instance, we randomly selected (with replacement) 60 deliveries from the training (testing) set.

Manufacturing Scheduling

The second domain is a single-machine manufacturing scheduling problem with sequence-dependent setup times on the machines, where the agent's objective is to minimize weighted tardiness

$$\sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(f_j - d_j, 0),$$

where T_j is the tardiness of job j, and w_j, f_j, d_j are the weight, finish time, and due-date of job j.

In our experiments, we used a state-of-the-art scheduler developed by others as the domain problem solver [19]. It is an iterative improvement algorithm that uses a scheduling algorithm called Heuristic Biased Stochastic Sampling [16]. We treated the domain problem solver as a black box without any modifications.

The problem instances were generated according to a standard benchmark [68]. The due-date tightness factor was set to 0.3 and the due-date range factor was set to

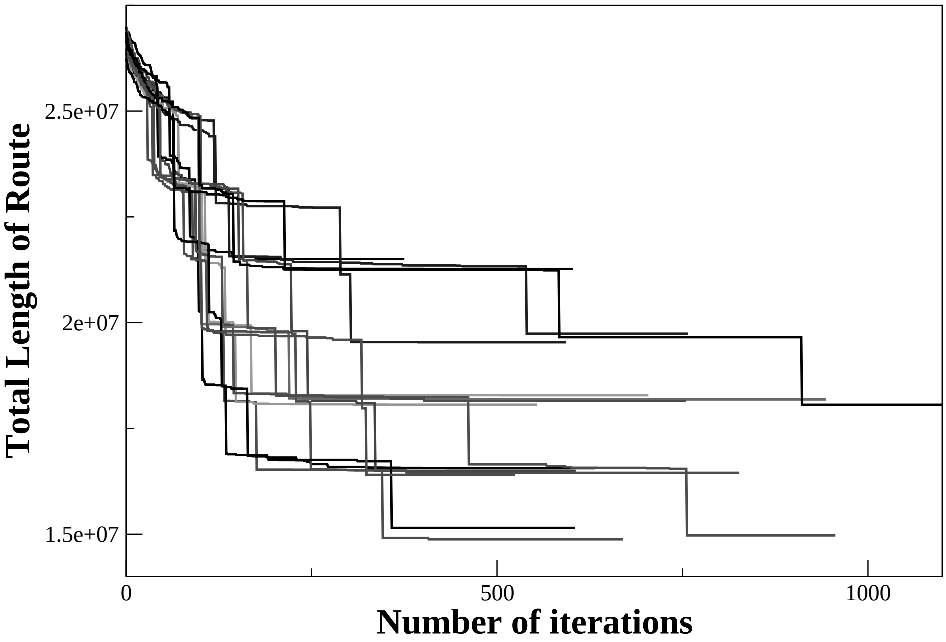


Figure 3.2: *Runs of the iterative improvement algorithm for the vehicle routing problem on different problem instances. The x-axis if the number of iterations of the algorithm and the y-axis is the total distance traveled by all trucks.*

0.25. The setup time severity was set to 0.25. Each instance consisted of 100 jobs to be scheduled. We generated the training instances and test instances using different random number seeds.

3.2.2 Constructing Performance Profiles

Performance profiles encapsulate statistical information about how the domain problem solver has performed on past problem instances. To build performance profiles, we generated

- 1000 instances for the vehicle routing domain. We ran the algorithm on each instance until there was no improvement in solution quality for 250 iterations.
- 10000 instances for the scheduling domain. We ran the algorithm until there was no improvement in solution quality for 400 iterations.

From this data, we generated the performance profiles using each of the three representations: PPCurve, PPTable, and PPTree. Like PPTable-based deliberation control, PPTree requires discretization of computation time and of solution quality (otherwise no two runs would generally populate the same part of the table/tree, in which case no useful statistical inferences could be drawn). The PPCurve does not require any discretization on solution quality, so we gave it the advantage of no discretization.

Computation time was discretized the same way for each of the three performance profile representations. We did this the obvious way in that one iteration of each algorithm (routing and scheduling) was equal to one computation step. For the solution quality we experimented with different discretizations. We present results where the scheduling data was discretized into buckets of width 100, and the vehicle routing data was discretized into buckets of width 50000. For the vehicle routing domain there turned out to be 1943 time steps, and 551 buckets of solution quality. For the scheduling domain there turned out to be 465 time steps, and 750 buckets of solution quality. The results obtained from these discretizations were representative of the results obtained across all the tested discretizations.

At first glance it may seem that this implies performance profile trees of size 551^{1943} for the trucking domain and 750^{465} for the scheduling domain. However, most of the paths of the tree were not populated by any run (because there are "only" 1000 (or 10000) runs, one per instance). We generated the tree dynamically in memory, such that only the populated parts were stored. This way, we could be assured that the number of edges in the tree for trucking was at most $1943 \times$ #instances (because each instance can generate at most one new edge for each compute step). Similarly, for scheduling it was $465 \times$ #instances.

In practice, the trees were easy to generate, but they are too large to display. Therefore, in Figure 3.3 we present a subtree of a performance profile tree generated from the same 10000 scheduling instances, but with a much coarser discretization: buckets of width 5000 on solution quality.

3.2.3 Cost Functions

In all the experiments we used cost functions of the form $C \cdot t$ where C was an exogenously given cost of one step of computing and t was the number of time steps

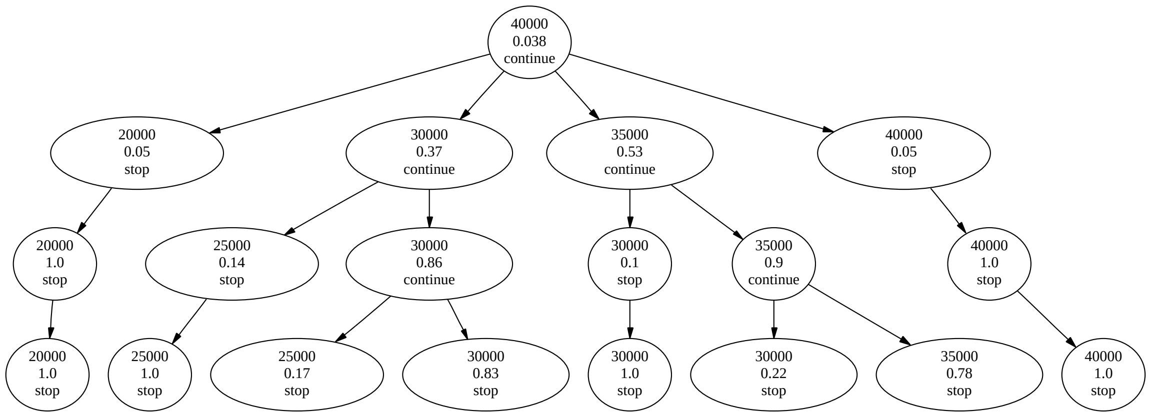


Figure 3.3: *Subtree of a performance profile tree generated from instances from the scheduling domain. In each node there are three entries. The first entry is the (discretized) solution quality. The second entry is the probability of reaching the node, given that its parent was reached. The final entry represents the stopping policy. An entry labeled "continue" means that the agent should compute another step, while an entry labeled "stop" means that the agent should stop all computation and act with the solution at hand.*

computed. We studied the behavior of deliberation control methods under a wide range of values of C. For the vehicle routing domain we used $C \in \{0, 10, 100, 500, 1000, 5000, 10000, 25000, 35000, 50000, 100000, 1000000\}$ In the scheduling domain we used $C \in \{0, 1, 10, 50, 100, 500, 1000, 5000, 10000, 100000\}$. These value choices span the interesting range: at the bottom end, no controller ever stops the deliberation, and at the top end, each controller stops it immediately.

3.3 Comparison of Performance Profiles

In the first set of experiments we tested the feasibility of PPTree-based deliberation control and compared its decision-making effectiveness against other performance-profile representations (PPCurve and PPTable).

To evaluate the performance, we generated $N = 500$ new test instances of the trucking problem and $N = 5000$ new test instances from the scheduling domain. Each of the three performance profile representations was evaluated on the same test set.

For each test instance we determined the optimal stopping point, t_i^{opt}, given that the entire path of the run was known in hindsight. (This stopping point is optimistic in the sense that it does not use discretization of solution quality. Furthermore, real deliberation controllers do not have hindsight at their disposal.) This allowed us to determine the optimal value that the agent could have possibly gotten on instance i in hindsight:

$$U_{\mathrm{opt}}(i) = q_i(t_i^{\mathrm{opt}}) + C \cdot t_i^{\mathrm{opt}}$$

where $q_i(t)$ is the solution quality after computing for t time steps, and C is the exogenously given cost of one step of computing. In both application domain, lower solution quality is better. Therefore, we add the cost function, instead of subtracting it in the utility equation.

We evaluated the three performance profile representations $P \in \{$PPTree, PPTable, PPCurve$\}$ separately on the test instances, recording for each instance the stopping point t_i^{P} that deliberation controller, P, imposed and the resulting value. That is, we stored

$$U_{\mathrm{P}}(i) = q_i(t_i^{\mathrm{P}}) + C \cdot t_i^{\mathrm{P}}.$$

We determined how far from optimal the resulting value was as a ratio

$$R_i^{\mathrm{P}} = \frac{U_{\mathrm{opt}}(i)}{U_{\mathrm{P}}(i)}.$$

Then,

$$R^{\mathrm{P}} = \frac{1}{N} \sum_i^N R_i^{\mathrm{P}}$$

gave an overall measure of performance (the closer R is to 1.0, the better). Figures 3.4 and 3.5 display the results.

When computation was free or exorbitantly expensive compared to the gains available from computing, then all the deliberation control methods performed (almost) optimally. With free computing, the deliberation control problem is trivial: it is simply best to compute forever. Similarly, when computation is extremely expensive, it is best to not compute at all. For the midrange costs (i.e., the interesting range), the deliberation controllers performed quite differently from each other. The PPTree outperformed both the PPCurve and the PPTable. In the vehicle routing experiments,

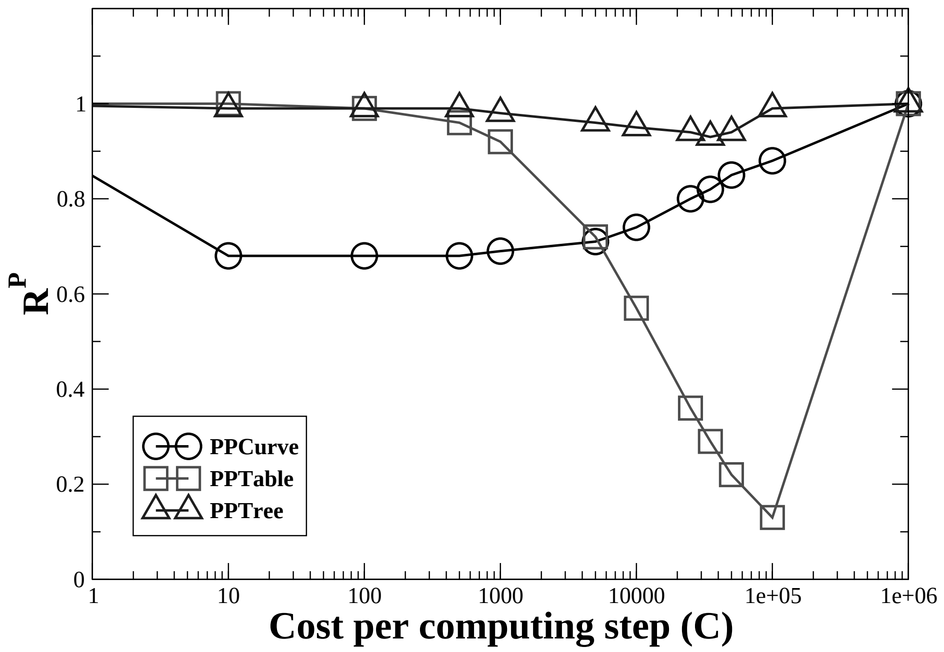


Figure 3.4: *Performance of the different performance profiles in the vehicle routing domain. Values closer to 1.0 (optimal) are better. PPTree outperforms both PPCurve and PPTable.*

the PPTree was, at worst, 93.0% of optimal (when $C = 35000$) and often performed better (Figure 3.4). In the scheduling experiments, the PPTree was always within 99.0% of optimal (Figure 3.5).

In the scheduling domain, the PPCurve performed reasonably well with R^P ranging from 0.95 to 1.00. In the vehicle routing domain, its performance was not as good, with R^P ranging from 0.68 to 1.00. A possible explanation for the difference is that in scheduling there was less variance (in the stopping time) among instances. Therefore, in the scheduling domain the optimal stopping point for the average algorithm run was a better estimate for any given run, compared to the routing domain.

The PPTable had the widest variability in behavior. For both low and high costs it performed well in both application domains. However, for midrange costs it performed poorly, as low as 0.07 in scheduling and 0.13 in vehicle routing. In particular, the PPTable appeared to be overly optimistic when determining a deliberation control policy, as it often allowed too much computing. It was not able to differentiate

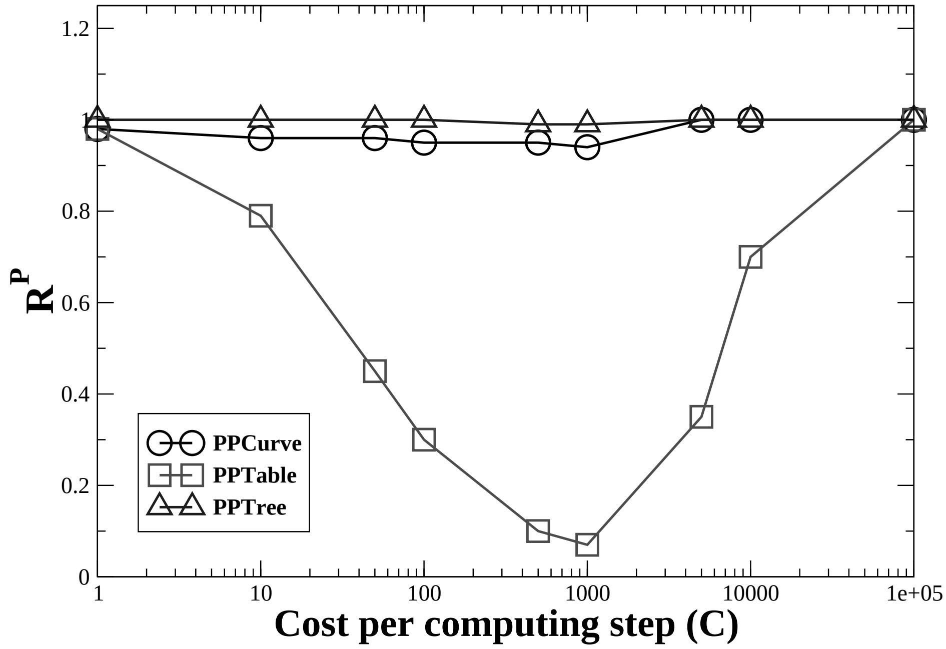


Figure 3.5: *Performance of the different performance profiles in the scheduling domain. Values closer to 1.0 are better. PPTree out performs both PPTable and PPCurve.*

between algorithm runs which had flattened out and ones in which additional improvement in solution quality was possible. While the dynamic programming approach used in the PPTable produces an optimal policy if solution quality improvement satisfies the Markov property, it appears to not be robust when this property does not hold.

While the PPTree performs better than PPCurve and PPTable with respect to absolute difference in performance, it is also important to determine whether there is statistical significance to the results. Using a Fisher sign test, since we were interested only in whether one performance profile made better deliberation control decisions and not in the magnitude of the difference in resulting solution quality, we compared the PPTree against both the PPCurve and the PPTable [121]. Table 3.1 reports the results of the test for the VRP. Table 3.2 reports the results of the test for the scheduling domain.

We found in both the trucking and scheduling domains that the dominance of the PPTree was truly significant, resulting in $p-$values less than 10^{-13} for all costs

Cost	n_+	n_-	p-value
0	0	0	–
10	500	0	6.11e-151
100	500	0	6.11e-151
500	500	0	6.11e-151
1000	500	0	6.11e-151
5000	500	0	6.11e-151
10000	500	0	6.11e-151
25000	500	0	6.11e-151
35000	493	7	9.21e-136
50000	466	34	4.12e-98
100000	493	7	9.21e-136
1000000	0	0	–

Cost	n_+	n_-	p-value
0	0	0	–
10	332	168	1.9e-13
100	432	68	8.34e-66
500	423	77	7.07e-59
1000	466	34	4.12e-98
5000	500	0	6.11e-151
10000	499	1	3.06-e148
25000	498	2	7.65e-146
35000	498	2	7.65e-146
50000	498	2	7.65e-146
100000	497	3	1.27e-143
1000000	0	0	–

Table 3.1: *Sign test results comparing PPTree against PPCurve (left) and PPTree against PPTable (right) in the vehicle routing domain. The first column in each table lists the cost constant C. The second column reports the number of test instances where $R^{\text{PPTree}} > R^{\text{PPCurve}}$ (left table) or $R^{\text{PPTree}} > R^{\text{PPCurve}}$ (right table). The third column in each table reports the number of instances where $R^{\text{PPTree}} < R^{\text{PPCurve}}$ (left table) or $R^{\text{PPTree}} < R^{\text{PPCurve}}$ (right table). The final column in each table reports the computed $p-$value.*

where there was any difference in the performance between the performance profiles. When costs were very low ($C = 0$) or very high ($C = 10^6$ in the trucking domain and $C = 10^5$ in the scheduling domain), all performance profiles performed optimally, and thus, identically.

We also experimented using different solution quality and time step discretizations. The general patterns seen in Figures 3.4 and 3.5 were again observed. Furthermore, we ran experiments where smaller training sets (of as low as 50 instances) were used. While the performance of all of the deliberation controllers was adversely affected, the relative ranking of their performance did not change.

In summary, this first set of experiments showed that PPTree-based deliberation control is feasible in practice and outperforms the earlier performance profile representations. It also showed that the method is close to optimal even when solution

Cost	n_+	n_-	p-value
0	0	0	–
1	500	0	6.11e-151
10	495	0	1.96e-149
50	495	5	1.58e-139
100	499	1	3.06e-148
500	477	23	1.77e-111
1000	460	40	1.5e-91
5000	458	42	1.85e-89
10000	0	0	–
100000	0	0	–

Cost	n_+	n_-	p-value
0	0	0	–
1	497	3	1.27e-143
10	499	0	1.22e-150
50	499	0	1.22e-150
100	499	0	1.22e-150
500	499	1	3.06e-148
1000	490	0	6.26e-148
5000	355	0	2.73e-107
10000	177	1	9.34e-52
100000	0	0	–

Table 3.2: *Sign test results comparing PPTree against PPCurve (left) and PPTree against PPTable (right) in the scheduling domain. The first column in each table lists the cost constant C. The second column reports the number of test instances where $R^{\text{PPTree}} > R^{\text{PPCurve}}$ (left table) or $R^{\text{PPTree}} > R^{\text{PPCurve}}$ (right table). The third column in each table reports the number of instances where $R^{\text{PPTree}} < R^{\text{PPCurve}}$ (left table) or $R^{\text{PPTree}} < R^{\text{PPCurve}}$ (right table). The final column in each table reports the computed $p-$value.*

quality and computation time are discretized and conditioning is done only on the path of the solution quality (instead of additional/all possible predictive features).

3.4 Conditioning on Additional Solution Features

Using the PPTree as our performance profile representation, we experimented using additional solution features to help in making deliberation control decisions. In the vehicle routing domain, in addition to solution quality (total distance driven) we also allowed conditioning on the following features:

1. variance across trucks in the number of tasks,

2. variance across trucks in the truck's average weight to max-weight ratio, and

3. variance across trucks in the truck's average volume to max-volume ratio.

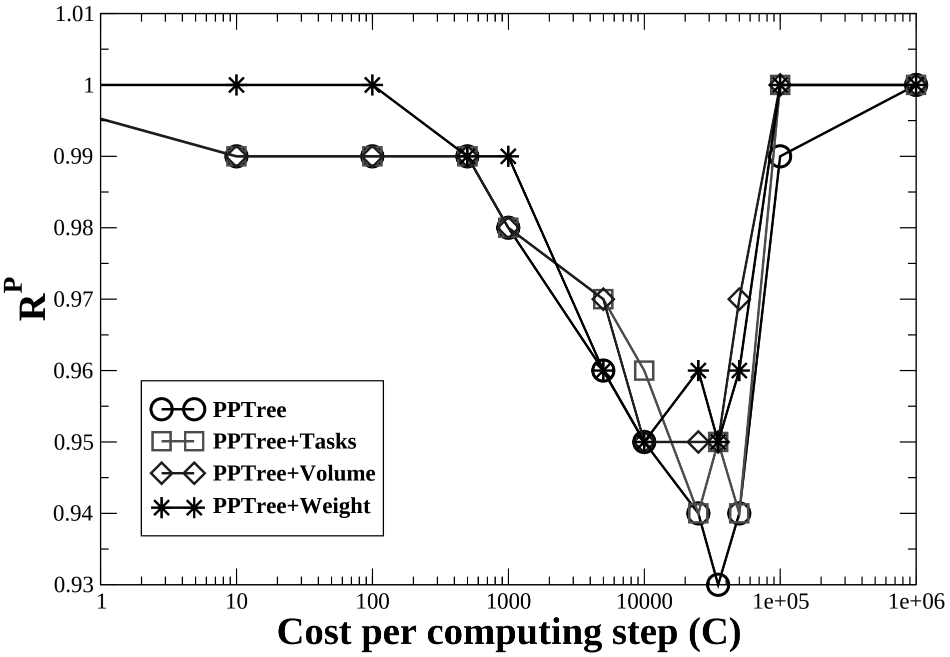


Figure 3.6: *Performance of the PPTree when deliberation-control decisions are conditioned on both solution quality and an additional solution feature (variance across trucks in number of tasks, variance across trucks of the average volume to max-volume ratio, and variance across trucks in the average weight to max-weight ratio.*

The results are reported in Figure 3.6. Using the additional solution features did lead to slight improvement in the quality of stopping decisions (a 3.0% absolute improvement for $C = 25000$; less improvement for other values of C). Similarly, in experiments conducted in the scheduling domain where slackness in the schedule was used as an additional feature, there was only slight improvement in the deliberation control results.

While the gains from using additional problem instance features in our experiments seem small, it must be recalled that the accuracy from conditioning on only solution quality was close to optimal already. For example when $C = 25000$ the overall performance improved from 94.0% to 97.0% of optimal. Using a Fisher sign test to analyze the significance of improvement in solution quality when using additional features, we found that for all features there was little significant improvement. Table 3.3 contains the results of the sign test on the stopping decisions made by the

PPTree+Weight compared to the PPTree using no additional problem features. We hypothesize that a ceiling effect is in action. In summary, the second set of experiments demonstrated that performance profile trees can support conditioning on (the path of) multiple solution features in practice.

Cost	n_+	n_-	p-value
10	272	212	0.00726
100	269	218	0.0234
500	268	220	0.0333
1000	249	239	0.684
5000	245	243	0.964
10000	230	257	0.239
25000	245	242	0.928
35000	246	242	0.892
50000	240	244	0.992
100000	163	70	10^{-9}

Table 3.3: *Sign test results comparing PPTree+Weight against PPTree (left) in the vehicle routing domain. The first column in each table lists the cost constant C. The second column reports the number of test instances where $R^{\text{PPTree+Weight}} > R^{\text{PPTree}}$. The third column in each table reports the number of instances where $R^{\text{PPTree+Weight}} < R^{\text{PPTree}}$. The final column in each table reports the computed $p-$value.*

3.5 Discussion

The results of our experiments illustrate that the performance profile tree is a powerful deliberation control tool. In this section we analyze some of the results in more detail and discuss how and whether the results of these experiments can be generalized.

While we expected the PPTree to outperform both the PPCurve and the PPTable in the experiments, we were surprised that the PPTable performed so poorly compared with the other two approaches. We hypothesize that there were two contributing factors to its poor performance; ignoring path information and overfitting the data.

Importance of Path Information

We believe that a major advantage of the PPTree is that it captures algorithm path information in a natural way, and uses this information when computing deliberation control policies. We observed that the PPTable tended to allocate too much time to runs, instead of stopping early and saving the computing cost, leading us to believe that it was not able to differentiate between runs where the path was promising with respect to further improvement, and runs where it was unlikely that further improvement in solution quality would take place. We use a small example to illustrate the problem we believe the PPTable encountered.

Assume that three algorithm runs are used to build the performance profile, where the runs are

1. $q(0) = 0$, $q(1) = 4$, $q(2) = 8$
2. $q(0) = 0$, $q(1) = 4$, $q(2) = 7$
3. $q(0) = 0$, $q(1) = 2$, $q(2) = 2$.

The performance profile tree generated from this data is presented in Figure 3.7. The

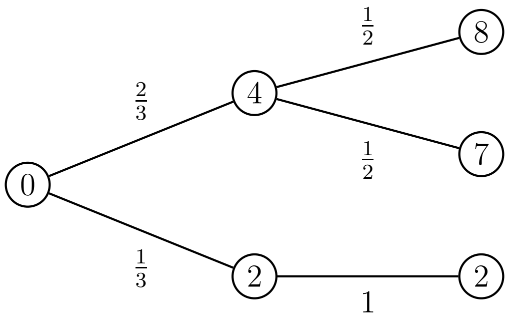


Figure 3.7: *A small performance profile tree.*

performance profile table generated from this data is presented in Figure 3.8.

Assuming that an agent has a cost function, $\text{cost}(t) = t$, the deliberation policy generated by the performance profile tree would specify that if, after one step of computation the solution quality is 2, the agent should stop computing on the problem. However, the deliberation policy generated by the performance profile table is different. If, after one step of computation, the solution quality is 2, the table is unable to deduce that additional computation will not improve the solution quality. Instead,

8.0	0	0	$\frac{1}{3}$
7.0	0	0	$\frac{1}{3}$
6.0	0	0	0
5.0	0	0	0
4.0	0	$\frac{2}{3}$	0
3.0	0	0	0
2.0	0	$\frac{1}{3}$	0
1.0	0	0	0
0.0	1	0	0

8.0	0	$\frac{1}{3}$
7.0	0	$\frac{1}{3}$
6.0	0	0
5.0	0	0
4.0	0	0
3.0	0	0
2.0	1	$\frac{1}{3}$

8.0	0	$\frac{1}{2}$
7.0	0	$\frac{1}{2}$
6.0	0	0
5.0	0	0
4.0	1	0

Figure 3.8: *The table on the left is the performance profile of the agent before it does any computing on the problem. The table in the center is the updated performance profile if the agent has computed for one step and has a solution quality of 2, and the table on the right is the performance profile of the agent after it has computed for one step and has obtained utility 4.*

due to its representation, it believes that in one more step of computation, solution quality 4, 7 and 8 are all equally likely, and thus, produces a deliberation control policy which states that the agent should continue computing on the problem. We hypothesize that the PPTable encountered similar situations in the experiments we conducted.

In our experiments the PPCurve performed quite well on average, even though it used no information on the current run of the algorithm. In particular, the PPCurve used no path information, which leads us to believe that while path information is clearly useful in deliberation control, it does not provide the entire answer as to why the PPTable performed so poorly. As to the strong behavior of the PPCurve, it appears as though the training and testing data used in the experiments was reasonable uniform so that calculating the best stopping point on average was an effective deliberation control policy.

Overfitting

Overfitting is a concern in performance profile based deliberation control as it may result in poorly performing deliberation control policies. In particular, overfitting occurs when an algorithm adapts so well to a training set that randomizations in the training set are included in the model as being meaningful, when in reality they are not. This can cause an algorithm to not perform as well on a test set. Often the problem of overfitting is caused by having too many parameters in the model. Ideally we want the performance profile to determine the underlying algorithm behavior, rather than the random errors and noise present in data. If there are too many parameters, the excess degrees of freedom allow the estimator to fit the random components of the training data as well, even though these are not representative of other data and fitting them makes the predictions worse. While there are ways of avoiding overfitting, a rule of thumb is that the more parameters there are in a model, the more training data is needed in order to avoid overfitting.

In theory, the performance profile tree has many more parameters than both the performance profile curve and the table-based approach, and thus, at first glance, would appear to be more likely suffer from the problem of overfitting. The performance profile curve (PPCurve) is a T parameter model, where T is the maximum number of time steps. For each time step, the only parameter of interest is the solution quality at that time step. The number of parameters in the PPTable is, in general, $Q(T-1)$ where Q is the number of buckets solution quality has been divided into and T is the maximum number of time steps captured in the table structure.[1] This is because for any run of an algorithm, there are $Q(T-1)$ possible entries (q_i, t_j) in which the algorithm could have passed through. In the PPTree the number of parameters is even larger as each leaf in the tree represents a path and thus a parameter. A naive estimate of the number of parameters is Q^T where Q is the number of buckets of solution quality and T is the number of time steps. In reality, the number of parameters in the PPTree model is slightly smaller. Experimentally, the PPTree is coupled with anytime algorithms which improve solution quality over time. Therefore, if solution quality is discretized into Q bins, then a node with solution quality in bin q has at most $Q-q$ children. Given any number of time steps, T, it is possible to count the number of leaves, $N(T)$, in the tree. If $T = 1$ then

$$N(Q,T) = Q$$

and for $T > 1$

$$N(Q,T) = \sum_{i_1}^{Q} \sum_{i_2}^{i_1} \dots \sum_{i_{T-1}}^{i_{T-2}} i_{T-1}.$$

[1]Recall that the PPTable and PPTree require that both solution quality and time are discretized, while the PPCurve does not require that solution quality is discretized.

While this formula is bounded above by Q^T, for many values of Q and T the actual number of leaves can be computed quickly, and is less than Q^T. For example, if $T = 7$ then the number of leaves for any value of Q is

$$\begin{aligned} N(Q,7) &= \frac{(Q+1)^7}{5040} + \frac{(Q+1)^6}{360} + \frac{(Q+1)^5}{72} + \frac{(Q+1)^4}{36} + \frac{7(Q+1)^3}{720} \\ &\quad - \frac{11(Q+1)^2}{360} - \frac{Q}{42} - \frac{1}{42}. \end{aligned}$$

As mentioned in an earlier paragraph, one way to avoid overfitting is to ensure that enough sample data has been used in building the performance profile. Since the PPTree has more parameters than the PPTable and PPCurve, in theory, more data is required to make it robust against overfitting. However, as users of the PPTree there is another trick which can be used in order to reduce concerns with overfitting. By appropriately discretizing solution quality it may be possible to reduce the amount of noise in the data, while also reducing the number of parameters in the tree. This is discussed in the next subsection.

If overfitting played the only role in the results of our experiments, then we would have expected the deliberation-control decisions of the PPCurve to be better than the PPTable and the PPTree. Instead, this was not the case as the PPTree outperformed the two other approaches. While we do not have a definitive explanation for the behavior of the different deliberation control procedures, we believe that their behavior was influenced by a variety of factors. First, we believe that path information was key in the performance of the PPTree (as previously discussed), but that overfitting also played an important secondary role, leading to the strong behavior of the robust PPCurve.

Discretization of Solution Quality

In all the experiments described in this chapter, a uniform, coarse discretization of solution quality was used. This was done in order to make computing the deliberation control policies of the agents faster as well as allowing us to populate the tree adequately with the data available. We conducted a small set of experiments where we varied the discretization of the solution quality (both refining and coarsening it for interesting cost functions), but observed no qualitative difference in performance.

We believe that an interesting research direction would be to study the importance of discretization with respect to the quality of deliberation control policies. There are several complementary directions in which this work could lead. First, there is an interesting tradeoff between coarseness of discretization, amount of data needed in order for the performance profile trees to be useful in predicting algorithm behavior, and the ability to quickly compute deliberation control policies. For example, coarse discretizations lead to smaller trees which mean that computing deliberation policies is not difficult. However, with too coarse a discretization, the predictive power of the performance profile may be lost. On the other hand, a fine discretization would result in larger trees, leading to potential overfitting problems, as well as causing the overhead of computing the deliberation control policies to increase. Additionally, the amount of data required to adequately populate a finely discretized tree could potentially be enormous, reducing the usefulness of such an approach. A second direction concerning discretization of solution quality would be to move away from using a uniform discretization, and instead vary the discretization based on the number of computing steps taken. For example, it may be useful to initially start with a coarse discretization in order to roughly categorize the algorithm runs into high and low solution quality runs, and then, only after computing for several steps, refine the discretization in order to better reflect the data. Learning techniques may be useful in determining optimal discretization levels.

Reliance on Algorithms

A possible question of the results reported in this chapter concerns whether behavior of the performance profiles studied on this chapter rely heavily on the actual algorithms used. Since we conducted the experiments using data and algorithms from two distinct domains we are confident that our findings will generalize to a wide range of settings, however it would be interesting to study what happens when a different style of algorithm, other than iterative improvement algorithms, are used. Theoretically, the PPTree is a generalization of both the PPTable and PPCurve. Thus, we believe that at worst the behavior of the PPTree should reduce to one of the two other performance profile representations, and whenever additional information (such as path) is useful, the PPTree can take advantage of it.

Multiple Features

In some of our experiments we used multiple features when determining the deliberation control policies. However we observed no significant improvement in the quality of deliberation control decisions made. We believe that we hit a "ceiling effect" in that path of solution quality was the major determining indicator of improvement. It is likely that in other domains conditioning on additional problem features will prove to be a useful tool. We believe that an interesting research direction could be to learn which features are useful in different domains, and thus tailor the performance profile trees to the domains in order to get the maximum performance possible.

3.6 Summary

We proposed performance profile tree based stopping control of anytime optimization algorithms as a theoretical basis for fully normative deliberation control. In this chapter, we compared different performance profile representations in practice, and showed that the performance profile tree is not only feasible to construct and use, but also leads to better deliberation control decisions than prior methods. We conducted experiments were deliberation control decisions were conditioned on (the path of) multiple solution features (not just solution quality), again demonstrating feasibility of that idea.

Section II: The Impact of Computational Limitations on Negotiation

Chapter 4

Game Theory and Mechanism Design

In the first section of this dissertation we presented our model of a computationally limited agent and described a normative model of deliberation control. In the rest of this dissertation we assume that we are in a setting where there are multiple computationally limited agents interacting. Using the tools provided to us by game theory and mechanism design, we study the impact that computational limitations has on the strategic behavior of agents in different settings. In this chapter we review the game theory and mechanism design concepts needed for the rest of this thesis.

Game theory is a branch of mathematics which studies the interactions of agents (sometimes referred to as players). Game theory has been successfully applied in such diverse subjects such as economics, to evolutionary biology [114]. Mechanism design is a sub-field of game theory and microeconomics which studies the problem of implementing good system-wide solutions to problems when the outcome depends on the actions of self-interested agents. Mechanism design has been successfully used in many important applications such as electronic market design and resource allocation problems.

In this chapter, we provide an overview of key game theoretic concepts, as well as mechanism design principles used in the rest of this dissertation. This chapter is not meant to be a complete overview of game theory and mechanism design. For a more general introduction to game theory, Osborne and Rubinstein provide a good reference [86]. For an overview of mechanism design, we suggest Mas-Colell *et al* [71]. Varian describe the role of mechanism design in systems with computational agents [117], while Papadimitriou discusses the use of game theory in interesting Internet applications [87].

4.1 Game Theory

A central feature of many multiagent interaction settings is *strategic interdependence*. The utility of each agent depends not only on its own actions, but also on the actions takes by other agents in the system. In particular, the actions that are best for an agent to take may depend on actions other agents have already taken, or on actions that the agent believes others have taken, or even actions the agent believes the others will take at some future point in time. The mathematical tool used to study these situations of strategic interaction is *game theory*.

4.1.1 Basic Definitions

A *game* consists of a set of agents, N, ($|N| = n$), a set of actions, A_i, for each agent $i \in N$, and a set of outcomes, O. The key concept in game theory is a *strategy*.

Definition 11 (Strategy). *A* strategy *for agent i, denoted by s_i, is a contingency plan that specifies the action an agent should take at every point in the game when it has to take an action.*

Strategies can be either *pure* or *mixed*. Pure strategies are deterministic plans. A mixed strategy, written $\tilde{s}_i \in \Delta(S_i)$ is a probability distribution over the set of all pure strategies for agent i, S_i.

A *strategy profile*, $s = (s_1, \ldots, s_n)$, is a vector specifying one strategy for each agent in the game. As is standard, we use the notation $s = (s_i, s_{-i})$ to denote the strategy profile where the strategy of agent i is s_i and $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$. The strategy profile, s, determines how the game is played and thus which outcome, $o(s) \in \mathcal{O}$, occurs. Each agent i tries to play a strategy such that its preferred outcome occurs. We assume that agents' preferences are expressed in terms of *utility functions*.

Definition 12 (Utility Function). *The* utility function *of agent i, $u_i(\cdot)$, is a mapping from outcomes to the real numbers;*

$$u_i : \mathcal{O} \mapsto \mathbb{R}.$$

		Player 2: A	Player 2: B
Player 1	A	1, 2	0, 0
	B	0. 0	2, 1

Figure 4.1: *A normal form game. The two agents can either play strategy A or strategy B. The payoffs for each strategy profile are listed in the table. For example, the upper left hand corner entry corresponds to both players choosing strategy A. In this case, the utility of Player 1 is 1 and the utility Player 2 is 2.*

An agent i prefers outcome o_1 to outcome o_2 if $u_i(o_1) > u_i(o_2)$. As is commonly done, when it is clear from the context, we will use the notation $u_i(s_i, s_{-i})$ to denote $u_i(o(s_i, s_{-i}))$.

The basic model of rationality in game theory is *expected utility maximizers*. An agent will select a strategy that maximizes its expected utility, given its preferences over outcomes, its beliefs about the strategies other agents are playing, and the structure of the game.

Games can be either static or dynamic. A static game is one in which all players select a strategy simultaneously, without knowledge of the strategies chosen by the other agents. Static games are often represented by the *normal form*, a table which specifies the payoffs to the agents if they play different combinations of strategies (Figure 4.1).

Dynamic games, where the game progresses over a series of steps or stages, are usually represented in *extensive form*. The extensive form captures who moves when, what actions each agent can take, what information agents have when they move, what the outcome is as a function of the actions taken by the agents, and what the payoffs are for each possible outcome.

An extensive game contains the following information.

- A set of agents, N,
- The order of the moves (histories)
- The agents' utilities,

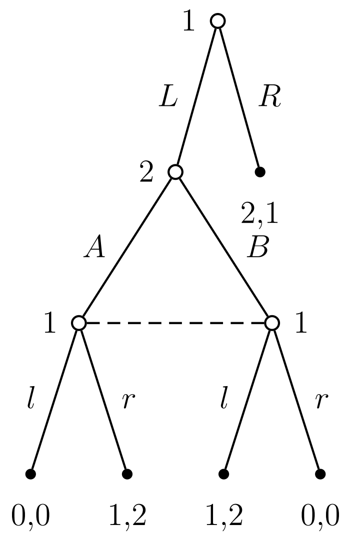


Figure 4.2: *An extensive form game with two players.*

- The actions available to each agent when it is its turn to move,
- The information each agent has available to it when it moves (information sets)

Extensive form games are usually represented by tree structures. They consist of a finite number of nodes and provide a complete description of all actions previously taken. Nodes which do not have children are called terminal nodes and are assigned agents' utilities. Nodes where an agent must select an action are called decision nodes. Each decision node is associated with a player, and the edges emanating from the node correspond to the actions that the agent is allowed to take at that point in the game. The nodes in the tree are partitioned into information sets, and any two nodes in the same information set must have the same choice of actions. If two nodes are in the same information set, then the agent can not fully distinguish between them. Instead, the agent has a probability distribution over the nodes, specifying the agent's belief that it is at a specific node. Figure 4.2 is an example of an extensive form game. In this game there are two players, agent 1 and agent 2. Agent 1 moves first, and chooses either action L or action R. Agent 2 observes this move. If agent 1 chose R then the game ends, and the payment to agent 1 is 2 and the payment to agent 2 is 1. If agent 1 chose L then agent 2 is allowed to move. It can choose to take either action A or action B. However, agent 1 does not observe which action agent 2 selects. When it is its turn to move again, agent 1 can not distinguish between the two nodes in its information set. All it knows is that it can either take move l or move r.

4.1.2 Solution Concepts

A key goal of game theory is to find the stable points in the space of strategy profiles. These stable points are the *equilibria* of the game. The most well known equilibrium concept is the Nash equilibria [80]. A Nash equilibrium is a strategy profile in which each agent is playing its optimal strategy, given the strategies the other agents are playing.

Definition 13 (Nash Equilibrium). *A strategy profile $s^* = (s_1^*, \dots, s_n^*)$ is a* Nash equilibrium *if no agent has incentive to deviate from its strategy, given that the other agents do not deviate. Formally,*

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

The definition above is technically for settings where the agents are playing pure strategies. The Nash equilibrium solution concept easily extends to mixed strategies. If agents are playing mixed strategies then the *expected* utility must be at least as large as that obtainable by any other strategy. A seminal result in game theory is that every finite game has at least one Nash equilibrium [80].

While the Nash equilibrium is one of the fundamental concepts in game theory, it does have several weaknesses. First, there may be multiple Nash equilibria in a game, and so agents may be uncertain as to which equilibrium to play. Second, the Nash equilibrium solution concept assumes that agents have perfect information about all agents in the game, that this is common knowledge, and that all agents are playing the same Nash equilibrium. In many settings these assumptions are too strong – limiting the practicality of this solution concept.

A stronger solution concept is the *dominant strategy equilibrium.* A strategy is dominant if it is an agent's best strategy against any strategies that the other agents may use.

Definition 14 (Dominant Strategy). *The strategy of agent i, s_i^*, is* dominant *if*

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

A dominant strategy equilibrium is an equilibrium in which every agent has a dominant strategy. The dominant strategy equilibrium is a robust solution concept since it makes no assumptions about the information that agents have available to them, nor does it assume that all agents know that the others will play rationally. However, many games do not have dominant strategy equilibria.

As mentioned earlier, the Nash equilibrium solution concept assumes that agents are fully informed about all aspects of the game, including the utility each agent receives from different outcomes. In many situations this is too strong an assumption. Instead, agents may only have beliefs about the preferences of other agents. This uncertainty can be captured in a *Bayesian game*. A Bayesian game can be modeled by assuming that each agent's preferences are determined by the realization of a random variable. In particular, it is assumed that some non-strategic player, called Nature, makes the first move in the game and chooses realizations of the random variable that determine each agent's preference type, $\theta_i \in \Theta_i$, where Θ_i is the set of all possible preference types for agent i. Each agent i observes θ_i, which determines its utility function $u_i((s_i, s_{-i}), \theta_i)$, but does not observe θ_j for any $j \neq i$. The joint probability distribution of the θ_i's is given by $p(\theta_1, \ldots, \theta_n)$. This is common knowledge.

A pure strategy in a Bayesian game, $s_i(\cdot)$, specifies the agent's strategy choice for each realization of θ_i. The utility of agent i given strategy profile $(s_i(\cdot), s_{-i}(\cdot))$ is

$$\overline{u_i}(s_i, s_{-i}) = \sum_{\theta_i} \sum_{\theta_{-i}} p(\theta_i, \theta_{-i}) u_i(s_i(\theta_i), s_{-i}(\theta_{-i})))$$

where $p_i(\theta_i, \theta_{-i})$ is the common prior over types.

The Bayes-Nash equilibrium is defined for Bayesian games.

Definition 15 (Bayes-Nash Equilibrium). *A strategy profile, s^* is a* Bayes-Nash equilibrium *if*

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

The solution concepts introduced so far – Nash, dominant strategy, and Bayes-Nash – apply in both static and dynamic games. There are several solution concepts specifically associated with extensive form games, including subgame perfect Nash equilibria, and sequential equilibria. The solution concept which we use in this dissertation is the perfect Bayesian equilibria (PBE). A key component in a PBE is the *system of beliefs*.

Definition 16 (System of beliefs). *A* system of beliefs*, μ, in extensive form game Γ is a specification of a probability $\mu(x) \in [0, 1]$ for each decision node x in Γ such* *that*

$$\sum_{x \in I} \mu(x) = 1$$

for all information sets I.

A system of beliefs specifies the relative likelihood that a specific node in an information set has been reached.

Another key concept required for a PBE is a *sequentially rational* strategy profile.

Definition 17 (Sequentially rational). *A strategy profile $s = (s_1, \ldots, s_n)$ in extensive form game Γ is sequentially rational for information set I given system of beliefs μ if*

$$E[u_i|I, \mu, s_i, s_{-i}] \geq E[u_i|I, \mu, s'_i, s_{-i}] \; \forall s'_i \neq s_i$$

where $E[u_i|I, \mu, s_i, s_{-i}]$ is the expected utility of agent i who's turn it is to move, given that it has beliefs μ. A strategy profile is sequentially rational *if this holds for all information sets.*

We are now able to define a perfect Bayesian equilibrium.

Definition 18. *A strategy profile and system of beliefs, (s, μ) is a (weak)* perfect Bayesian equilibrium *in extensive form game Γ if the following holds:*

1. *The strategy profile, s, is sequentially rational given belief system μ.*
2. *The system of beliefs, μ, is derived from strategy profile s through Bayes rule when possible. That is, for any information set I such that the probability of reaching I is positive under strategy profile s, it must be*

$$\mu(x) = \frac{Pr(x|s)}{Pr(I|s)} \text{ for all } x \in I.$$

4.2 Mechanism Design

In this section we present an overview of pertinent mechanism design concepts. For a broader overview we recommend Mas-Colell *et al* [71] and Myerson [78].

We assume that there is a set of agents, N, $|N| = n$. Each agent, i, has a *type*, $\theta_i \in \Theta_i$, which represents the private information of the agent that is relevant to the agent's decision making. In particular, an agent's type determines its preferences over different outcomes. We use the notation $u_i(o, \theta_i)$ to denote the utility of agent i with type θ_i for outcome $o \in \mathcal{O}$ ($\mathcal{O}$ is the space of possible outcomes). The goal of mechanism design is to implement some system-wide solution. This is defined in terms of a *social choice function*.

Definition 19 (Social Choice Function). *A* social choice function *is a function $f : \Theta_1 \times \ldots \times \Theta_n \mapsto \mathcal{O}$, that, for each possible profile of agents' types, $\theta = (\theta_1, \ldots, \theta_n)$, it assigns an outcome $f(\theta) \in \mathcal{O}$.*

The mechanism design problem is to implement a set of "rules" so that the solution to the social choice function is implemented despite agents' acting in their own self-interest.

Definition 20 (Mechanism). *A mechanism $M = (S_1, \ldots, S_n, g(\cdot))$ defines the set of strategies S_i available to each agent and an outcome rule $g : S_1 \times \ldots \times S_n :\mapsto \mathcal{O}$, such that $g(s)$ is the outcome implemented by the mechanism for strategy profile $s = (s_1, \ldots, s_n)$.*

A mechanism *implements* a social choice function $f(\cdot)$ if there is an equilibrium of the game induced by the mechanism which results in the same outcomes as $f(\cdot)$ for every profile of types, θ.

Definition 21 (Implementation). *A mechanism $M = (S_1, \ldots, S_I, g(\cdot))$* implements *social choice function $f(\cdot)$ if there is an equilibrium strategy profile $s^* = (s_1^*, \ldots, s_n^*)$ such that $g(s^*(\theta)) = f(\theta)$ for all θ.*

The equilibrium concept is left undefined at the moment. It can be Nash, Bayesian-Nash, dominant or some other equilibrium concept. Ideally, one wishes to implement a social choice function using as strong a solution concept as possible. In particular, dominant strategy implementation is preferred as it makes less assumptions about the participating agents.

The question as to what social choice functions are implementable seems to be overwhelming. However, due to a key result, the *Revelation Principle*, we need only restrict ourselves to a very small class of mechanisms, called *direct mechanisms*.

Definition 22 (Direct mechanism). *A* direct mechanism *is a mechanism in which $S_i = \theta_i$ and $g(\theta) = f(\theta)$ for all θ.*

In other words, direct mechanisms are mechanisms where agents are asked to reveal their type, and given an announcement $(\hat{\theta}_1, \ldots, \hat{\theta}_n)$, $f((\hat{\theta}_1, \ldots, \hat{\theta}_n))$ is chosen.

An example of a direct mechanism is a first-price sealed bid auction. In the auction the agents are asked to submit bids to the auctioneer. These bids represent the type of the agent. The mechanism then determines an allocation based on the announcements received, where the highest bidder is allocated the item, and pays the amount that it announced.

An important class of direct mechanisms are ones where truth telling is an optimal strategy for each agent. The social choice functions implemented by such mechanisms are said to be *incentive compatible*.

Definition 23 (Incentive compatible). *A social choice function is* incentive compatible *if the direct revelation mechanism $M = (\Theta_1, \ldots, \Theta_n, f(\cdot))$ has an equilibrium $(s_1^*(\cdot), \ldots, s_n^*(\cdot))$ where $s_i^*(\theta_i) = \theta_i$ for all $\theta_i \in \Theta_i$ and for all i.*

In other words, a social choice function is incentive compatible if truth telling by each agent constitutes an equilibrium. If the equilibrium concept is dominant-strategy, then the mechanism is said to be *strategy-proof*.

Another important mechanism property is *individual rationality*. Agents usually have the freedom to choose whether they wish to participate in the mechanism. If the utility that they can achieve by not participating is greater than what they can obtain through the mechanism, then the mechanism is not individually rational.

Definition 24 (Individual rationality). *A mechanism is* individual-rational *if for all types θ_i, it implements a social choice function $f(\theta)$ such that*

$$u_i(f(\theta_i, \theta_{-i})) \geq \overline{u_i}(\theta_i)$$

where $\overline{u_i}(\theta_i)$ is the utility the agent could get with non-participation.

4.2.1 The Revelation Principle

The revelation principle is one of the fundamental results in mechanism design. It states that under very weak conditions, mechanism designers need to only focus on incentive-compatible direct mechanisms in order to determine which social choice functions are possible to implement, and which are not possible. The revelation principle was first proposed by Gibbard [37], and was extended by Green and Laffont [40] and Myerson [77].

The revelation principle for dominant strategies states that any social choice function which is implementable in dominant strategies, is also implementable in a strategy-proof mechanism. That is, a designer need only consider social choice functions which can be implemented in truth-telling direct mechanisms.

Theorem 1 (Revelation principle). *Suppose that there exists a mechanism $M = (S_1, \ldots, S_n, g(\cdot))$ that implements a social choice function $f(\cdot)$ in dominant strategies. Then $f(\cdot)$ is truthfully implementable in dominant strategies.*

A similar revelation principle holds for situations where a social choice function is implementable in Bayesian-Nash equilibrium.

Theorem 2 (Bayes-Nash revelation principle). *Suppose that there exists a mechanism $M = (S_1, \ldots, S_n, g(\cdot))$ that implements a social choice function $f(\cdot)$ in Bayesian-Nash equilibrium. Then $f(\cdot)$ is truthfully implementable in Bayesian-Nash equilibrium.*

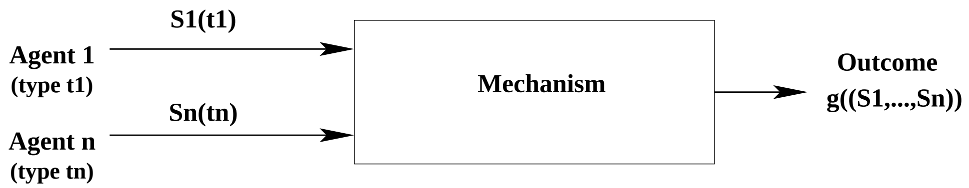
The intuition behind the proofs of both revelation principles is similar. Assume that it is possible to build a simulator which will execute an agent's optimal strategy in the original mechanism, given its type. Then, in theory, it is possible to create a new mechanism which incorporates this simulator into the design, and so that agent need only reveal its type in order for the optimal strategy to be executed. Figure 4.3 illustrates this.

4.2.2 Quasi-Linear Preferences

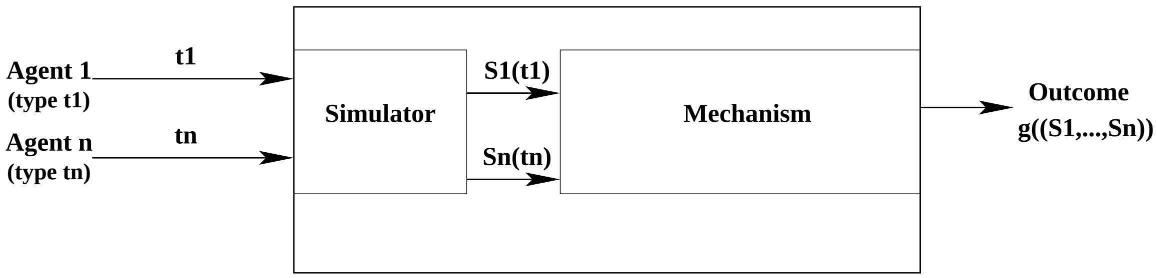
While mechanisms can be implemented across a wide spectrum of environments, in this dissertation we restrict ourselves to settings where agents are risk neutral and have *quasi-linear preferences.*

Definition 25 (Quasi-linear Preferences). *A quasi-linear utility function for agent i with type θ_i is of the form:*

$$u_i(o, \theta_i) = v_i(x, \theta_i) + t_i$$



Original Mechanism



New (direct revelation) Mechanism

Figure 4.3: *The Revelation Principle states that any social choice function which is implementable in dominant strategies can be implemented by a truth-telling direct mechanism. In particular, it is possible to build a simulator which simulates the optimal strategy an agent would play, if given the agent's type.*

where outcome o defines a choice $x \in \mathcal{K}$ from a discrete choice set $\mathcal{K}$ and a transfer t_i by the agent. The notation $v_i(x, \theta_i)$ represents the valuation function of agent i, that is, the value the agent places on $x \in \mathcal{K}$.

An advantage of quasi-linear preferences is that it allows for the transfer utility across agents by using side payments. Many real-world settings are ones where the participants have quasi-linear preferences. For example, in a single item auction, the outcome of the auction is an allocation of the item to an agent along with the payments that the agents have to make. If an agent i has value v_i for the item, then its utility, if it is allocated the item, is $u_i = v_i - p$ where p is the price it must pay for the item.

With quasi-linear preferences, it is possible to separate the outcome of a social choice function, into a choice, $x(\theta)$, which effects the values of the agents, and the transfers t_i. This allows one to define a general mechanism for quasi-linear preferences.

Definition 26 (Mechanisms for quasi-linear environments). *A mechanism for quasi-linear environments is a mechanism $M = (S_1, \ldots, S_n, (k(\cdot), t_1(\cdot), \ldots, t_n(\cdot)))$ such that the outcome function $g(\cdot) = (k(\cdot), t_1(\cdot), \ldots, t_n(\cdot))$ where $k : S_1 \times \ldots \times S_n :\mapsto \mathcal{K}$ is a choice rule which selects some choice from choice set $\mathcal{K}$, and transfer rules $t_i : S_1 \times \ldots \times S_n :\mapsto \mathbb{R}$. one for each agent, compute the payment $t_i(s)$ made by agent i.*

An important property of a social choice function is whether it is *efficient*.

Definition 27 (Efficient). *A social choice function $f(x(\theta), t(\theta))$ is* efficient *if for all types $\theta = (\theta_1, \ldots, \theta_n)$*

$$\sum_{i=1}^{n} v_i(x(\theta), \theta_i) \geq \sum_{i=1}^{n} v_i(x'(\theta), \theta_i) \; \forall x'(\theta) \in \mathcal{K}.$$

We will often use the phrase *social welfare maximizing* in place of *efficient* since in the quasi-linear environment, an efficient allocation maximizes social welfare.

Another property of interest is whether the social choice function is *budget balanced.*

Definition 28 (Budget balanced). *A social choice function $f(\theta) = (x(\theta), t(\theta))$ is* budget-balanced *if for all preferences $\theta = (\theta_1, \ldots, \theta_n)$*

$$\sum_{i=0}^{n} t_i(\theta) = 0.$$

If a social choice function is budget-balanced then there is no net payments being made into the system or being taken out of the system. In many situations budget-balance is too strong a condition. Therefore, often one resorts to a notion of weak budget-balance.

Definition 29 (Weak budget balance). *A social choice function $f(\theta) = (x(\theta), t(\theta))$ is* weakly budget-balanced *if for all preferences $\theta = (\theta_1, \ldots, \theta_n)$*

$$\sum_{i=0}^{n} t_i(\theta) \geq 0.$$

In a weak budget-balance setting, there may be a net payment being made by the agents to the mechanism center, but the center is not required to subsidize the agents.

4.2.3 Vickrey-Clarke-Groves Mechanisms

An important, and widely studied, family of mechanisms are the Vickrey-Clarke-Groves mechanisms (VCG) [20, 42, 118]. These quasi-linear mechanisms are efficient and strategy-proof direct mechanisms. In fact, it has been shown that the family of VCG mechanisms are the *only* quasi-linear mechanisms that are both efficient and strategy-proof among all direct mechanisms [40].

In a VCG mechanism each agent reports a type $\hat{\theta}_i$ to the mechanism. This type is not required to be its true type. Given the reported types, the mechanism produces an allocation $k^*(\hat{\theta})$ which is efficient. That is

$$k^*(\hat{\theta}) = \arg\max_{x \in \mathcal{K}} \sum_i v_i(x, \hat{\theta}_i).$$

The payment rules of the VCG mechanism are defined as

$$t_i(\hat{\theta}) = h_i(\hat{\theta}_{-i}) - \sum_{j \neq i} v_j(k^*, \hat{\theta}_j)$$

where $h_i : \Theta_{-i} \mapsto \mathbb{R}$ is an arbitrary function which does not depend on the declared type of agent i.

Clearly, a VCG mechanism is allocatively-efficient since the choice rule, k, is defined to be the one which produces the efficient outcome. A VCG mechanism is incentive-compatible since the announcements of the agents only influence the allocation, and not the their own transfers.

The Vickrey Auction

A special case of a VCG mechanism is the Vickrey auction. The Vickrey auction, also known as a second-price sealed-bid auction, allocates a single item to one agent out of a set [118]. Each agent i submits a bid b_i to the mechanism. The mechanism allocates the item to the agent with the highest bid. If an agent is not allocated the item then it pays nothing. If the agent is the winner, then it pays an amount equal to the second highest bid.

The Vickrey auction is incentive-compatible. An agent has no incentive to submit a bid (b_i) higher than its true value (v_i) for the item. If agent i had not been allocated the item when it announced v_i then there must have been another agent j with bid

b_j such that $b_j > v_i$. If agent i submits a bid $b_i > b_j$ then, even though it is allocated the item, it must pay b_j. Its utility would be $v_i - b_j < 0$. On the other hand, agent i has no incentive to submit a bid $b_i < v_i$. Lowering the bid below its true value only reduces the chances that agent i will be allocated the item. It does not change the price it would have to pay if it did win the auction. The Vickrey auction is also efficient, since it allocates the item to the agent who values it the most.

Pivotal Mechanism

Another important VCG mechanism is the Pivotal, or Clarke, mechanism [20]. The allocation function in the Clarke mechanism, as in all VCG mechanisms, is chosen so as to maximize the sum of the agents valuations, given their declared types. What defines the Clarke mechanism is the additional transfer term $h_i(\cdot)$ defined as

$$h_i(\hat{\theta}_{-i}) = \sum_{j \neq i} v_j(k^*_{-i}(\hat{\theta}_{-i}), \hat{\theta}_j)$$

where $k^*_{-i}(\hat{\theta}_{-i})$ is defined to be the optimal allocation when agent i does not participate. That is

$$k^*_{-i}(\hat{\theta}_{-i}) = \arg\max_{x \in \mathcal{K}} \sum_{j \neq i} v_j(x, \hat{\theta}_j).$$

This is a valid transfer function since it is independent of the declaration of agent i. This means that this mechanism is a member of the family of VCG mechanisms.

An interesting application of the Pivotal mechanism is in combinatorial auctions. 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 $X = \{x_1, \ldots, x_m\}$ be a set of items. A bundle is a subset of the items, for example, $\{x_1\}$ or $\{x_1, x_m\}$. An allocation of items among a set of agent n agents is $k = (k_1, \ldots, k_n)$ where $k_i \subseteq X$, $\cup_{i=1}^{n} k_i \subseteq X$ and $k_i \cap k_j = \emptyset$ for $i \neq j$. The generalized Vickrey auction (GVA) is an application of the Pivotal mechanism and works in the following way.

1. Each agent declares a valuation function. So $v_i(y_i)$ is agent i's valuation for allocation k where it is given y_i.

2. The GVA chooses an optimal allocation $k^* = (k_1^*, \ldots, k_n^*)$ that maximizes the sum of all the agents' declared valuations.

3. The GVA announces the winners and their payment p_i:

$$p_i = \sum_{j \neq i} v_j(k'_j) - \sum_{j \neq i} v_j(k^*_j)$$

where $k' = (k'_1, \ldots, k'_{i-1}, k'_{i+1}, \ldots, k'_n)$ 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 $u_i(k_i) = v_i(k_i) - p_i$, the utility of bidder i in the GVA is

$$u_i(k_i^*, p_i) = v_i(k_i^*) - p_i = v_i(k_i^*) + \sum_{j \neq i} v_j(k_j^*) - \sum_{j \neq i} v_j(k'_j).$$

GVA Example: We now provide an example to illustrate how the GVA works. Let there be two agents, agent α and agent β, and let there be two items, g_1 and g_2. Agents can bid on either item or on the bundle $\{g_1, g_2\}$. An agent's bid is represented by a tuple: (a bid for $\{g_1\}$, a bid for $\{g_2\}$, a bid for $\{g_1, g_2\}$ where the bids are XOR'ed together). Suppose the agents bid as follows

- Agent α's bid: (20, 5, 25)
- Agent β's bid: (10, 15, 30)

The GVA allocates g_1 to agent α and g_2 to agent β since this allocation maximizes the sum of the agents' valuations. The amount that each agent pays is computed as follows. If agent α did not bid, then $\{g_1, g_2\}$ would have been allocated to agent β whose valuation for this bundle is 30. When g_1 is allocated to agent α, agent β's valuation is only 15 since it receives g_2. Therefore, agent α's payment is calculated as $30 - 15 = 15$ and its utility is $20 - 15 = 5$. Agent β's payment is $25 - 20 = 5$ and its utility is $15 - 5 = 10$.

Chapter 5

Game Theory for Computationally Limited Agents

In the Chapters 2 and 3 we have presented a model for computationally limited agents and showed how effective deliberation tools for agents can be used for determining how to best allocate their limited computing resources. In the rest of this thesis we study self-interested computationally limited agents in different multiagent settings. In multiagent settings the impact of limited computing resources can be large. In multiagent systems there has been a move from having 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 system), while allowing each agent to choose their own actions. The actions that the agents choose determine the outcome. To guarantee desirable outcomes, the system designer has to engineer the game so as to make sure that each agent is motivated to behave in a desired way. This can be done by using the Nash equilibrium solution concept from game theory (or one of its refinements) [71, 80]. The problem is that the equilibrium for rational agents does not generally remain an equilibrium for computationally limited agents. This leaves a potentially hazardous gap, since naively applied game-theoretic solutions may not provide appropriate incentives to agents, leading to outcomes which may be arbitrarily far from expected.

In this chapter we propose a game theoretic formulation for settings where there are interacting computationally limited agents. We introduce a state of deliberation and provide a formal definition of a deliberation strategy and a deliberation equilibrium. Finally, we coin the term *strategic deliberation* to refer to a new strategic behavior where computationally limited agents invest their own resources towards other agents' valuation problems.

5.1 Computationally Limited Agents

In Chapter 2 we formally defined a computationally limited agent. We summarize the definition in this section. A computationally limited agent, i, is defined by

$$\langle T_i, \text{cost}_i(\cdot), \mathcal{A}_i, \mathcal{PP}_i \rangle$$

where

- T_i is the set of computing resources owned be agent i,
- $\text{cost}_i(\cdot)$ is the cost function of agent i which limits the computing capabilities of the agent,
- $\mathcal{A}_i$ is the set of anytime algorithms that the agent can use, and
- $\mathcal{PP}_i$ is the set of performance profiles (performance profile trees).

The set of anytime algorithms, $\mathcal{A}_i$, contains any algorithms that the agent has. In particular, the agent may have access to, and use, algorithms for its own problems, as well as algorithms used to solve the problems faced by other agents. Each algorithm has its own performance profile. We use the notation $A_i^j \in \mathcal{A}_i$ to denote the anytime algorithm of agent i for problem j. We use the notation $PP_i^j \in \mathcal{PP}_i$ to denote the associated performance profile tree. In particular, we place no restrictions on what problems an agent can compute on.

5.2 The State of Deliberation

As agents allocate computing resources to different problems, their knowledge about the solution quality that is achievable for these problems changes. In order to make good decisions about how to allocate additional computing resources as well as determine which other actions should be taken, an agent must know its current computing results. We store this information in a *state of deliberation*.

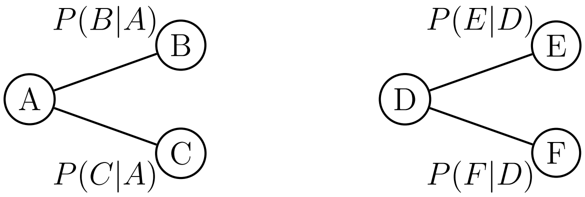


Figure 5.1: *Two performance profile trees.*

Definition 30 (State of deliberation). *Assume that an agent i, defined by $\langle T_i$, $\text{cost}_i(\cdot)$, $\mathcal{A}_i$, $\mathcal{PP}_i\rangle$, has m problems on which it can compute. Assume that agent i has allocated $\mathbf{t} = (t_1, \ldots, t_m)$ computing resources (where t_j is the amount of computing time the agent has spent on problem j). Let $n_j(t_j)$ be the node in performance profile tree PP_i^j that the agent has reached. The* state of deliberation *for agent i at $\mathbf{t}$ is*

$$\phi_i(\mathbf{t}) = \langle n_1(t_1), \ldots, n_m(t_m)\rangle.$$

Performance profile trees capture uncertainty that arises in the deliberation process, and thus are not necessarily branches. Therefore, for any computing resource allocation $\mathbf{t} = (t_1, \ldots, t_m)$, there are multiple possible states of deliberation. We refer to all states of deliberation at $t = \sum_{k=1}^{m} t_k$ as a *deliberation set*.

Definition 31 (Deliberation set). *The* deliberation set *for agent i at t is*

$$\Phi_i(t) = \{\phi_i(\mathbf{t})| \textit{ for } \mathbf{t} = (t_1, \ldots, t_m),\ t = \sum_{k=1}^{m} t_k\}.$$

We provide an example to illustrate the two definitions. Assume that an agent can compute on two problems and that the performance profiles associated with the two problems are shown in Figure 5.1. Assume that the agent has allocated $\mathbf{t} = (1, 1)$ resources. A possible state of deliberation is $\phi((1, 1)) = \langle B, F\rangle$. The deliberation set is

$$\Phi(2) = \{\langle B, E\rangle, \langle B, F\rangle, \langle C, E\rangle, \langle C, F\rangle\}.$$

5.3 Deliberation Strategies and Equilibria

A strategy for an agent is a mapping from its history to an action, for all possible stages in a game. We propose incorporating agents' computing actions into this

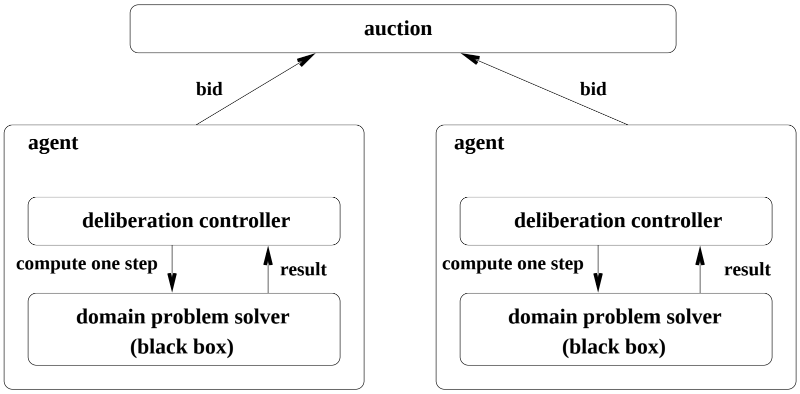


Figure 5.2: *An auction with two computationally limited agents. In order to submit a reasonable bid, each agent needs to first (approximately) compute its valuation for the item that is up for auction.*

strategic setting. That is, an agent's strategy must specify which computing actions it will take, as well as any noncomputing actions. These two aspects are interrelated. For example, consider the auction portrayed in Figure 5.2. Before an agent can submit a bid to the auctioneer, it must compute its value of being allocated the item(s) up for auction. However, what problems it decides to compute on (and how much resources it allocates to the problems) depends on how it is planning on bidding.

Let C_i be the set of computing actions that an agent can take at any time step. That is,

$$C_i = \{\emptyset, c_1, \ldots, c_m\}$$

where c_j is the act of computing one step on problem j, and $\emptyset$ is the action of not computing. The choice of computing actions induces a state of deliberation. Let B_i be the set of other *non-computing* actions. This set of actions is game-specific. For example, in an auction setting B_i is the set of allowable bidding actions, while in a bargaining setting B_i includes all allowable proposals and responses.

Games where the agents are computationally limited are dynamic in nature as the computing actions of the agents take place over time. We use the notation stage_k to denote the k'th stage of the game. When it is clear from the context, we will sometimes use the notation t to represent both stage_t and the number of computing steps that the agents have taken. A history, $H_i(\text{stage}_k)$, for agent i at stage_k is a list of all actions (both computing and non-computing) that the agent has taken, its current state of deliberation, as well as all actions that the agent has observed others taking.

Definition 32 (History). *Let* $c(\text{stage}_k)$ *denote the computing action and* $b(\text{stage}_k)$ *be the non-computing action taken at stage* stage_k. *Let* $\text{obs}(\text{stage}_\text{k})$ *be the set of actions taken by other agents that agent* i *has observed by* stage_k. *A* history *at* stage_k *is*

$$H_i(\text{stage}_k) = \langle (c(\text{stage}_l), b(\text{stage}_l))_{l=0}^{k}, (\text{obs}(\text{stage}_l))_{l=0}^{k}, \phi_i(\mathbf{t}) \rangle$$

where $\mathbf{t} = (t_1, \ldots, t_m)$ *when* t_j *is the number of times the agent has selected computing action* c_j*, and* $\phi_i(\mathbf{t})$ *is the state of deliberation induced by the sequence of computing actions taken by agent* i.

Just as there can be multiple states of deliberation, there can also be multiple possible histories at a stage. We let $\mathcal{H}_i(t) = \{H_i(t)\}$.

It is now possible to define a strategy for a computationally limited agent. We call these strategies *deliberation strategies* since they incorporate the computing actions of the agents.

Definition 33 (Deliberation strategy). *A* deliberation strategy *for agent* i*, is*

$$S_i = (\sigma_i(\text{stage}_k))_{k=0}^{\infty}$$

where

$$\sigma_i(\text{stage}_k) : \mathcal{H}_i(\text{stage}_k) \mapsto C_i \times B_i.$$

That is, given the set of actions the agent has taken, its current results from computing, and any information it has observed concerning the other agents in the game, the strategy specifies what actions the agent should take in the next stage.

Using the deliberation strategy, it is possible to define a solution concept - the *deliberation equilibrium*.

Definition 34 (Deliberation equilibrium). *A (Nash, dominant strategy, perfect Bayesian etc.)* deliberation equilibrium *for computationally limited agents is an equilibrium where the agents' deliberation strategies form a (Nash, dominant strategy, perfect Bayesian etc.) equilibrium.*

	Rational Agent	Computationally Limited Agent
Valuation	v_i	to be determined by computing using $\langle T_i, \text{cost}_i(\cdot), \mathcal{A}_i, \mathcal{PP}_i\rangle$
Strategy	submit a bid b_i	$S_i = (\sigma_i(t))_{t=0}^{D}$ where $\sigma_i(t) : \Phi_i(t-1) \mapsto C_i$ if $t < D$ and $\sigma_i(t) : \Phi_i(t-1) \mapsto C_i \times \mathbb{R}$ if $t = D$
Utility if it wins the auction and pays price p	$v_i - p$	$V(n_i(t_i)) - p - \text{cost}(\mathbf{t})$
Utility if it loses the auction	0	$-\text{cost}_i(\mathbf{t})$

Table 5.1: *The differences between a fully rational agent and a computationally limited agent participating in the same auction.*

For example, for an equilibrium to be a dominant strategy deliberation equilibrium, the strategy of each agent must be the best one for the agent, independent of all other agents. This includes both the deliberation policy used by the agent as well as any noncomputing actions.

We provide an example to illustrate how we incorporate the deliberation actions of an agent into a strategic setting. Assume that a single item is to be allocated via a sealed-bid auction, and assume that a fully rational agent has value v_i for the item. Assume that the value of this item to agent i, defined by $\langle T_i, \text{cost}_i(), \mathcal{A}_i, \mathcal{PP}_i\rangle$, is determined by running algorithm A_i^i, though agent i can also compute on other problems using other algorithms in its set $\mathcal{A}_i$. Finally, assume that the all bids must be submitted by the close of the auction at time D. Table 5.1 compares and contrasts the different aspects of the game for fully rational and computationally limited agents. First, the fully rational agent knows its value for the item in the auction, while the computationally limited agent does not know its value *a priori*, but instead is defined by a set tools that it can use to compute its value. Second, the strategy of a fully rational agent specifies what bid it should submit, while the strategy of the computationally limited agent specifies which problems it will compute on, depending on the results it has currently obtained, as well as how it will bid at the time when the auction closes. The utility of a fully rational agent depends only on whether it was allocated the item or not, and what price it was charged. The utility of a computationally limited agent depends on the allocation, the price and the cost that it incurred while computing.

5.4 Strategic Behavior of Agents

We place no restrictions on what problems an agent is allowed to compute on. In particular, agents are allowed to allocate computing resources in order to determine solutions for their own problems, but are also free to use their limited computing resources in order to evaluate the solution quality of other agents' problems. An agent may do this for many reasons. Agents may find that there are mutual benefits by coordinating their actions, and to discover this they may need to understand each others' goals and problems. In other environments, agents may find themselves at a competitive advantage if they have information about their competitors.

We call the behavior where an agent actively uses some of its computing resources in order to intentionally determine the solution quality of another agent's problem *strategic deliberation*

Definition 35 (Strategic deliberation). *If an agent i uses part of its computing resources to compute on another agent's problems, then agent i is performing* strategic deliberation. *That is, a strategy S_i involves strategic deliberation if for some time step t, there exists history $H \in \mathcal{H}_i(t)$ such that*

$$\sigma_i(t)(H) = (c_j, b)$$

where $b \in B_i$ is a non-computing action, and $c_j \in C_i \setminus \{c_k | k \text{ is a problem of agent } i\}$, the set of computing actions where agent i computes on some problem that is not its own.

Another strategic behavior which might arise among computationally limited agents is something which we call *weak strategic deliberation.*

Definition 36 (Weak Strategic Deliberation). *Assume there are two agents, i and j. Assume that agent j has two possible performance profiles, PP_1 and PP_2. Let $\mathcal{S}_i^{P_1}$ be the set of strategies for agent i if agent j's performance profile is PP_1 and let $\mathcal{S}_i^{PP_2}$ be the set of strategies for agent i if agent j's performance profile is PP_2. Agent i is performing* weak strategic deliberation *if it does not actually use its computing resources to compute on another agent's valuation problems, but does use information* *from the opponents performance profile to devise a strategy. That is, agent i is not performing strong strategic deliberation and*

$$\mathcal{S}_i^{PP_1} \neq \mathcal{S}_i^{PP_2}.$$

In weak strategic deliberation, an agent does not use its computing resources to compute on a competitor's problem. Instead it forms its strategies based on information obtained by examining the competitors' performance profiles. It is then able to deduce information about the deliberation control policies of the competitor.

An alternative way of approaching strategic deliberation is as additional conditioning. Agents are provided with tools (performance profiles) that allow them to condition their computational decisions on the results they obtain from computing on their own problems. Strategic deliberation allows them to further condition their strategies based on results they think their competitors may have obtained. Ideally, neither form of strategic deliberation is present. However, strategic deliberation is the least desirable since agents not only counterspeculate on other agents', but also use their own limited resources in the process, leading to higher costs and less computing time (and therefore, possibly worse solutions) on their own actual problems.

One potential problem with strategically deliberation is that the results obtained by one agent may not be the same as the results obtained by another agent computing on the same problem. This may be due to the use of randomized algorithms, or uncertainty as to what algorithm is used for the problem, or uncertainty as to what problem instance is being worked on. Performance profile trees model such uncertainty through random nodes, which allow an agent to *emulate* possible algorithms and algorithms runs. Emulation is different from actually running an algorithm. When an agent is running an algorithm, if a random node in the performance profile is reached, the uncertainty is resolved in some way (for example, in a randomized algorithm a random number generator would generate some number which would specify the path the algorithm should take). In emulation, agents take an active role. When a random node is reached, the agent *chooses* a "random" path (instead of, for example, using a number generated by a 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 chosen by the agent itself. This means that the agent can emulate different random numbers that a competitor may have used and thus get a better idea of what solutions their competitors may have obtained (and thus also a better idea of how the opponent may have allocated its computation as a function of the results it has obtained).

5.5 Common Knowledge and Independence Assumptions

There are two assumptions which we make in our game theoretic model for computationally-limited agents. First we assume that the performance profiles, algorithms and cost functions are common knowledge, or at a minimum, are drawn from a distribution which is common knowledge. Second, we assume that the algorithms and performance profiles of the agents are independent, in that the results from running one algorithm on one problem does not provide an agent with information about possible results with respect to a different problem or algorithm. These two assumptions, while quite strong, are used to make the analysis of different negotiation mechanisms feasible, and we weaken these assumptions whenever possible. In the rest of this section, we discuss the implications of these assumptions in more detail.

A piece of information is common knowledge if all players know it and all of the players know that all other players know it, *ad infinitum*. Many solution concepts in game theory rely on agents having common knowledge about their situation in the game being played. An exception is the dominant strategy equilibrium solution concept (Chapter 4), where each agent has an optimal strategy independent of what any other agent does. Therefore, agents are not required to have any information about the other agents in the game, eliminating the need for common knowledge. Since we propose placing computationally limited agents into a game theoretic model, we also require the common knowledge assumption for the deliberation equilibrium solution concept. In general, we assume that each agent's performance profiles are common knowledge, and, additionally, that all agents in the game have access and can potentially use the algorithms to determine valuations for different problems. Whenever possible (i.e. when agents' have dominant strategies) we relax this assumption. In our results we clearly state when we are able to do the relaxation.

The second assumption we make is that the performance profiles are independent. In particular, we assume that the results obtained by computing on one problem provide no information about possible obtainable results for a different problem. This means that if an agent wishes to gather information on another agent, it must explicitly study the performance profiles of the second agent, or actively compute on the other agent's problem. It is unable to deduce anything from the results obtained by computing on its own problem(s). For some settings this assumption is very strong. There are situations where agents are faced with similar problems, and so the results of computing on one problem could potentially provide useful information about the possible results obtainable in another problem. However, this assumption allows us to analyze different negotiation mechanisms as it makes the analysis feasible by cleanly defining the results of deliberating on a specific problem. When ever possible, we relax this assumption, and clearly state which results are unaffected and do not depend on the assumption.

5.6 Summary

In this chapter we described how agents' computing actions can be placed into a game theoretic setting. We defined strategies for a computationally limited agent by incorporating the deliberation actions of the agents into the strategy itself. Using this formalism we introduced the deliberation equilibrium which requires that both an agent's deliberation actions as well as other actions are optimal given what strategies other agents are following. Finally, we discussed some new strategic behavior which may arise among computationally limited agents. We introduced

- Strategic deliberation, where agents use some of their computing resources on other agents' problems,
- Weak strategic deliberation, where agents do not actively compute on each others' problems, but do make strategic decisions based on information obtained from competitors' performance profiles, and
- Emulation, where agents use their computing resources to learn about possible outcomes when there is uncertainty as to the path of different algorithms, or even what algorithms are being used by other agents.

We believe that by including the computing actions of agents into the formal definition of a strategy, it is possible to

- understand the strategic impact limited computing resources have on the strategies of agents, and

- lead to the design of robust mechanisms which take into account the fact that the participating agents may not be fully rational.

Chapter 6

One-to-One Negotiation: Bargaining

One of the goals of much work in multiagent systems has been the development of protocols for automated negotiation between agents [56, 96]. In particular, it has been proposed that computational agents would be better at finding and negotiating contracts on behalf of real-world users, since this automation would save human negotiation time, as well as allowing agents to possibly find beneficial deals in combinatorially and strategically complex settings. Competitive bargaining between self-motivated, rational and autonomous agents has been proposed as an appropriate negotiation model as it captures many interesting applications including resource sharing and task allocation [57], as well as many dynamic ecommerce applications.

Bargaining has been well studied by both economists [85] and AI researchers [56, 96]. The term bargaining refers to situations where groups of agents commit themselves voluntarily to a course of action which is beneficial to them all. Negotiation comes into play when there is disagreement as to which course of action is the best. In the economics literature there have been two general approaches to the problem of bargaining. One form, the axiomatic form, proposes a set of desirable properties for a bargaining outcome, and then finds solutions which satisfy as many of the properties as possible [79, 97]. Another approach has been to model bargaining as a dynamic game and then to determine the equilibria of the game (see, for example, [9, 18, 85, 100]) In artificial intelligence, bargaining has been proposed in such diverse settings as a solution for coordinating robots and software agents in time-critical domains [57], to focusing attention on important attributes when there are multiple issues in ecommerce paradigms on which agreement must be reached [34]

In this chapter we focus attention on a setting where there is the possibility of a mutually beneficial agreement where two agents coordinate their actions in order to accomplish some task. However, in order to accomplish the task the agents must compute solutions. The agents also have the possibility of acting independently, and so are faced with the decision of how to negotiate as well as which problems they should compute on. We start this chapter with a motivating example problem where our methods may be needed, and then describe more general settings. In Section 6.3 we present the model of the agents as well as a high level description of the role of bargaining. In Section 6.4 and 6.5 we describe and analyze a single-shot bargaining protocol, while in Section 6.6 we describe and analyze an alternating offers bargaining protocol.

6.1 Example

To make the presentation more concrete, we now discuss an example domain where our methods are needed. Consider a building scenario where two different contractors are each building a different house. In particular, each needs a number of pipes of different, specific lengths for their house. Pipes are sold at some fixed length (for example 2 meters). This scenario is presented in Figure 6.1.

Each agent's *individual problem* is to buy enough large pipes so that it can cut them into the required pieces, while wasting as little as possible. This is a bin-packing problem and is $\mathcal{NP}$-hard.

It is also possible for the two agents to pool their resources and buy enough pipe to cover both of their needs at the same time. By coordinating the cutting of the pipe, the agents may be able to realize savings over acting independently, since it is possible that there will be less wasted pipe due to a better packing. To determine whether and what the savings would be, the agents need to solve another bin-packing problem. This problem is again $\mathcal{NP}$-hard.

Whether the agents actually decide to coordinate their buying is determined by the costs associated with the different solutions. The agents must negotiate, or bargain, about whether to independently buy their own materials, or whether to coordinate in order to reduce costs. They must also negotiate as to how they will split the costs

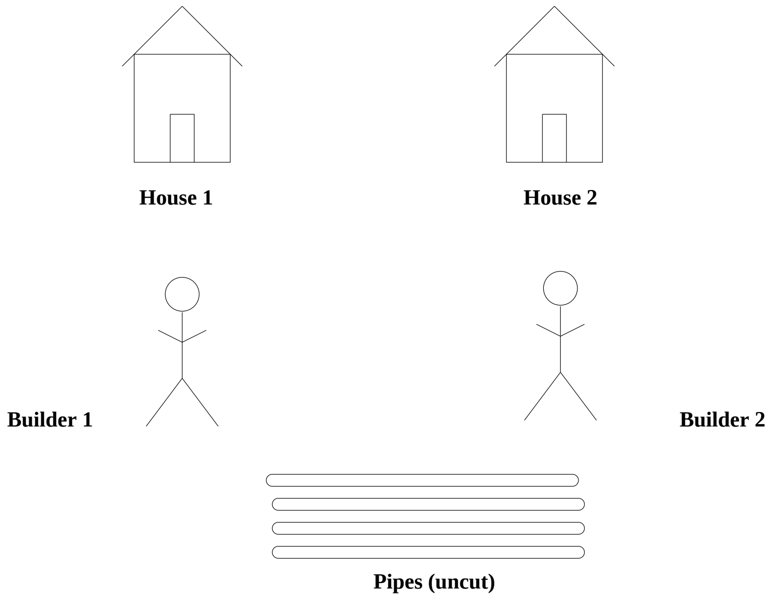


Figure 6.1: *Two builders need to decide how much pipe they should buy. They can either buy the pipe independently, or buy it together, leading to possible savings.*

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 packings) for the three problems.

6.2 The General Setting

In general the methods in this chapter are needed in any setting with two self-interested agents where each agent has an intractable *individual problem* and there is a potential savings from pooling the problems, giving rise to an intractable *joint problem.* We also assume that the value of any solution to an agent's individual problem is not affected by what solution the other agent uses for its individual problem.

Applications with these characteristics are ubiquitous, including transportation as discussed above, manufacturing (where two companies that potentially subcontract with each other need to construct their manufacturing plans and schedules), electric power negotiation between a custom provider and an industrial consumer (where the participants need to construct their their production and consumption schedules), classroom scheduling, scheduling of scientific equipment among multiple users, and bandwidth allocation and routing in multi-provider multi-consumer computer networks to name a few.

In order to determine the gain generated by pooling instead of each agent operating individually, agents need to compute solutions to both agents' individual problems as well as to the joint problem.

By computing on the joint problem, an agent reduces the amount of resources it has for computing on its individual problem. This may increase the joint value to the agents (reduce the agents' costs), but makes this agent's fall back position worse when it comes to bargaining how the joint value should be divided between the two agents. Also, if one agent is computing on the joint problem, would it not be better for the other agent to compute on something different so as not to waste computation? In this chapter we present models where each agent strategically decides on how to use its limited deliberation resources in order to maximize its own expected payoff in such settings.

6.3 The Model

6.3.1 The Agents

We assume that two agents, α and β, are computationally limited, and as described in Chapter 2, are defined by their cost functions, sets of anytime algorithms, and performance profile trees. The agents do not know their values for different outcomes. For example, neither agent has a solution in hand for what to do if no agreement is reached. Instead, each agent must compute or gather information in order to be able to fully participate in the bargaining process. Each agent is allowed, and has the appropriate algorithms and performance profiles, to compute on the problem of agent α, the problem of agent β, or the joint problem.

We let $\mathcal{T}_i^j$ denote the performance profile tree that agent i has for problem j. As described in Chapter 5, we assume that performance profiles and algorithms are common knowledge, accessible to all agents, and are independent, unless otherwise noted. We use the notation $n_i^j(t_j)$ to denote the node in the performance profile tree $\mathcal{T}_i^j$ that agent i has reached after allocating t_j steps to problem j. We let $V(n_i^j(t_j))$ denote the value or solution quality associated the node n_i^j. We will sometimes use the abbreviation v_i^j to also represent the value, if the actual node is not important to the discussion.

While in theory we put no restrictions on the cost functions of the agents, there is a natural restriction which arises from the bargaining interactions. At some point in time, an agreement must be reached. This agreement may be to either cooperate on the joint problem, or for each agent to act independently. We call this agreement time the *deadline* D and model the cost function of agent i as as

$$\text{cost}_i((t_\alpha, t_\beta, t_{\text{joint}})) = \begin{cases} \infty & \text{if } t_\alpha + t_\beta + t_{\text{joint}} > D \\ 0 & \text{otherwise} \end{cases}$$

where t_j is the number of computing steps that agent i has allocated to problem $j \in \{\alpha, \beta, \text{joint}\}$.

The values that have been computed by the agents affect the bargaining process and outcomes. For example, if both agents decide to not compute on the joint problem, then it is unlikely that any agreement will be reached in the bargaining process on whether to execute the joint solution. Instead, both agents would likely act independently, implementing their own individual solutions. If, on the other extreme, both agents compute only on the joint problem, then it is more likely that agreement will be reached. The bargaining strategies of the agents are determined by the solutions that have obtained for all problems. The offer that an agent makes is determined by the value of the joint solution that it has obtained as well as the solution it has obtained for its individual solution. Similarly, the offer that an agent will accept is determined by the value that it has obtained for its individual problem, since that is its fallback value (that is, the agent is guaranteed to receive at least that amount of the agreement is not reached).

6.3.2 Bargaining

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

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

Bargaining models for agents have been well studied by both economists (see, for example [85, 100]) and AI researchers [57, 111]. In our setting, two agents, α and β, bargain over how to divide a surplus (or cost) associated with implementing some joint solution. Call the value of the solution computed by the deadline by agent $i \in \{\alpha, \beta\}$ to agent α's problem v_i^α, to agent β's problem v_i^β, and to the joint problem v_i^{joint}. Through bargaining, the agents decide whether to pool or not, and in the former case they also decide how to divide the value of the solution to the joint problem. If the value of the solution to the joint problem is higher than the sum of the values of the solutions to the individual problems, then there is a potential gain from agreeing to implement the joint solution.

We study two different negotiation settings; a single-shot bargaining game and an alternating offers bargaining game. We start with the single-shot bargaining game.

6.4 Single-shot Bargaining Game

The first negotiation model we study is *single-shot bargaining*, also know as an *ultimatum game*. We restrict the bargaining protocol so that only one agent is allowed to make an offer, while the other agent has the ability to either accept or reject the offer made (that is, the agents are involved in an ultimatum game). If a proposal is accepted, the joint solution is implemented and the surplus is divided as determined by the agreed upon proposal. If no agreement is reached, then the agents implement their individual solutions with no further interaction. We assume that if there is an agreement, it must occur at or before a specified deadline D.

Say that agent α is the proposer. It makes a take-it-or-leave-it offer, x_α^o, to the other agent, β, about how much agent β's payoff will be if they pool.[1] Agent β can then accept or reject the offer. If agent β accepts the offer, the agents pool and use agent α's solution to the joint problem. Agent β's payoff is x_α^o as proposed and agent α gets the rest of the value of the solution: $v_\alpha^{\text{joint}} - x_\alpha^o$. If agent β rejects, both agents implement their own computed solutions to their own individual problems, in which case agent α's payoff is v_α^α and agent β's payoff is v_β^β. The payoffs are presented in Table 6.1.

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

[1] We allow an agent to make a negative "unacceptable" offer which signals that it does not want to coordinate or implement the joint solution.

Agent	Payoff if the offer is accepted	Payoff if the offer is rejected
α	$v_\alpha^{\text{joint}} - x_\alpha^o$	v_α^α
β	x_α^o	v_β^β

Table 6.1: *Each agent i has computed a value v_i^j for problem j. If agent α makes an offer, x_α^o, then agent β has the choice of either accepting or rejecting it. The payoffs for the agents in either situation are listed in the table below.*

α 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 α is $OA_\alpha = (x_\alpha^o, x_\alpha^a) \in \mathbb{R}^2$, where x_α^o is the amount that agent α would offer if it were the proposer, and x_α^a is the minimum value it would accept if agent β made the proposal. The offer-accept vector for agent β is defined similarly.

6.4.1 Strategies for the Single-shot Game

The strategies of the agents in this ultimatum game incorporate both computing and bargaining actions.

Definition 37 (Strategy for the ultimatum game). *Let c^j denote the action of taking one computing step on problem j, let $\emptyset$ denote the act of not computing, and let $\Phi_i(t)$ denote the set of all states of deliberation for agent i at time t. A strategy for agent i in the ultimatum game with deadline D is*

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

where

$$\sigma_i^t : \Phi_i(t) \mapsto \{c^\alpha, c^\beta, c^{\text{joint}}, \emptyset\} \quad \textit{for } t < D$$

and

$$\sigma_i^t : \Phi_i(D) \mapsto \mathbb{R}^2.$$

If it is before the deadline, the strategy specifies which computing action the agent should take, given its current state of deliberation. If it is at the bargaining deadline, the strategy specifies which offer-accept vector an agent should declare.

Our analysis will also allow *mixed strategies.* A mixed strategy for agent i, $\tilde{S}_i$ is a mapping from a state of deliberation to a probability distribution over computing actions if t is before the deadline, otherwise it is a mapping from a state of deliberation to a probability distribution over offer-accept vectors.

6.5 Analysis of Single-shot Bargaining

There are many parameters in the single-shot bargaining mechanism which influence what strategies agents will choose to follow in equilibrium. These parameters include

- which agent is the proposer,
- whether or not the proposer is known in advance of the deadline,
- whether the deadlines are common knowledge, and
- how much uncertainty arises while computing.

In this section we study the effects of these parameters on the equilibria which arise in the bargaining game. In addition, we investigate properties of the equilibria, such as the existence of pure strategy equilibria, and whether equilibria are also Pareto optimal.

6.5.1 Known Proposer

For an agent that is never going to make an offer, we can prescribe a dominant strategy independent of the statistical performance profiles:

Theorem 3. *If an agent, β, knows that it cannot make a proposal at the deadline D, then it has a dominant strategy of computing only on its own problem and accepting any offer x^o_α such that $x^o_\alpha \geq V(n)$ where n is the node in the performance profile $\mathcal{T}^\beta$ that agent β has reached at time D. If the performance profile does not flatten before the deadline ($V(n') < V(n)$ for every node n' on the path to n), then this is the unique dominant strategy. This result does not depend on a common knowledge assumption.*

Proof. In the event that an agent is not reached, agent β could not have achieved higher payoff than by computing on its individual problem (even if it knows that further computation will not improve its solution). In the event that an agreement is reached, agent β would have been best off by computing so as to maximize the minimal offer it will accept, $V(n^{\beta}_{\beta})$. Since solution quality is nondecreasing in computation time, if agent β deviates and computes t steps on a different problem, then the value of its fallback is $V(n'^{\beta}_{\beta}) \leq V(n^{\beta}_{\beta})$ where $\text{time}(n^{\beta}_{\beta}) = \text{time}(n'^{\beta}_{\beta}) + t$. If $V(n') < V(n)$ for every node n' on the path to n, then this inequality is strict. □

Corollary 1. *In the games where the proposer is known, there exists a pure strategy PBE.*

Proof. By Theorem 3, the receiver of the offer has a dominant strategy. Say the proposer were to use a mixed strategy. In general, every pure strategy that has nonzero probability in a best-response mixed strategy has equal expected payoff [71]. Since mixing by the proposer will not affect the receiver's strategy, the proposer might as well use one of the pure strategies in its mix. □

The equilibrium differs based on whether or not the deadline is known, as discussed in the next subsections.

Known Proposers, Known Deadline

In the simplest setting, both the deadline and proposer are common knowledge. Without loss of generality we assume that agent α is the proposer and the deadline is at time T. Therefore, from Theorem 3, agent β has a dominant strategy, S_{β}, which is to compute on the solution for it's own problem and accept any offer that is greater than (or equal to) the value of its computed solution. Knowing this, agent α can determine a strategy which is a best–response.

Assume that by computing according to a certain strategy S_{α}, the proposing agent α reaches deliberation state $\phi_{\alpha}(D) = \langle n^{\alpha}_{\alpha}, n^{\beta}_{\alpha}, n^{\text{joint}}_{\alpha} \rangle$ at time D. It is possible to compute agent α's expected utility, u_{α}, from offering some amount x^{o}_{α} to agent β in this situation. The expected utility is

$$E[u_{\alpha}(x^{o}_{\alpha}|\phi_{\alpha}(D), S_{\beta})] = P_a(x^{o}_{\alpha}|n^{\beta}_{\alpha})[V(n^{\text{joint}}_{\alpha}) - x^{o}_{\alpha}] + (1 - P_a(x^{o}_{\alpha}|n^{\beta}_{\alpha}))V(n^{\alpha}_{\alpha}) \quad (6.1)$$

where $P_a(x^{o}_{\alpha}|n^{\beta}_{\alpha})$ is the probability that agent β will accept an offer x^{o}_{α}, conditioned on agent α reaching node n^{β}_{α}. When it is clear from the context, we will use $P_a(x^{o}_{\alpha})$ to represent $P_a(x^{o}_{\alpha}|n^{\beta}_{\alpha})$.

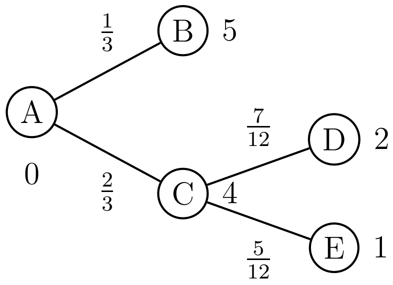


Figure 6.2: *A performance profile tree for agent β. The probability that agent β would accept an offer $x^o_\alpha = 2$ is equal to $\frac{13}{18}$*

These probabilities are determined by agent α's beliefs about what value agent β has computed for its own individual problem. In a setting where agent β has a dominant strategy (that is, it computes only on the solution for its own problem), agent α can compute its beliefs that agent β will accept an offer of x with probability $P_a(x)$, simply by noting the values of the nodes that can be reached by time D in the performance profile for agent β's individual problem, and computing the probability of reaching each node. Since we use performance profile trees, this is easily done, since the probability of reaching some node $n(t)$ in tree $\mathcal{T}$ at depth t is simply the product of the probabilities on the edges in the path to node $n(t)$. If we let $P(n(t))$ denote the probability of reaching node $n(t)$, then

$$P_a(x^o_\alpha) \equiv P_a(x^o_\alpha | n^\beta_\alpha) = \prod_{n(D)\in\{\mathcal{T}^\beta_\beta(n^\beta_\alpha)|V(n(D))\geq x^o_\alpha\}} P(n(D))$$

Figure 6.2 provides an example to illustrate this.

We can determine the proposer's expected utility from following a particular strategy as follows. Assume agent α is executing strategy $S_\alpha = (\sigma^t_i)^D_{t=0}$. At time D, when the agent must make a proposal, it is in some deliberation state $\phi_\alpha(D) = \langle n^\alpha_\alpha, n^\beta_\alpha, n^{\text{joint}}_\alpha \rangle$ where $\text{time}(n^\alpha_\alpha) = t^\alpha_\alpha$, $\text{time}(n^\beta_\alpha) = t^\beta_\alpha$, and $\text{time}(n^{\text{joint}}_\alpha) = t^{\text{joint}}_\alpha$.

If σ^D_i dictates that in deliberation state $\phi_\alpha(D)$ it makes an offer of x^o_α, then agent α's expected utility from following S_α is

$$E[U_\alpha(S_\alpha, S_\beta)] = \sum_{\phi_\alpha(D)\in\Phi_\alpha(t^\alpha_\alpha, t^\beta_\alpha, t^{\text{joint}}_\alpha)} p(\phi_\alpha(D))(P_a(x^o_\alpha)[V(n^{\text{joint}}_\alpha) - x^o_\alpha] + (1 - P_a(x^o_\alpha))V(n^\alpha_\alpha)) \tag{6.2}$$

where $p(\phi_\alpha(D))$ is the probability of being in deliberation state $\phi_\alpha(D)$.

The game differs based on whether the performance profiles of the proposing agent have a branching factor greater than one (stochastic performance profiles), or whether they are simply branches (deterministic performance profiles).

Deterministic Performance Profiles

In an environment where the performance profiles are deterministic, the equilibria can be analytically determined.[2]

Theorem 4. *Assume that the agents' performance profiles $\mathcal{T}_\alpha^\alpha$, $\mathcal{T}_\alpha^\beta$, $\mathcal{T}_\alpha^{\text{joint}}$,$\mathcal{T}_\beta^\alpha$, $\mathcal{T}_\beta^\beta$, and $\mathcal{T}_\beta^{\text{joint}}$ are deterministic and agent α knows agent β's performance profiles. Then there exists a PBE where agent β will only compute on its own problem, and agent α will never split its computation. It will either compute solely on its own problem or solely on the joint problem. The PBE payoffs to the agents are unique, and the PBE is unique unless the performance profile that an agent is computing on flattens, after which time it does not matter where the agent computes since that does not change its payoff or bargaining strategy. The PBEs are also the only Nash equilibria.*

Proof. Let n_α^{joint} be the node in $\mathcal{T}_\alpha^{\text{joint}}$ that agent α reaches after allocating all of its computation on the joint problem. Let n_α^α be the node in $\mathcal{T}_\alpha^\alpha$ that agent α reaches after allocating all of its computation on its own problem. Let n_β^β be the node in $\mathcal{T}_\beta^\beta$ that agent β reaches after allocating all of its computation on its own problem.

By Theorem 3, agent β has a dominant strategy to compute on its own solution (unless its performance profile flattens after which time it does not matter where the agent computes since that does not change its payoff). Agent α's strategies are more complex since they depend on agent β's final fallback value, $V(n_\beta^\beta)$, and also on what potential values the joint solution and α's individual solution may have.

1. **Case 1:** $V(n_\alpha^{\text{joint}}) - V(n_\beta^\beta) > V(n_\alpha^\alpha)$. Agent β will accept any offer greater than or equal to $V(n_\beta^\beta)$ since that is its fallback. If agent α makes an offer that is acceptable to agent β, then the highest payoff that agent α can receive is $V(n_\alpha^{\text{joint}}) - V(n_\beta^\beta)$. If this value is greater than $V(n_\alpha^\alpha)$ –that is, the highest fallback value agent α can have—then agent α will make an acceptable offer. To maximize the amount it will get from making the offer, agent α must compute

[2] In Chapter 2 we defined deterministic performance profiles. While there is no uncertainty as to what value will result from computing, an agent still needs to compute in order to obtain the value.

only on the joint problem. Any deviation from this strategy will result in agent α receiving a lesser payoff (and strictly less if its performance profile has not flattened).

2. **Case 2:** $V(n_\alpha^{\text{joint}}) - V(n_\beta^\beta) < V(n_\alpha^\alpha)$. Any acceptable offer that agent α makes results in agent α receiving a lesser payoff than if it had computed on its own solution solely, and made an unacceptable offer (and strictly less if its performance profile has not flattened). Therefore agent α will compute only on its own problem until that performance profile flattens, after which it does not matter where it allocates the rest of its computation.

3. **Case 3:** $V(n_\alpha^{\text{joint}}) - V(n_\beta^\beta) = V(n_\alpha^\alpha)$. By computing only on its own problem, agent α's payoff is $V(n_\alpha^\alpha)$. By computing only on the joint problem, the payoff is $V(n_\alpha^{\text{joint}}) - V(n_\beta^\beta)$. These payoffs are equal. However, by dividing the computation across the problems, both payoffs decrease (unless at least one of the two performance profiles has flattened, after which it does not matter where the agent allocates the rest of its computation).

The above arguments also hold for Nash equilibrium. □

Stochastic Performance Profiles

If the performance profiles are shared but stochastic, determining the equilibrium is more difficult. By Theorem 3, agent β has a dominant strategy, S_β, and only computes on its individual problem. If that performance profile has flattened and agent β has computed on agent α's or the joint problem thereafter, this does not change agent β's fallback, and this is the only aspect of agent β that agent α cares about.

However, based on the results it has obtained so far, agent α may decide to switch between problems on which it is computing—possibly several times. The problem is similar to computing values and policies for a sequential decision problem with stochastic actions, except that the deadlines mean that the game has a finite horizon. The payoffs can be seen as state–dependent reward values and the accessibility functions can be modeled as the probability of transferring into a deliberation state, given the action taken.

There are two different cases that affect agent α's capabilities when it comes to speculating as to what value agent β has obtained from deliberation. If the two agents share algorithms and, therefore, performance profiles, then agent α can deliberate on agent β's problem, and be sure that the results obtained are related to those that agent β has obtained. Agent α might then find it useful to deliberate on agent β's problem in order to refine its beliefs as to what value agent β has obtained. On the other hand, if the agents have different algorithms and, therefore, different performance profiles, any deliberation that agent α does on agent β's problem, using its own algorithm, may not correctly reflect the solutions that agent β has achieved. It gets no utility from computing on agent β's problem since its beliefs can not be updated.

In this section we do not assume that the performance profiles are common knowledge. However, it is required that at least agent α can observe the performance profiles for agent β. Agent β does not need to know that agent α can view its performance profiles. This knowledge does not change agent β's behavior as it has a dominant strategy.

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

$$P_a(x) = \sum_{n^\beta} P(n^\beta) \tag{6.3}$$

where $n^\beta \in \{n | n \text{ is in subtree } \mathcal{T}^\beta(n^\beta_\alpha) \text{ at depth } D - \text{time}(n^\beta_\alpha) \text{ and } V(n) \leq x\}$.[3]

The offer, x^o_α, that maximizes the expected utility of agent α, given that α is in the state of deliberation $\phi_\alpha(D) = \langle n_\alpha,^\alpha, n^\beta_\alpha, n^{\text{joint}}_\alpha \rangle$ is

$$x^o_\alpha(\phi_\alpha(D)) = \arg\max_x \left[P_a(x)(V(n^{\text{joint}}_\alpha - x)_(1 - P_a(x))V(n^\alpha_\alpha)\right]. \tag{6.4}$$

We denote the optimal bargaining action, given the deliberation state $\phi_\alpha(D)$ by

$$B^*_\alpha(\phi_\alpha(D)) = (x^o_\alpha(\phi_\alpha(D)), V(n^\alpha_\alpha)). \tag{6.5}$$

The expected utility to agent α from following a strategy S_α which results in deliberation state $\phi_\alpha(D)$ and where agent α offers $B^*_\alpha(\phi_\alpha(D))$ to agent β who has

[3]Since the agents have performance profile trees, it is straightforward to compute $P(n^\beta)$. It is simply the product of the probabilities on the edges of the path to node n^β, starting at node n^β_α. That is, the performance profile tree representation stores the conditional probabilities.

executed its dominant strategy S^*_β is

$$E[U_\alpha(S_\alpha, S^*_\beta)] = P_\alpha(x)(V^{\text{joint}}_\alpha) - x) + (1 - P_\alpha(x))V(n^\alpha_\alpha) \tag{6.6}$$

where $x = x^o_\alpha(\phi_\alpha(D))$.

It is possible to work backwards and to determine which computing action c^z is optimal if agent α finds itself in deliberation state $\phi_\alpha(t)$ at time t once the optimal offers have been determined for each final deliberation state. Let

$$U^C_\alpha((c^z, \phi_\alpha(t)), S_\beta)$$

denote the utility of agent α of computing on problem z at time $t+1$, given that at time t it is in deliberation state $\phi_\alpha(t)$ and it will make the optimal offer at time D given that agent β is following strategy S^*_β. Then

$$E[U^C_\alpha((c^z, \phi_\alpha(D)), S_\beta)] = P_\alpha(x)(V^{\text{joint}}_\alpha) - x) + (1 - P_\alpha(x))V(n^\alpha_\alpha) \tag{6.7}$$

where $x = x^o_\alpha(\phi_\alpha(D))$, and

$$E[U^C_\alpha((c^z, \phi_\alpha(t)), S_\beta)] = \sum_{\phi_\alpha(t+1) \in \Phi_\alpha(t+1)} P(\phi_\alpha(t+1)|\phi_\alpha(t), c^z) \max_{c^y} E[U^C_\alpha((a^y, \phi_\alpha(t+1)), S_\beta)]$$

where $P(\phi_\alpha(t+1)|\phi_\alpha(t), c^z)$ is the probability of reaching deliberation state $\phi_\alpha(t+1)$ given that in deliberation state $\phi_\alpha(t)$ the agent took computing action c^z. This probability is read directly from the performance profiles.

The optimal computing action in deliberation state $\phi_\alpha(t)$ is simply

$$c^{z*}(\phi_\alpha(t)) = \arg\max_{c^z} E[U^C_\alpha((c^z, \phi_\alpha(t)), S_\beta)].$$

The sequence of actions $(c^{z*}(\phi_\alpha(t)))^D_{t=0}$ coupled with the offer-accept vectors $(x^o_\alpha(\phi_\alpha(D)), V(n^\alpha_\alpha))$ for each deliberation state $\phi_\alpha(D)$, define a best-response strategy to S^*_β. Algorithm 1 computes the best-response strategy for agent α.

Theorem 5. *Algorithm 1 correctly computes a PBE strategy for agent α.*[4] *Assume that the number of children of any node in $\mathcal{T}^\alpha_\alpha$, $\mathcal{T}^\beta_\alpha$ and $\mathcal{T}^{\text{joint}}_\alpha$ is at most k. Algorithm 1 runs in $O(k^{D-1}D^3)$ time.*

[4]By keeping track of equally good actions at every step, Algorithms 1, 2, and 3 can return all PBE strategies for agent α. Again, the dominant strategy of agent β is to compute on its own problem (unless the performance profile flattens out after which it does not matter what agent β computes on.

Algorithm 1 Known Proposer, Known Deadline

input: Performance profile trees, D
for each deliberation state $\phi_\alpha(D) \in \Phi_\alpha(D)$ **do**
  $x^o_\alpha(\phi_\alpha(D)) \leftarrow \arg\max_x \left[P_a(x)(V(n^{\text{joint}}_\alpha - x) + (1 - P_a(x)V(n^\alpha_\alpha)\right]$
end for
for each t from $D-1$ down to 0 **do**
  for each deliberation state $\phi_\alpha(t) \in \Phi_\alpha(t)$ **do**
    $c^{z*}(\theta_\alpha(t)) \leftarrow \arg\max_{c^z} E[U^C_\alpha((c^z, \phi_\alpha(t)), S_\beta)]$
  end for
end for
return $(c^{z*}(\phi_\alpha(t)))^D_{t=0}$ and $(x^\alpha_o(\phi_\alpha(D)), V(n^\alpha_\alpha))_{\phi_\alpha(D)\in\Phi_\alpha(D)}$

Proof. The nonproposing agent, β, has a dominant strategy, S_β. Algorithm 1 computes the best–response for agent α, at every time and for every state of deliberation.

Let k be the maximum number of children of any node in the performance profiles. At time t the number of states of deliberation is at most $k^t\binom{t+2}{2} = \frac{k^t(t+2)(t+1)}{2}$. Since

$$\begin{aligned}\sum_{t=0}^{D-1} k^t \frac{(t+1)(t+2)}{2} &\leq k^{D-1}\sum_{t=0}^{D-1}\frac{(t+1)(t+2)}{2} \\ &= k^{D-1}\frac{D(D+1)(D+2)}{6}\end{aligned}$$

the algorithm runs in $O(k^{D-1}D^3)$ time. □

Known Proposer, Unknown Deadline

There are situations where agents may not know the deadline. We represent this by a probability distribution $Q = \{q(i)\}^D_{i=1}$ over possible deadlines where D is some maximum deadline. Q is assumed to be common knowledge.

Whenever time t is reached but the deadline does not arrive, agents update their beliefs about Q. The new distribution is $Q' = \{q'(i)\}^D_{i=t}$ where

$$q'(t) = \frac{q(t)}{\sum_{j=t}^D q(j)}.$$

Algorithm 2 Deterministic PPTrees, Known Proposer, Unknown Deadline

input: Performance profile trees, Q
for each deliberation state $\Gamma_\alpha(D)$ **do**
  $x^o_\alpha(\Gamma_\alpha(D)) \leftarrow \arg\max_x \left[P_\alpha(x)(V(n^{\text{joint}}_\alpha) - x_+(1 - P_\alpha(x))V(n^\alpha_\alpha)\right]$
end for
for each t from $D - 1$ down to 0 **do**
  $q'(t) \leftarrow \frac{q(t)}{\sum_{j=t}^{D} q(j)}$
  for each deliberation state $\Gamma_\alpha(t)$ **do**
    $x^o_\alpha(\Gamma_\alpha(t)) \leftarrow \arg\max_x \left[P_\alpha(x)(V(n^{\text{joint}}_\alpha) - x_+(1 - P_\alpha(x))V(n^\alpha_\alpha)\right]$
    $c^{z*}(\Gamma_\alpha(t)) \leftarrow \arg\max_{c^z} \left[(1 - q'(t))E[U^C_\alpha((c^z, \Gamma_\alpha(t)), S_\beta)\right]$
  end for
end for
return $(c^{z*}(\Gamma_\alpha(t)))^D_{t=0}$ and $(x^o_\alpha(\Gamma_\alpha(t)), V(n^\alpha_\alpha))^D_{t=0}$

Deterministic Performance Profiles

Since there is no uncertainty as to agent β's fallback value, agent α need never compute on agent β's problem. Therefore, agent α will only be in deliberation states $\langle n^\alpha_\alpha, n^\beta_\alpha, n^{\text{joint}}_\alpha\rangle$ where $\text{time}(n^\beta_\alpha) = 0$. Therefore, strategies that include computation actions a^β need not be considered. This, and the lack of uncertainty in which deliberation state action a leads to, greatly reduce the space of deliberation states to consider. Denote by $\Gamma_\alpha(t)$ any deliberation state of agent α where $\text{time}(n^\alpha_\alpha) + \text{time}(n^{\text{joint}}_\alpha) = t$ and $\text{time}(n^\beta_\alpha) = 0$. The algorithm (Algorithm 2) for determining agent α's equilibrium strategy differs from Algorithm 1 in that it incorporates the probability that the deadline may arrive at any time, and considers only the restricted space of deliberation states.

Theorem 6. *With deterministic performance profiles, Algorithm 2 correctly computes a PBE strategy for agent α in $O(D^2)$ time.*

Proof. The nonproposing agent, β, has a dominant strategy, S_β. Algorithm 2 computes the best–response for agent α, at every time and for every state of deliberation.

Since the setting is deterministic, agent α need only consider deliberation states at time t of the form $\theta_\alpha(t) = \langle n^\alpha_\alpha, n^\beta_\alpha, n^{\text{joint}}_\alpha\rangle$ where $\text{time}(n^\beta_\alpha) = 0$. At time t there are $t+1$ deliberation states of this form. Each calculation in the algorithm takes constant time. The first loop is repeated $D+1$ times. In the second loop, the calculations are

Algorithm 3 Stochastic PPTrees, Known Proposer, Unknown Deadline

input: Performance profile trees, Q
for each deliberation state $\phi_\alpha(D) \in \Phi_\alpha(D)$ **do**
  $x^o_\alpha(\phi_\alpha(D)) \leftarrow \arg\max_x \left[P_a(x)(V(n^{\text{joint}}_\alpha) - x) + (1 - P_a(x))V(n^\alpha_\alpha)\right]$
end for
for each t from $D-1$ down to 0 **do**
  $q'(t) \leftarrow \frac{q(t)}{\sum_{j=t}^{D} q(j)}$
  for each deliberation state $\phi_\alpha(t) \in \Phi_\alpha(t)$ **do**
    $x^o_\alpha(\phi_\alpha(t)) \leftarrow \arg\max_x \left[P_a(x)(V(n^{\text{joint}}_\alpha) - x) + (1 - P_a(x))V(n^\alpha_\alpha\right]$
    $c^{z*}(\theta_\alpha(t)) \leftarrow \arg\max_{(c^z}(1 - q'(t))E[U^C_\alpha((c^z, \phi_\alpha(t)), S_\beta)]$
  end for
end for
return $(c^{z*}(\phi_\alpha(t)))_{t=0}^{D}$ and $(x^\alpha_o(\phi_\alpha(D)), V(n^\alpha_\alpha))_{\phi_\alpha(D) \in \Phi_\alpha(D)}$

done $t+1$ times for $t = 0$ to $D-1$, or $D(D+1)/2$ times. Therefore, the algorithm takes $O(D^2)$ time. □

Stochastic Performance Profiles

The algorithm differs from Algorithm 1 in that it considers the probability that the deadline might arrive at any time. Any deliberation state is possible in equilibrium.

Theorem 7. *Algorithm 3 correctly computes a PBE strategy for agent α. Assume that the number of children of any node in $\mathcal{T}^\alpha_\alpha$, $\mathcal{T}^\beta_\alpha$ and $\mathcal{T}^{\text{joint}}_\alpha$ is at most k. Algorithm 3 runs in $O(k^{D-1}D^3)$ time.*

Proof. The nonproposing agent, β, has a dominant strategy, S_β. Algorithm 3 computes the best–response for agent α, at every time and for every state of deliberation. Let k be the maximum branching factor for the performance profiles. At time t the number of states of deliberation is $k^t\binom{t+2}{2} = \frac{k^t(t+2)(t+1)}{2}$. Since

$$\sum_{t=0}^{D-1} k^t \frac{(t+1)(t+2)}{2} \leq k^{D-1} \sum_{t=0}^{D-1} \frac{(t+1)(t+2)}{2} = k^{D-1} \frac{D(D+1)(D+2)}{6}$$

the algorithm runs in $O(k^{D-1}D^3)$ time. □

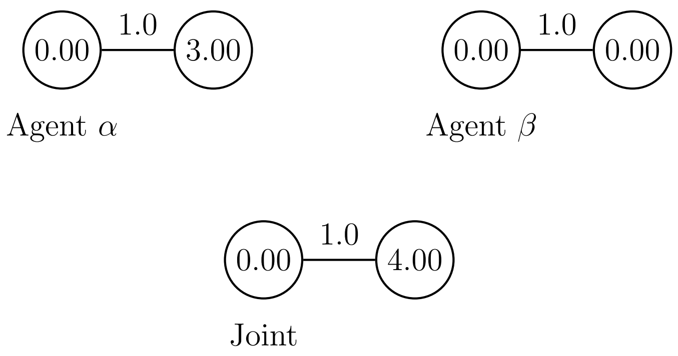


Figure 6.3: *Performance profile trees for the example where there is no pure strategy Nash equilibrium.*

6.5.2 Unknown Proposer

This section discusses the case where the proposer is unknown but the probability of each agent being the proposer is common knowledge. The deadline may be common knowledge. Alternatively, the deadline is not known but its distribution is common knowledge. This is a more complex setting since neither agent may have a dominant strategy.

Nonexistence of a Pure Strategy Equilibrium

In this subsection we show that in some cases, there is no pure strategy equilibrium.

Theorem 8. *There exist instances (defined by the performance profile trees) of the game that have a unique mixed strategy PBE, but no pure strategy PBE (not even a pure strategy Nash equilibrium) This result does not reply on an assumption about independence of performance profiles..*

Proof. Assume that the agents have the performance profiles in Figure 6.3 and are allowed to take only one deliberation action ($D = 1$). Assume, also, that with equal probability either agent may be named as the proposer, that is, agent α is the proposer with probability $\frac{1}{2}$. Let $\emptyset$ represent a null offer, where the proposer does not want to implement a joint solution.

The undominated strategies for agent α are

- $S_\alpha^1 = \{a^\alpha, (\emptyset, 3.0)\}$: Agent α computes one time step on its own problem. If it is chosen as the proposer then it makes a null offer, otherwise it accepts any offer that is greater than or equal to the fallback value of 3.0.

- $S_\alpha^2 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent α computes one time step on the joint problem. If it is chosen as the proposer then it makes an offer of 0.0, otherwise it accepts any offer greater than or equal to the fallback value of 0.0.

The undominated strategies for agent β are

- $S_\beta^1 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent β computes on the joint problem. If it is chosen proposer then it offers 0.0, otherwise it accepts anything greater than or equal to the fallback value of 0.0.
- $S_\beta^2 = \{a^{\text{joint}}, (3.0, 0.0)\}$: Agent β computes on the joint problem. If it is chosen as the proposer, then it offers 3.0, otherwise it accepts anything greater than or equal to the fallback value of 0.0.

The game can be represented in normal form (Table 6.2). There is no pure strategy

	S_β^1	S_β^2
S_α^1	3.0, 0.0	3.0, 0.5
S_α^2	2.0, 2.0	3.5, 0.5

Table 6.2: *Reduced normal form of the bargaining game with performance profiles in Figure 6.3. There is no pure strategy Nash equilibrium.*

Nash equilibrium for this game. It is easy to prove this by checking each strategy profile.

1. (S_α^1, S_β^1) is not a Nash equilibrium since agent β would respond with S_β^2 if α played S_α^1.
2. (S_α^1, S_β^2) is not a Nash equilibrium since agent α would respond with S_α^2 is agent β played S_β^2.
3. (S_α^2, S_β^2) is not a Nash equilibrium since agent β would respond with S_β^1 if agent α played S_α^2.
4. (S_α^2, S_β^1) is not a Nash equilibrium since agent α would respond with S_α^1 if agent β played S_β^1.

There does exist a mixed strategy Nash equilibrium. If agent α plays S^1_α with probability γ and if agent β plays S^1_β with probability δ then α's expected payoff is

$$\begin{aligned} u_\alpha &= \gamma(3.0\delta + 3.0(1-\delta)) + (1-\gamma)(2.0\delta + 3.5(1-\delta)) \\ &= 1.5\gamma\delta - 0.5\gamma - 0.5\delta + 3.5. \end{aligned}$$

The first order condition is

$$\begin{aligned} 0 &= \frac{du_\alpha}{d\gamma} = 1.5\delta - 0.5 \\ \Rightarrow \delta &= \frac{1}{3}. \end{aligned}$$

Similarly for agent β

$$\begin{aligned} u_\beta &= \delta(2.0(1-\gamma)) + (1-\delta)(0.5\gamma + 0.5(1-\gamma)) \\ &= -2.0\gamma\delta + 1.5\delta + 0.5. \end{aligned}$$

The first order condition is

$$\begin{aligned} 0 &= \frac{du_\beta}{d\delta} = -2.0\gamma + 1.5 \\ \Rightarrow \gamma &= \frac{3}{4}. \end{aligned}$$

Therefore, in Nash equilibrium, agent α plays S^1_α with probability $\frac{3}{4}$ and agent β plays S^1_β with probability $\frac{1}{3}$. $\square$

Suboptimal Outcome

It is often of interest to ask whether an outcome is "optimal". An essential requirement for any optimal outcome is that it possess the property of *Pareto efficiency.* An outcome is Pareto efficient if there is no alternative outcome where some agent is better off without making some other agent worse off. Unfortunately, as we show below, in the setting where there is uncertainty as to which agent will be the proposer, agents may allocate their deliberation resources in a nonoptimal manner in equilibrium, so the outcome will not be Pareto efficient. In other words, if the agents would use different deliberation strategies, they would both be better off.

Theorem 9. *There exist instances (defined by $\mathcal{T}^\alpha$, $\mathcal{T}^\beta$, and $\mathcal{T}^{\text{joint}}$) of the game where the outcome of the unique Nash equilibrium is not Pareto efficient. This result does not depend on independence of performance profiles.*

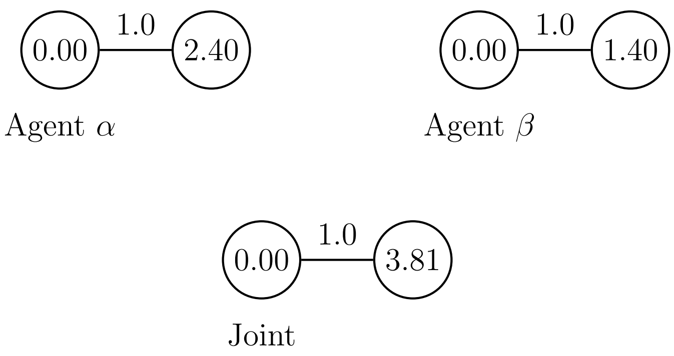


Figure 6.4: *Performance profile trees where the equilibrium outcome is not Pareto efficient.*

Proof. Consider the performance profiles in Figure 6.4. Let the probability that agent α will be the proposer be $\frac{1}{2}$. The agents are allowed only one deliberation step each, $(D = 1)$. Let $\emptyset$ represent a null offer, where the proposer does not want to implement a joint solution.

The undominated strategies for agent α are

- $S^1_\alpha = \{a^\alpha, (\emptyset, 2.4)\}$: Agent α computes on its own problem and makes a null offer if it is the proposer. Otherwise, it accepts anything greater than or equal to 2.4.

- $S^2_\alpha = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent α computes on the joint problem and offers nothing if it is the proposer. Otherwise, it accepts anything greater than or equal to 0.0.

- $S^3_\alpha = \{a^{\text{joint}}, (1.4, 0.0)\}$: Agent α computes on the joint problem and offers 1.4 if it is the proposer. Otherwise, it accepts anything greater or equal to 0.0.

The undominated strategies for agent β are

- $S^1_\beta = \{a^\beta, (\emptyset, 1.4)\}$: Agent β computes on its own problem and makes a null offer if it named as the proposer. Otherwise it accepts any offer greater than or equal to 1.4.

- $S^2_\beta = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent β computes on the joint problem and offers nothing if it is the proposer. Otherwise, it accepts anything greater than or equal to 0.0.

	S_β^1	S_β^2	S_β^3
S_α^1	2.4, 1.4	2.4, 0.0	2.4, 0.705
S_α^2	0.0, 1.4	1.905, 1.905	3.105, 0.705
S_α^3	1.205, 1.4	1.205, 2.605	2.405, 1.405

Table 6.3: *Normal form representation of the bargaining game where the performance profiles are found in Figure 6.4, and the probability of agent α being the proposer is $\frac{1}{2}$. The pure strategy Nash equilibrium is (S_α^1, S_β^1). The Pareto efficient outcome is (S_α^3, S_β^3).*

- $S_\beta^3 = \{a^{\text{joint}}, (2.4, 0.0)\}$: Agent β computes on the joint problem and makes an offer of 2.4 if it is the proposer. Otherwise it accepts anything greater or equal to 0.0.

The game can be represented in normal form (Table 6.3). There is a unique pure Nash equilibrium where agent α plays strategy S_α^1 and agent β plays strategy S_β^1, that is, both agents compute on their own problems. However, the equilibrium outcome is not Pareto efficient. Both agents would be strictly better off if agent α played S_α^3 and agent β played S_β^3. Unfortunately, the strategies S_α^3 and S_β^3 are not in equilibrium. If agent α played S_α^3 then agent β would deviate to S_β^2. Similarly, if β played S_β^3 then agent α would deviate to S_α^2. □

A General Method for Solving the Game with an Unknown Proposer

In general, solving an unknown proposer problem is difficult, as neither agent may have a dominant strategy. Instead, the strategy of one player depends on the strategy of the other. One approach of solving for perfect Bayesian equilibria is to convert the game into its *normal form* by considering all pure strategies for each player and the resulting payoffs when these strategies are employed. There are relatively efficient algorithms for solving normal form games [1, 119], but the conversion itself usually incurs an exponential blowup since the number of pure strategies is often exponential in the depth of the game tree because a pure strategy specifies a move for each information set of the player.

A more recent approach is to represent the extensive form game in its *sequence form* [115]. In the rest of this subsection we show how that technique can be used in our setting. A sequence of choices of a player corresponds to a node a in the game tree. The sequences replace the set of pure strategies in the normal form. In our setting a sequence is either;

1. $\emptyset$, the empty sequence,
2. a sequence of deliberation actions,
3. a sequence of deliberation actions followed by a proposal, or
4. a sequence of deliberation actions followed by either an accept or reject action.

All nodes in an information set, I, of an agent are defined by the same sequence, σ_I. If, after reaching information set I, an agent then makes action c, the new sequence is denoted $\sigma_I c$. The set SQ_α denotes the set of all sequences for agent α. Similarly the set SQ_β is the set of all sequences for agent β.

Payoffs to agents α and β are represented by matrices A and B respectively. Each row corresponds to a sequence of agent α and each column corresponds to a sequence of agent β. Every leaf in the game tree defines a pair of sequences, that is, actions that both agents must have taken in order to reach that node. For each sequence pair defined by a leaf node, the agent's payoff is the payoff it received at the leaf node if there are no chance moves. If there are chance moves, as there are in our setting, then a pair of sequences may correspond to more than one leaf node. The payoff entry is the sum of the payoffs over all leaves that correspond to the sequence pair weighted by the probabilities of reaching the leaves given the sequence pair. If a pair of sequences does not correspond to a leaf node, then the payoff entry is zero. The matrices, A and B, are sparse as the only (possible) nonzero entries occur at sequence pairs defined by leaf nodes which is linear in the size of the game tree.

Both agents also have realization plans, which are nonnegative vectors that represent the realization probabilities for the sequences of the agent when it is playing a mixed strategy. Let x be the realization plan for agent α and let y be the realization plan for agent β. The plans for agent α are characterized by the following constraints.

$$x(\emptyset) = 1$$

$$-x(\sigma_I) + \sum_{c \in C_I} x(\sigma_I c) = 0$$

for all information sets I of agent α where C_I is the set of possible moves that agent α can make at information set I. This means that at any information set, I, the probability of reaching I is the same as the sum of the probabilities of taking an action that leaves I. The constraints for the realization plan, y, for agent β are similarly defined.

The realization plans can be represented by

$$Ex = e \text{ and } Fy = f$$

where E and F are constraint matrices. The first row is $(1, 0, 0, \dots)$ and corresponds to the empty sequence having probability one. The other rows corresponds to the information sets of the respective agent. The vectors e and f are equal to the vector $(1, 0, 0, \dots, 0)^T$ which is of the appropriate size.

The Nash equilibrium of the game is a solution to a linear programming problem. The vectors x and y are in Nash equilibrium if they are mutual best responses. If y is fixed, then x is a best response if and only if it is an optimal solution to the linear program

$$\begin{aligned} \text{maximize}_x \quad & x^T(Ay) \\ \text{subject to} \quad & x^T E^T = e^T, \\ x \quad & \geq 0. \end{aligned}$$

The dual linear program is

$$\begin{aligned} \text{minimize}_p \quad & e^T p \\ \text{subject to} \quad & E^T p \geq Ay \end{aligned}$$

where p is an unconstrained vector of variables.

Similarly, y is a best response to x if it is an optimal solution to

$$\begin{aligned} \text{maximize}_y \quad & y^T(Bx) \\ \text{subject to} \quad & y^T F^T = f^T, \\ y \quad & \geq 0. \end{aligned}$$

with dual

$$\begin{aligned} \text{minimize}_q \quad & f^T q \\ \text{subject to} \quad & F^T q \geq Bx \end{aligned}$$

where q is an unconstrained vector of variables.

The feasible solutions to the linear programming problems are optimal only if the two objective function values are equal. That is, x is a best response to y only if

$$x^T(-Ay + E^T p) = 0$$

and y is a best response to x only if

$$y^T(-Bx + F^T q) = 0.$$

Any Nash equilibrium x, y is part of a solution x, y, p, q to the previous constraints. These constraints define a linear complementarity problem and therefore the solution to the linear complementarity problem is also a Nash equilibrium [1]. The standard linear complementarity problem is to find a vector $z \in \mathbb{R}^n$ so that

$$\begin{aligned} z &\geq 0 \\ b + Mz &\geq 0 \\ z^T(b + Mz) &= 0. \end{aligned}$$

where $b \in \mathbb{R}^n$ and M is an $n \times n$ matrix.

It is possible to create a linear complementarity problem that is equivalent to the linear programming problems [53]. First, set

$$M = \begin{pmatrix} & -A & E^T & -E^T & & \\ -B^T & & & & F^T & -F^T \\ -E & & & & & \\ E & & & & & \\ & -F & & & & \\ & F & & & & \end{pmatrix}$$

and

$$b = \begin{pmatrix} 0 \\ 0 \\ e \\ -e \\ f \\ -f \end{pmatrix}.$$

Let $z = (x, y, p', p'', q', q'')^T$ where p', p'', q', q'' are nonnegative vectors such that $p = p' - p''$ and $q = q' - q''$.

Using this representation, the equilibria for the extensive form game can be determined by similar algorithms that are known for the normal form, such as Lemke's algorithm for solving linear complementarity problems. Often, these algorithms run exponentially faster than with the standard approach since the size of the sequence form is linear, and not exponential, in the size of the game tree [53].

However, these are general techniques which do not take advantage of specific properties of the particular game. It can be the case that the performance profiles will affect the payoffs in such a way that finding the equilibrium strategies is straightforward (for example, in situations where it is always better not to coordinate actions and implement the joint solution). Thus, specially designed algorithms can sometimes take advantage of the special structure and be more efficient than general techniques for computing equilibria. Earlier in this chapter we presented such specialized algorithms for the setting with a known proposer, but currently we only have general algorithms for the setting with an unknown proposer.

6.6 Alternating-offers Bargaining Game

In this section we study a different bargaining game: the alternating offers model. In this model the agents take turn making offers and counteroffers about whether to perform a joint plan or whether to act individually. However, unlike in the single-shot bargaining model where most computation is done before the agents negotiate, in the alternating-offers model the computing actions and the negotiation actions are intertwined. The process is divided into discrete stages. At each stage agents can compute one step on one of the three problems. After each agent has made the single computing step, one agent is allowed to make an offer to the other, specifying how much it would be willing to pay if the other agent agreed to cooperate on the joint problem, using the solution computed by the proposing agent. The agent receiving the proposal can either accept or reject. If the offer is accepted, the joint solution is implemented and the game ends. If the offer is rejected, the game continues for another stage, where the roles of the agents are switched. While both agents can observe all proposals and responses, the computational actions of the agents are private, i.e., an agent cannot directly observe what the other has computed on, but can only try to deduce this information from the other agent's offers and accept–reject decisions.

6.6.1 Formal Model

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

Definition 38. *Assume that at time t agent i is the proposer. An* action *for agent i at time t is*

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

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

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

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

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

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

Definition 39. *At time t, a* history*, $H(t)$, is*

$$H(t) = c^z_\alpha(t-1) \times c^z_\beta(t-1) \times \phi_\alpha(t-1) \times \phi_\beta(t-1) \times ((x, \text{res})_i)_{i=0}^{t-1}$$

where $(c^z_i(t-1))$ is the sequence of computing steps that agent i has taken up until time $t-1$, $\phi_i(t-1)$ is its current state of deliberation, and $((x, \text{res}_i)_{i=0}^{t-1}$ is the sequence of all offers and responses observed up until time $t-1$.

Let $\mathcal{H}_i(t) = \{H(t)\}$ be the set of all possible histories at time t. A strategy for an agent specifies an action for every possible history after which it has to move.

Definition 40. *A* strategy *for agent i, S_i, is*

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

where D is agent i's deadline and

$$\sigma_i^t : \mathcal{H}_i(t-1) \to A_i(t).$$

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

In our analysis we use the *perfect Bayesian equilibrium* which was described in Chapter 4.

6.6.2 Agents' Beliefs and the Role of Signaling

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

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

First, we require that beliefs be consistent. For example, if at time $t-1$, agent i believed that $b_i(\phi_j(t-1)) = 0$ for some computation state $\phi_j(t-1)$, then agent i must have belief $b_i(\phi_j(t)) = 0$ if computation state $\phi_j(t)$ can only be reached by passing through $\phi_j(t-1)$. Second, agents obtain information from their own states of computation. This information can be used to update their beliefs about what computation states the other agent is in. In particular, given its own state of deliberation at time t, $\phi_i(t)$, agent i is able to compute the probability with which it is possible to move from some state $\phi_j(t-1)$ to $\phi_j(t)$ by either reading the probabilities from the edges of the performance profile subtrees rooted at the nodes in $\phi_i(t)$, or from already having computed along paths that contained nodes in $\phi_j(t)$. We denote by $P_{c^z}(\phi_j(t-1) \to \phi_j(t)|\phi_i(t))$ to be the probability that agent j may have moved from computation state $\phi_j(t-1)$ to $\phi_j(t)$, given that it took computing action c^z and agent i's own knowledge from computing.[5] Then

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

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

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

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

$$b'_i(\phi'_j(t)) = \frac{P(x|\phi'_j(t)) b_i(\phi'_j(t))}{\sum_{o \in \{\phi_j(t)\}} P(x|o) b_i(o)}.$$

[5]Even though agent i does not know with certainty what action agent j has taken, it does know that agent j is trying to maximize its own utility and thus, can reason about the likelihood of a certain action being taken since it has information about possible outcomes of each action.

6.6.3 Results

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

Common Deadline

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

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

Deterministic Performance Profiles

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

Theorem 10. Case 1: *Deadline D is odd.*

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

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

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

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

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

That is, agent α computes only on the joint solution, makes no offers and accepts only offers that are greater than the difference between the best computable joint solution and the best computable solution for agent β's individual problem. It is required that agent α knows the performance profiles of agent β, but the result is independent of whether agent β knows the performance profiles of agent α.

2. *$v^\alpha(D) \le v^{\text{joint}}(D) - v^\beta(D)$ and the solution to β's individual problem flattens out. Agent β has multiple equilibrium strategies. It computes until it obtains the maximum value for its own problem and then may compute on any of the problems. It makes empty proposals and accepts any offer greater than its fallback. Agent α's best response strategy remains the same as in the first situation.*

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

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

5. *$v^\alpha(D) > v^{\text{joint}}(D) - v^\beta(D)$ and β's performance profile flattens out. There is no longer an undominated strategy for agent β. Once it has computed the maximum value for its individual problem then it may continue computing on any of the three problems. It makes empty proposals and accepts any offer which is greater than its maximum fallback value. Agent α has the same dominant strategy as in situation 4.*

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

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

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

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

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

Stochastic Performance Profiles

In the stochastic setting there is uncertainty associated with the values being computed. Agents no longer generally have dominant strategies when it comes to computing and bargaining. A proposal becomes a signal which leaks information to the other agent about what values the agent has computed. Each agent would like the other to believe that it is in a strong bargaining position, i.e., it has a high fallback value and is willing to bargain hard in order to get the best possible deal. We obtain the following result.

Theorem 11. *Assume that the deadline D is odd and that there exists at least one path in each performance profile that does not flatten out before D. Agent β has a dominant strategy $(\sigma_\beta(t))_{t=1}^{D}$ where*

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

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

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

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

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

Proof. The proof follows a similar argument to that of Theorem 10. The difference is that the agents no longer know with certainty what final solutions their opponent may have obtained from computing as there is uncertainty in the performance profiles. Therefore, the agent who gets to make the final offer at time D may use some of its own computational resources in order to refine its beliefs about what its opponent's fallback value is. □

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

The beauty of this result is that it takes a complex situation and reduces it to a simple one. While the space of possible actions for both agents is extremely large, in equilibrium agents behave as if they were in a restricted action space where only one agent is allowed to propose at only one time point (i.e., at the deadline). This simplified problem was discussed in an earlier paper where an algorithm was presented for determining the agents' equilibrium strategies (Algorithm 1). Given Theorem 11, this algorithm can also be used in the alternating offers setting. Then, using the equilibrium strategies, and depending on the algorithms' performance profiles and the problem instance, agreement may or may not be reached at the deadline.

Different Deadlines

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

What is interesting in this setting is that the agent who has the most *power* in the negotiation is not necessarily the one with the latest deadline (unlike in, e.g., [111]).

Instead, which agent gets to make the offer at the earliest deadline also plays a vital role in which strategies the agents will want to execute.

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

Theorem 12. *With different deadlines that are common knowledge, Theorems 11 and 12 hold if $v^\alpha(D)$ is replaced by $v^\alpha(d_\alpha)$, $v^\beta(D)$ is replaced by $v^\beta(d_\beta)$ and $v^{\text{joint}}(D)$ is replaced by $v^{\text{joint}}(\min[d_\alpha, d_\beta])$.*

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

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

Theorem 13. *If the agents' private deadlines are drawn from a joint distribution, $f(d_\alpha, d_\beta)$, then there will be no proposing and counter-proposing until the earliest deadline is reached. At that time an agreement may or may not occur.*

6.6.4 Comparison with Classic Results

The alternating-offers bargaining game has been well studied in the game theory and economics literature (see, for example [71, 100]). The standard model of the game consists of two agents bargaining over how to split v dollars. In the first period, one agent can propose any split and the other agent can accept of reject it. If the offer is accepted then the proposed split is implemented and the game ends. If the offer is rejected, then the second agent proposes a split and the first agent has the option of accepting or rejecting. Additionally, a time discount factor, δ, is introduced so that a dollar now is worth more than a dollar later.

There are two variations of the game that are normally studied. In the first, it is assumed that there is a finite number of bargaining rounds, D, after which, no agreement can be reached and both agents get payoffs of zero. In the second variation there is no deadline, instead agents are allowed to bargain for an infinite number of rounds. Interestingly, the equilibrium in both variations specifies that agents should reach agreement in the first round. In the finite horizon model, the first agent is best off offering a split $(\delta^{D-1}v, 0)$ and the second agent should accept it. In the infinite horizon model the first agent is best off offering a split $(\frac{v}{(1+\delta)}, \frac{\delta v}{(1+\delta)})$, and the second agent should accept it.

In this chapter we showed that in the alternating offers game, agents do not communicate or reach agreement until the deadline. At first glance this appears to be in contradiction of the classic results. However, there are some fundamental differences in our alternating-offers game compared to the classic game. First, in our model the agents do not start with some initial value v in which they have to decide whether to split. Instead, they require time to compute on different problems in order to determine the joint value, as well as their fall back values if the bargaining fails to reach an agreement on a split. Second, we do not introduce a time discount factor in our model. Instead, time is used to *improve* the values (or improve knowledge about the values), which can result in better payoffs for the agents. Additionally it should be noted that cost functions are different from time discount factors. The cost incurred by an agent only increases (or remains the same) if the agent actively takes a computing step. In our model agents are permitted to not compute for a time step, thus not incurring any penalty. A discount factor, on the other hand, decreases the value each time step, no matter what the agents do. If we modified our alternating-offers model, so that *all* computation must be completed before any bargaining occurred, and additionally introduced a time discount factor for each bargaining round, then we would have a similar game to the classic alternating-offers game, and the results in the classic model could be applied (to at least the bargaining part of the game).

There has also been research on bargaining models where agents have deadlines [111]. Sandholm and Vulkan present a model where agents with private deadlines negotiate over how to split an item. At each point in time an agent can change its offer, however once the first deadline is reached, if there is no agreement, then neither agent gets the item. The only sequential equilibrium outcome is one where the agents wait until the first deadline, and the agent with the earlier deadline concedes everything to the other agent. This result still holds if discounting is included in the model.

There are some parallels between our results and the Sandholm-Vulkan results. In our model, if an agreement is reached, it is reached at the first deadline. One of the reasons for this is, like in the Sandholm-Vulkan paper, early offers signal too much information about the agent, thus weakening its bargaining position. However, our results do not directly follow from the earlier work since our model has some substantial differences. First, in our model it is possible that the agents will reach an agreement to split the value of the joint solution, since agents potentially have non-zero fallback values. Second, time plays an additional role in our model. Time is considered to be a resource that agents use to compute or gather information on both the joint value and the fallback values. This potentially provides an additional motivation for agents to postpone agreement, as they wish to compute for the appropriate amount of time so as to improve the payoffs they will receive in the final outcome.

6.7 Summary

In this chapter we studied negotiation protocols where two agents must reach agreement as to whether to cooperate in order to execute some joint problem, or whether to both act independently. Additionally, if agents decide to coordinate their actions, they must reach agreement as to how to divide any costs or awards associated with the successful completion of the task. In our model, the agents must exert computational effort in order to determine the value of the different outcomes.

We studied two different protocols. The first bargaining model we used was one where one agent is allowed to make a single proposal which the other agent can choose to accept or act independently. Under different assumptions about whether it was known in advance which agent would be the proposer, when the deadline would occur, and the degree of uncertainty that arose through computing, we were able to analyze different scenarios. We showed that if an agent knew that it would never propose then it had a dominant strategy which was to compute only on its own problem. The other agent, who knew it was the proposer, could best respond to this strategy. We provided algorithms for determining the best-response strategy under settings where the deadline was known in advance or only probabilistically. If the proposer was unknown then agents did not necessarily have dominant strategies. Instead, we described a general method that could be used for finding the equilibrium. We also showed that there a pure strategy equilibrium may not exist, and that the equilibrium outcome may not be efficient.

The second setting we studied was an alternating-offers model where agents could take turns making offers and counter-offers, at the same time as they computed to find solutions for the different problems. We provided an equilibrium analysis of this setting, and showed that because of the information leaked during the negotiating rounds, agents were best off acting as though they were playing an ultimatum game, where the proposing agent was the one who was able to propose at the deadline. This means that in order to determine the optimal deliberation policies for agents in an alternating-offers negotiation protocol, we can use the algorithms developed for the ultimatum game.

A problem that is not addressed in the model presented in this chapter, is the situation where after agents have reached an agreement, a better, outside offer appears, thus causing one of the agents to want to cancel the agreement. However, we believe that the bargaining approaches outlined in this chapter can be coupled with *leveled commitment contracts* in order to provide additional stability for the agreements [110, 112].

Chapter 7

One-to-Many Negotiation: Auctions

In this chapter we study the behavior of computationally limited agents who are bidding in different auction mechanisms. We assume that the agents do not *a priori* know their valuations for the items being auctioned. Instead, they must use their limited computing resources in order to determine their valuations, which in turn influences how they bid. We are interested in discovering two things:

1. How different forms of computational limitations influence the strategic actions taken by the agents.

2. How different types of auction mechanisms influence the strategic behavior of computationally limited agents.

The rest of the chapter is organized in the following way. In the next section (Section 7.1) we describe the auction mechanisms studied in this chapter and describe the equilibria which occur when bidding agents are fully rational. In Section 7.2 we study the impact that computational limitations have on the optimal strategies of agents participating in different auction mechanisms. In particular, we show that strategic deliberation occurs in all common auction mechanisms when agents have a cost associated with computing. We provide a discussion about what happens when agents are faced with a choice of algorithms for computing their valuations (Section 7.3) and also show that the Vickrey auction and ascending auction are not strategically equivalent if the bidding agents are computationally limited (Section 7.4).

We conclude this chapter by presenting results from a series of experiments that were performed in order to discover whether the strategic deliberation was an artifice of the analysis, or whether it was something which occurred under realistic conditions (Section 7.5).

They can also compute on other agents' valuation problems so as to gain information about the others' valuations, allowing for better strategic bidding. We present a fully normative model of deliberation control and define agents' strategies in this setting, as well as the concepts of strong and weak strategic computing. We analyze different auction mechanisms and settings, presenting results on whether or not strategic computing occurs in equilibrium.

7.1 Auctions

Auctions are stylized markets and have been well studied in the game theory and economics literature [74, 123]. The term *auction* is used for settings where there is one seller of items and multiple buyers. In particular, auctions are bidding mechanisms that are described by a set of rules which specify how a winner is determined, and how much the winner should pay. Auctions can be classified in many different ways. As is commonly done, in this paper we focus solely on auction categories where agents have private values, are risk neutral and have quasilinear utility functions. That is, the value of an item to a bidder depends only on that bidder's own preferences (private value), and if an agent wins an item in the auction, the agent's utility is equal to its own value for the item minus the amount that it must pay ($u_i = v_i - p_i$). There are many different types of auction mechanisms that have these properties. In this paper we will study standard single item and multiple item auctions.

Single-Item Auctions

As the name implies, in single item auctions there is only one item for sale. Agents place bids on the item and the auctioneer determines who gets the item and what amount should be paid. There are three commonly studied single item auctions; first-price sealed-bid, ascending, and Vickrey.

- **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 its bid. There is no dominant bidding strategy for rational agents. An optimal strategy depends on the bids of the other agents [58].

- **Ascending auction:** The auctioneer raises the price of the item. At any point in time a bidder can decide to withdraw from the auction. It does this by making an announcement to the auctioneer. Once a bidder has left the auction, it is not allowed to reenter. For rational agents there is a dominant strategy. An agent should stay in the auction until the price is equal to its value. As soon as the price rises above an agent's value, then that agent should withdraw.[1]

- **Vickrey auction:** The Vickrey auction is also known as the *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 (weakly) dominant strategy which is for each agent to bid its true valuation [58]. The Vickrey auction is a member of the Vickrey-Clarke-Groves family of mechanisms (Chapter 4).

Multi-Item Auctions

In auctions where multiple distinguishable items are sold, bidding strategies for agents can be complex. 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 [107].

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 $X = \{x_1, \ldots, x_m\}$ be a set of items. A bundle is a subset of the items, for example, $\{x_1\}$ or $\{x_1, x_m\}$. An allocation of items among a set, N, of agents is $Y = (y_1, \ldots, y_{|N|})$ where $y_i \subseteq X$, $\cup_{i=1}^{|N|} y_i \subseteq X$ and $y_i \cap y_j = \emptyset$ for $i \neq j$. The *generalized Vickrey auction* (GVA) is a

[1]Note that an ascending auction is not the same as an English auction.

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 [20].

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

1. Each agent declares a valuation function. So $v_i(y_i)$ is agent i's valuation for allocation Y where it is given y_i.

2. The GVA chooses an optimal allocation $Y^* = (y_1^*, \ldots, y_{|N|}^*)$ that maximizes the sum of all the agents' declared valuations.

3. The GVA announces the winners and their payment p_i:

$$p_i = \sum_{j \neq i} v_j(y'_j) - \sum_{j \neq i} v_j(y_j^*)$$

   where $Y' = (y'_1, \ldots, y'_{i-1}, y'_{i+1}, \ldots, y'_{|N|})$ 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 $u_i(y_i) = v_i(y_i) - p_i$, the utility of bidder i in the GVA is

$$u_i(y_i^*) = v_i(y_i^*) - p_i = v_i(y_i^*) + \sum_{j \neq i} v_j(y_j^*) - \sum_{j \neq i} v_j(y'_j).$$

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).

GVA Example: We now provide an example to illustrate how the GVA works. Let there be two agents, agent α and agent β, and let there be two items, g_1 and g_2. Agents can bid on either item or on the bundle $\{g_1, g_2\}$. An agent's bid is represented by a tuple: (a bid for $\{g_1\}$, a bid for $\{g_2\}$, a bid for $\{g_1, g_2\}$ where the bids are XOR'ed together). Suppose the agents bid as follows

Auction	Dominant strategy equilibrium?
First-price-sealed-bid	no
Ascending	yes
Vickrey	yes
GVA	yes

Table 7.1: *For rational agents the type of equilibrium outcome depends on the auction rules. For example, in a Vickrey auction, agents have dominant strategies while in a first-price-sealed-bid auction, an agent's optimal strategy depends on what strategies other agents are following.*

- Agent α's bid: (20, 5, 25)
- Agent β's bid: (10, 15, 30)

The GVA allocates g_1 to agent α and g_2 to agent β since this allocation maximizes the sum of the agents' valuations. The amount that each agent pays is computed as follows. If agent α did not bid, then $\{g_1, g_2\}$ would have been allocated to agent β whose valuation for this bundle is 30. When g_1 is allocated to agent α, agent β's valuation is only 15 since it receives g_2. Therefore, agent α's payment is calculated as $30 - 15 = 15$ and its utility is $20 - 15 = 5$. Agent β's payment is $25 - 20 = 5$ and its utility is $15 - 5 = 10$.

Comparison of Auctions

The auctions described have been well studied for the rational agent setting. The ascending auction, Vickrey auction and the generalized Vickrey auction all have dominant strategy equilibria or optimal best response equilibria [58, 118] . This means that agents participating in the auction do not have to guess what strategies other agents are following in order to play optimally. The first-price-sealed-bid auction has only Nash equilibria. An agent's optimal strategy depends on the strategies of the other agents. Table 7.1 shows which auctions have dominant strategy equilibria for rational agents.

7.2 Analysis

In this section we study different auction types in order to understand how the restrictions placed on the bidding agents computing capabilities affect their strategies. There are some standard game-theoretic assumptions that are made in this section. Unless otherwise noted, we assume that the agents' performance profiles and cost functions are common knowledge and that they are independent. We also assume that given a feature vector for a problem of some agent, it is possible for all other agents to determine the value of that solution from the feature vector. We assume that all agents are risk neutral, that we are in a private value setting, and an agent's utility is defined as

$$u = \begin{cases} v - p - \text{cost} & \text{if the agent is allocated the item} \\ -\text{cost} & \text{otherwise} \end{cases}$$

where v is the (computed) value of the item, p is the price paid for the item, and cost is the cost the agent incurred by determining the value.

7.2.1 First-price Sealed-bid Auctions

In a first–price sealed–bid auction each agent submits one bid without knowing what the other agents' have bid. The strategy, S_i, of agent i is defined as follows

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

where

$$\sigma_i^t : \Phi_i(t-1) \mapsto C_i \text{ if } t \neq D$$

and

$$\sigma_i^t : \Phi_i(t-1) \mapsto C_i \times \mathbf{R} \text{ if } t = D$$

where $\Phi_i(t)$ is the set of deliberation states at time t and C_i is the set of all possible computing actions (see Chapter 5 for full explanation of notation).

The highest bidder wins the item and pays the amount of its bid. For fully rational agents there is no dominant bidding strategy for agents. An optimal strategy depends on the bids of the other agents. If an agent knows what the submitted bids of other agents are, then it can tailor its bid so as to maximize its own utility. Not surprisingly, computing agents may have incentive to use some of their computing resources in order to discover what valuations the other agents may have for the item up for auction as allowing agents to compute on each others problems simply supplies the agents with a strong tool for concrete speculation. This is independent of whether the agents have costly or free but limited computing resources, whether agents must do all their computing before the auction occurs or whether they are allowed to compute after the bids have been submitted and a winner has been determined. The agents deliberation and bidding strategies depend upon the performance profiles that the agents have for their valuation problems.

While in this auction strategic computing is the norm, situations where there are no strategic deliberation is of interest. We find that in general, for the first-price sealed-bid auction the only time when agents do not use strategic deliberation is when the performance profiles of any type contain no uncertainty. That is, that all the performance profiles can be transformed to deterministic performance profiles. This is independent of whether computing is free or limited, and whether agents compute to improve or refine their valuations.

Theorem 14. *Assume that agents in a first-price sealed-bid auction all have deterministic performance profiles. Then only weak strategic deliberation will occur in Nash equilibrium. This is independent of whether agents have costly or limited computing and whether agents are computing to improve or refine their valuations.*

Proof. Assume that all agents are computing to refine their valuations. Assume also that all agents have deterministic performance profiles. As there is no uncertainty to be removed by computing, no strategic deliberation will occur either. However, weak strategic deliberation can occur as agents can check each others' performance profiles so as to learn what the different valuations are.

Assume that agents are computing to improve their valuations. If the agents all have deterministic performance profiles then all agents can determine what valuations the other agents will be able to compute, given their deadlines (weak strategic deliberation). Agents have (weakly) dominant deliberation strategies where they compute solely on their own problem. Overall, the agents' strategies will not be (weakly) dominant as the bid submitted by one agent depends on what it believes the other agents will submit as bids. $\square$

7.2.2 Ascending and Vickrey Auctions

In this subsection we study the strategic behavior of computationally limited agents in Vickrey and ascending auctions. The strategy space for agents participating in the Vickrey auction is the same as the strategy space for agents participating in a first-price sealed-bid auction (though the equilibrium strategies differ). The strategies of a computationally limited agent, i, in an ascending auction are defined as

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

where

$$\sigma_i^t : \Phi_i(t-1) \times \mathbb{R} \mapsto C_i \times \{\text{stay in, exit}\}$$

where $\Phi_i(t)$ is the set of deliberation states at time t, C_i is the set of computing actions, and {stay in, exit} is the set of decisions that the agent can make about whether to stay in the auction at the current price, or leave the auction for good. In particular, the strategy specifies for each state of deliberation and price, what computing action the agent should take and whether it should remain in the auction.

If agents are fully rational, then in both the ascending and Vickrey auctions agents have dominant bidding strategies. In particular, the two auctions are strategically equivalent as it is possible to find an isomorphism between the equilibrium strategies. For computationally limited agents the optimal bidding strategies for agents in both the ascending and Vickrey auction are similar to those of fully rational agents. Using arguments identical to those used in the fully rational setting, it is possible to show that in a Vickrey auction, a computationally limited agent is best off submitting a bid equal to its computed value or equal to its expected computed value (since we assume agents are risk neutral). Similarly, an agent in an ascending auction is best off staying in the auction only until the price becomes greater than its computed value (or expected computed value). However, there is a potential difference between the deliberation strategies of agents in the auctions.

Consider the following simple example. Assume that there are two agents, α and β. Each agent has their own cost function, $\text{cost}_\alpha()$ and $\text{cost}_\beta()$, and performance profiles PP_α and PP_β. Finally, assume that the performance profiles are such that after one step of computing each agent will know its true value. $v_i \in [a_i, b_i]$. In a Vickrey auction, an agent has two time points where it is useful to acquire information about its value. It can either acquire information before it places a bid, or it can bid its expected value and then wait to determine its true value only after it has been allocated the item.[2] Since the auction is sealed-bid, an agent learns nothing about its competitor's actions. In an ascending auction, the agents have the luxury of being able to wait for further information before deciding whether to compute on their value problem. In the small example, even without computing agent i knows that its value must be at least a_i. It need not do any computing to learn its value while the price is below a_i since, if it is allocated the item, its utility will surely be greater than zero. Therefore it is possible that there exist situations where agents would compute to find their valuations in a Vickrey auction, but would not need to in an ascending auction.

The previous discussion was independent of the type of restriction on the agents computing resources. We will now study whether having costly or limited computing affects agents equilibrium strategies.

Limited Computing

If agents have limited computing, then in both an ascending auction and a Vickrey auction no form of strategic deliberation occurs. Agents have (weakly) dominant strategies which have them computing only on their own value problems, independent of their own and other agents' performance profiles and deadlines, and whether they compute to improve or refine their values. We show this by first proving two lemmas and then combining the lemmas into the final theorem.

Lemma 1. *Assume agents have limited deliberation and compute to improve their values. In an ascending auction or an a Vickrey auction agents have (weakly) dominant strategies which involve no strategic deliberation (not even weak strategic deliberation).*

Proof. Since agents have limited computing, for each agent i there exists a deadline D_i such that the cost function of agent i is

$$\text{cost}_i(\mathbf{t}) = \begin{cases} 0 & \text{if } \sum_j t_j \leq D_i \\ \infty & \text{if } \sum_j t_j > D_i \end{cases}$$

where $\mathbf{t} = (t_1, \ldots, t_m)$. This means that agent i can compute up to D_i time steps for free, but will never compute more than D_i steps as then $u_i = -\infty$. Assume that agent i computes $k_i \leq D_i$ on its own problem. Then, its value is $v_i(f_i(k))$.

[2]Recall, that if the agent computes to improve its value, then it *must* compute once it has acquired the item as the item has no value without the occurrence of computing.

First, consider a Vickrey auction. Assume that the highest bid among the set of agents not including agent i is b. If $v(f_i(k)) > b$ then agent i is allocated the object and $u_i = v(f_i(k)) - b$. Since agents are computing to improve their values, $v_i(f_i(k)) \leq v_i(f_i(D_i))$ and so agent i would have higher utility if it computed D_i steps on its own problem. If $b > v_i(f_i(k))$ then $u_i = 0$. If agent i computed for D_i steps then either $b > v(f_i(D_i))$ and so $u_i = 0$ or $b \geq v(f_i(D_i))$ and $u_i \geq 0$. Computing on another agent's problem does not change the second highest bid, and computing less time on its own problem only lowers its utility, therefore in a Vickrey auction agents should compute only on their own problem until their deadline.

In an ascending auction the argument is identical.

□

Lemma 2. *Assume agents have limited computing and compute to refine their values. In an ascending auction or Vickrey auction agents have (weakly) dominant strategies which involve no strategic deliberation (not even weak).*

Proof. Since agents have limited computing, for each agent i there exists a deadline D_i such that the cost function of agent i is

$$\text{cost}_i(\mathbf{t}) = \begin{cases} 0 & \text{if } \sum_j t_j \leq D_i \\ \infty & \text{if } \sum_j t_j > D_i \end{cases}$$

where $\mathbf{t} = (t_1, \ldots, t_m)$. This means that agent i can compute up to D_i time steps for free, Agent i will not compute for more than D_i steps since then $u_i = -\infty$.

Let Pr_i be the initial probability distribution over the set of all feature tuples which describe solutions for the problem of agent i. Let $\hat{s}_i$ denote the true solution and let $v_i(\hat{s}_i)$ denote the value of solution $\hat{s}_i$ to agent i. Recall from Chapter 2 that $v_i : \mathcal{F}_i \mapsto \mathbb{R}$.

Assume agent, i has computed $k_i \leq D_i$ steps on its own problem. This means that it has collected information info(k_i), and has current distribution $Pr_i^{\text{info}(k_i)}$ (definition of refining). This distribution, $Pr_i^{\text{info}(k_i)}$, induces a probability distribution over the space of values for solutions for problem i, $Val_i = \{v_i(s_i) | s_i \in \mathcal{F}_i\}$. Define the probability distribution over Val_i induced by $Pr_i^{\text{info}(k_i)}$ as $Pr_{Val_i}^{\text{info}(k_i)}$ where

$$Pr_{Val_i}^{\text{info}(k_i)}(v) = \sum_{\{s_i | v_i(s_i) = v\}} Pr_i^{\text{info}(k_i)}(\hat{s}_i = s_i)$$

Using this probability distribution, agent i is able to determine the expected value of the solution, given its current information info(k_i). That is

$$E[v_i(\hat{s}_i)|\text{info}(k_i)] = \sum_{v \in Val_i} v Pr_{Val_i}^{\text{info}(k_i)}(v).$$

In a Vickrey auction, an agent is best off submitting a bid equal to its expected value of the solution. At time k_i, agent i can either stop computing and submit a bid equal to $E[v_i(\hat{s}_i)|\text{info}(k_i)]$, compute on an opponent's problem, or compute on its own problem. Let b be the largest bid submitted among all agents other than i.

The utility of the agent for submitting a bid equal to $E[v_i(\hat{s}_i|\text{info}(k_i)]$ is

$$u_i = Prob(E[v_i(\hat{s}_i)|\text{info}(k_i)] \geq b) \left[\hat{s}_i - b\right].$$

If the agent begins to allocate resources to a competitors problem then either, it results in a utility $u_i = -\infty$ since they compute for more than D time steps in total, or, if the total time allocated is less than D, then the utility of the agent does not change. This is because the information gathered on other agents' problems does not change the information about the agent's own problem, which is the only factor that could improve the utility of the agent (independence of performance profiles). Thus, the agent is better off not computing as opposed to strategically deliberating.

In an ascending auction the argument is identical.

□

Combining Lemmas 1 and 2 we derive the following theorem.

Theorem 15. *If agents have limited computing then in both the ascending and Vickrey auctions there always exist (weakly) dominant strategy equilibria where no strategic deliberation occurs (not even weak strategic deliberation).*

Costly Computing

If agents incur a cost while computing then their optimal strategies may be quite different. It may be of use for an agent to actually compute on other agents' problems in order to learn what their values are. That is, strategic deliberation may occur in equilibrium.

To illustrate the phenomena we present a simple scenario. Assume that there are two agents, α and β, participating in an ascending auction. (The argument for a Vickrey auction is identical except that the timing of the computing may be different.) Assume that agent α has the performance profile PP_α such that before computing it knows that its value, v_α, after one step lies in the interval $[a, b]$, and the probability that its value is $v_\alpha = x$ is $f(x)$. After one step of computing it knows its true value. Assume that agent β has a performance profile PP_β such that before computing it knows that its value, v_β, after one step lies in the interval $[c, d]$ where $c \leq a$, and the probability that its value of $v_\beta = y$ is $g(y)$. After one step of computing it knows its true value. Finally, assume that the cost function of agent α is

$$\text{cost}_\alpha(\langle t_\alpha, t_\beta \rangle) = Bt_\beta \quad \text{where } B > b$$

and the cost function of agent β is

$$\text{cost}_\beta(\langle t_\alpha, t_\beta \rangle) = t_\alpha + 2t_\beta.$$

It is easy to show that agent α has a dominant strategy which is to compute one step on its own problem. Since the cost of computing on its competitor's, β, problem is higher than its highest possible computed value, agent α will never compute on it. If agent α chooses to not participate then its utility is

$$u_\alpha^{\text{nothing}} = 0.$$

An agent also has the possibility of computing for one step on its problem before submitting a bid. It would incur a cost for doing so (though in this example the cost to agent α is 0), but would be able to bid knowing its true value. An agent is also able to bid blindly. By this we mean that an agent can submit a bid without knowing in advance what its true value is. By doing so, the agent does not incur a cost if it does not win the auction since it never needs to compute on the problem, however the agent only knows its expected value for the item, and not its true value.

In expectation, for agent α the strategies of bidding blindly, or computing one step on its problem to determine its value have the same utility. That is, if agent β submits a bid, b^*, then

$$u_\alpha^{\text{blind}} = u_\alpha^\alpha = \int_a^b \int_c^{v_\alpha} (v_\alpha - b^*) f(v_\alpha) g(b^*) db^* dv_\alpha \geq 0.$$

Therefore, agent α is best off computing on its own problem only and staying in the auction until the price reaches its computed value.

To determine the best-response for agent β, one must recall that each computing step it takes incurs a cost. While in an ascending auction, agent β can wait until the price reached c before making a decision as to whether to compute or not, since in this example $c \leq a$, waiting provides no information about agent α's value.

Agent β always has the option of doing nothing. In such a situation, its utility is

$$u_\beta^{\text{nothing}} = 0. \tag{7.1}$$

Agent β may decide to compute only on its own problem. If it does decide to do this, then it will compute only one step since more steps incur a cost but do not refine or improve its value. Following this strategy results in expected utility

$$u_\beta^\beta = -2 + \int_c^d \int_a^{v_\beta} (v_\beta - v_\alpha) f(v_\alpha) dv_\alpha g(v_\beta) dv_\beta. \tag{7.2}$$

Agent β may decide to bid blindly. By this, we mean that agent β may decide to not compute on its value before bidding. Instead, it may bid only using its expected value. If it is allocated the item then under the improving model it must compute on its own problem, while under the refining model it does not need to compute since it learns its value once it obtains the item. Let

$$V = \int_c^d v_\beta g(v_\beta) dv_\beta.$$

The utility for agent β under the improving model is

$$u_\beta^{\text{blind\_improve}} = \int_c^d \int_a^V (v_\beta - v_\alpha - 2) f(v_\alpha) dv_\alpha g(v_\beta) dv_\beta. \tag{7.3}$$

and the utility for agent β under the refining model is

$$u_\beta^{\text{blind\_refine}} = \int_c^d \int_a^V (v_\beta - v_\alpha) f(v_\alpha) dv_\alpha g(v_\beta) dv_\beta. \tag{7.4}$$

Finally, it is possible that agent β will compute on agent α's problem. It will never compute on agent α's problem after it knows its own value since once it knows its own value it can bid optimally. However, there may exist situations where it can use information about agent α's problem to help decide whether to compute on its own problem. In particular, if agent β learns that the value for agent α is greater than some threshold v^* then agent β may decide not to compute on its own problem since the likelihood of it winning the auction is small, and so it wishes to avoid the cost of computing. The utility for agent β following a strategy where it first computes on agent α's value problem, and then computes on its own problem only if $v_\alpha \leq v^*$ is

$$\begin{aligned} u_\beta^{\alpha\text{-}\beta} &= -\int_{v^*}^{b} f(v_\alpha)dv_\alpha - 3\int_c^{v^*}\int_a^{v^*} f(v_\alpha)dv_\alpha g(v_\beta)dv_\beta \\ &+ \int_{v^*}^{d}\int_a^{v_\beta}(v_\beta - v_\alpha)f(v_\alpha)dv_\alpha g(v_\beta)dv_\beta \end{aligned} \tag{7.5}$$

where

$$\begin{aligned} v^* &= \arg\max_x \Bigg[-\int_x^b f(v_\alpha)dv_\alpha - 3\int_c^x\int_a^x f(x)dxg(v_\beta)dv_\beta \\ &+ \int_x^d\int_a^{v_\beta}(v_\beta - v_\alpha)f(v_\alpha)dv_\alpha g(v_\beta)dv_\beta\Bigg]. \end{aligned} \tag{7.6}$$

Agent β will partake in strategic deliberation only when Equation 7.5 is greater than Equations 7.1, 7.2, and 7.3 or 7.4. The question is

> ***Does there exist actual performance profiles such that strategic deliberation occurs?***

Using the performance profiles that were mentioned earlier, let agents α's performance profile be $[a, b] = [12, 30]$ with the distribution

$$f(x) = \begin{cases} p & \text{if } x = 30 \\ 1 - p & \text{if } x = 12 \\ 0 & \text{otherwise.} \end{cases}$$

Let the performance profile of agent β be $[3, 22]$ with distribution

$$g(x) = \begin{cases} q & \text{if } x = 22 \\ 1 - q & \text{if } x = 3 \\ 0 & \text{otherwise.} \end{cases}$$

Using these performance profiles, it is possible to determine the utility agent β would have from following the strategies outlines above. That is

$$u_\beta^{\text{nothing}} = 0$$

$$u_\beta^\beta = -2 + 10(1-p)q$$

$$u_\beta^{\text{blind\_imp}} = 8(1-p)q - 11(1-p)(1-q)$$

or

$$u_\beta^{\text{blind\_ref}} = 10(1-p)q - 9(1-p)(1-q)$$

and

$$u_\beta^{\alpha\_\beta} = -p + 7(1-p)(1-q) - 3(1-p)(1-q).$$

It is possible to determine for which values of p and q each strategy is dominant. In the case where agents compute to improve their values,each strategy is the dominant strategy for agent β is the following regions.

1. **Nothing**: The agent does not compute on any problem, and does not bid in the auction.

$$0 < p \leq \frac{1}{2}, \;\; 0 < q < \frac{-1}{5p-5}$$

$$\frac{1}{2} < p \leq \frac{53}{72}, \;\; 0 < q < \frac{2p-3}{10p-10}$$

$$\frac{53}{72} < p < 1, \;\; 0 < q < \frac{11}{19}$$

2. **Blind**: The agent submits a bid before its has done any computing. It only determines its value once it has won the auction.

$$0 < p \leq \frac{1}{2}, \;\; \frac{11p-9}{9p-9} < q < 1$$

$$\frac{1}{2} < p \leq \frac{53}{72}, \;\; \frac{9p-8}{9p-9} < q < 1$$

$$\frac{53}{72} < p < 1, \;\; \frac{11}{19} < q < 1$$

3. **Alpha_Beta**: The agent first computes on its competitor's problem, and then, based on the results it obtained, decides whether to compute on its own problem. Strategic deliberation occurs.

$$\frac{1}{2} < p < \frac{53}{72}, \quad \frac{2p-3}{10p-10} < q < \frac{9p-8}{9p-9}$$

4. **Beta**: The agent only computes on its own problem.

$$0 < p < \frac{1}{2}, \quad \frac{-1}{5p-5} < q < \frac{11p-9}{9p-9}$$

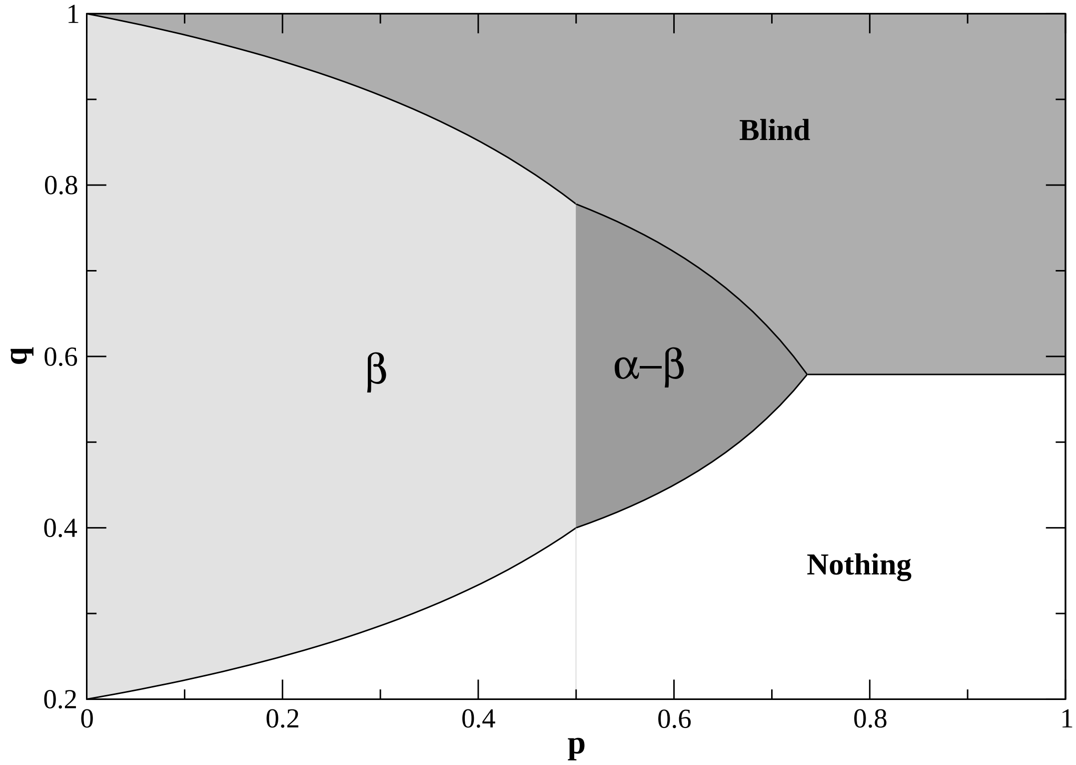


Figure 7.1: *Agents are computing to improve their valuations. For different values of p and q, different strategies for agent β are the best response to agent α's dominant strategy. If $\frac{1}{2} < p < \frac{53}{72}$ and $\frac{2p-3}{10p-10} < q < \frac{9p-8}{9p-9}$ agent β is best off computing one step on agent α's problem.*

If the two agents are computing to *refine* their values, then each of agent β's strategies are dominant in the following regions.

1. **Nothing**: The agent does not compute on any problem, nor does it submit a bid to the auction.

$$0 < p \leq \frac{1}{2}, \quad 0 < q < \frac{-1}{5p-5}$$

$$\frac{1}{2} < p \leq \frac{33}{52}, \quad 0 < q < \frac{2p-3}{10p-10}$$

$$\frac{33}{52} < p < 1, \quad 0 < q < \frac{9}{19}$$

2. **Beta**: The agent computes on its own problem only.

$$0 < p < \frac{1}{2}, \quad \frac{-1}{5p-5} < q < \frac{9p-7}{9p-9}$$

3. **Alpha_Beta**: The agent computes first on its competitor's problem, and then based on the results, decides whether to compute on its own problem. Strategic deliberation occurs.

$$\frac{1}{2} < p < \frac{33}{52}, \quad \frac{2p-3}{10p-10} < q < \frac{7p-6}{9p-9}$$

4. **Blind**: The agent bids without first computing on its problem. Since this is the refining model, once the agent is given the item it learns its true value.

$$0 < p \leq \frac{1}{2}, \quad \frac{9p-7}{9p-9} < q < 1$$

$$\frac{1}{2} < p \leq \frac{33}{52}, \quad \frac{7p-6}{9p-9} < q < 1$$

$$\frac{33}{52} < p < 1, \quad \frac{9}{19} < q < 1$$

Figure 7.2 illustrates these regions.

The example that has just been presented made no assumption as to the type of performance profile used, other than to assume that it was not deterministic. The example did not make any assumptions on whether the anytime algorithm was a contract or interruptible algorithm either. The following theorem has been proved.

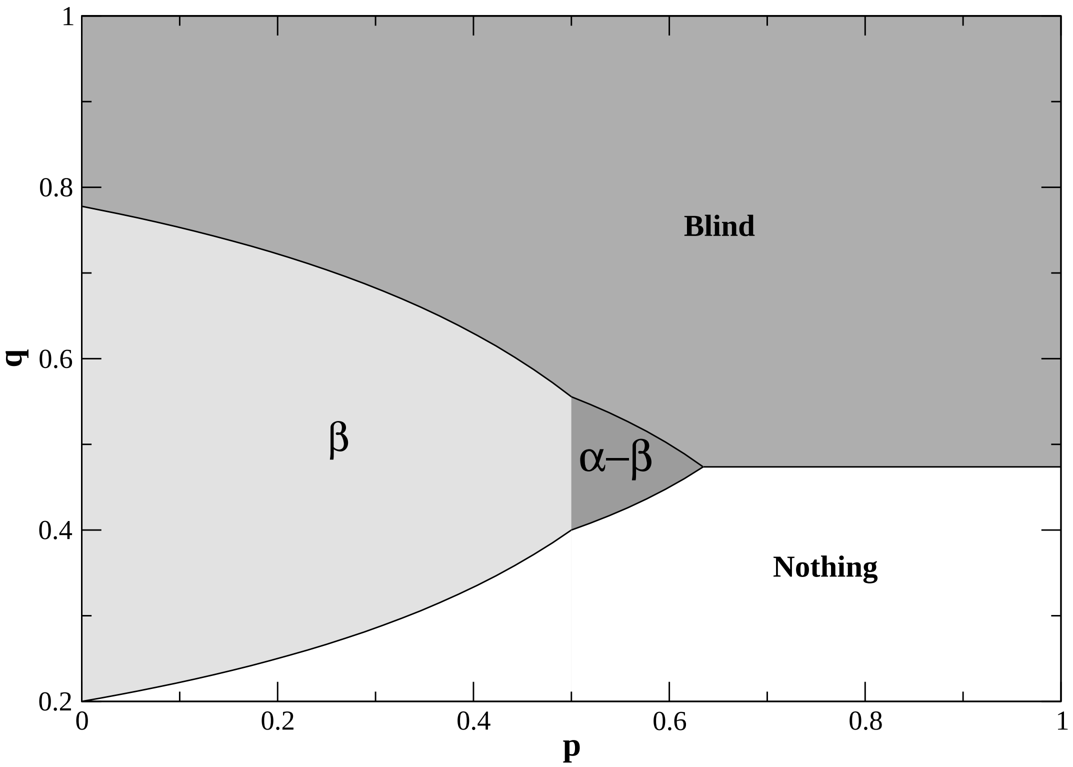


Figure 7.2: *Agents are computing to refine their values. For different values of p and q, different strategies for agent β are the best response to agent α's dominant strategy. If $\frac{1}{2} < p < \frac{33}{52}$ and $\frac{2p-3}{10p-10} < q < \frac{7p-6}{9p-9}$ agent β is best off computing one step on agent α's problem.*

Theorem 16. *If agents have anytime algorithms, costly computing and their performance profiles are not deterministic then in ascending and Vickrey auctions strategic deliberation may occur in equilibrium.*[3]

Theorem 16 only is true if the performance profiles are not deterministic. Agents compute on each others' problems in order to remove uncertainty as to their competitors possible values. If all agents have deterministic performance profiles then all agents know what the results of computing are with certainty.

Lemma 3. *If agents have anytime algorithms, costly computing, and their performance profiles are deterministic, then in an ascending or Vickrey auction no strategic*

[3]This result holds whether agents are computing to improve or refine their values, and does not depend on an independence of performance profiles assumption.

deliberation occurs. This result holds whether agents compute to improve or refine their values.

Proof. If agents compute to refine their values, then, if their performance profiles are deterministic, agents do not need to compute. Their performance profiles tell the agents with certainty what their values are, without having to compute and thus incur a cost. Trivially, since agents do not compute, there is no strategic deliberation.

Assume agents compute to improve their values. Since all performance profiles and cost functions are common knowledge each agent is able to determine what each agent's value function and cost function is. If agent j computes on agent i's value problem, it does not obtain any new information, but does incur a cost, thus reducing its utility. Therefore, no strategic deliberation occurs. $\square$

Even though there is no strategic deliberation, weak strategic deliberation can still be a problem.

Lemma 4. *If agents have anytime algorithms, costly computing, and their performance profiles are deterministic, then in an ascending or Vickrey auction weak strategic deliberation can occur in equilibrium. This result holds whether agents compute to improve or refine their values.*

Proof. Assume there are two agents, α and β, competing in either an ascending or Vickrey auction, with deterministic performance profiles PP_α and PP_β. Assume that the cost functions of the agents are common knowledge. Using the deterministic performance profiles, each agent it able to determine

$$t^*_\alpha = \arg\max_{t_\alpha} [v_\alpha(f_\alpha(t_\alpha)) - \text{cost}_\alpha(\langle 0, \ldots, 0, t_\alpha, 0, \ldots, 0\rangle)]$$

and

$$t^*_\beta = \arg\max_{t_\beta} [v_\beta(f_\beta(t_\beta)) - \text{cost}_\beta(\langle 0, \ldots, 0, t_\beta, 0, \ldots, 0\rangle)]$$

In a Vickrey auction, agent i will submit a bid equal to $v_i(f_i(t_i))$, and in an ascending auction an agent will drop out when the price reaches $v_i(f_i(t_i))$. If agent β knows (from PP_α) that

$$v_\alpha(f_\alpha(t_\alpha)) > v_\beta(f_\beta(t_\beta))$$

then it will not compute because it is certain to lose to auction and thus have utility

$$u_\beta = -\mathrm{cost}_\beta(\langle 0, \ldots, 0, t_\beta, 0, \ldots, 0\rangle)$$

Since agent β's utility from not computing is 0, its best strategy is to not participate. That is, agent β determines its strategy using information from agent α's performance profile. In other words, weak strategic deliberation can occur. □

Combining Lemmas 3 and 4 leads to Theorem 17

Theorem 17. *If agents have anytime algorithms, costly computing, and their performance profiles are deterministic, then in an ascending or Vickrey auction weak strategic deliberation can occur in equilibrium but strategic deliberation does not. This result holds whether agents compute to improve or refine their values.*

7.2.3 Generalized Vickrey Auction

The generalized Vickrey auction (GVA) inherits some of the properties of the single item Vickrey auction. For rational agents, the dominant bidding strategy is to submit bids which are equal to the agents' true values. For agents with bounded resources for determining values, properties from the single item Vickrey auction also are prevalent in the GVA. In particular, if agents have costly computing resources then strategic deliberation may occur in equilibrium.

Theorem 18. *If agents have anytime algorithms, costly computing and their performance profiles are not deterministic, then in the generalized Vickrey auction strategic deliberation can occur in equilibrium. This result holds whether agents are computing to improve of refine their values. This result does not depend on an assumption about independence of performance profiles.*

Proof. From Theorem 16 it is known that in a single item Vickrey auction with costly computing strategic deliberation can occur in equilibrium. Since the single item Vickrey auction is a special case of the generalized Vickrey auction, it can be concluded that strategic deliberation can also occur in the generalized Vickrey auction. □

There are additional problems faced by agents participating in the GVA. Agents have multiple valuation problems of their own. If there are M items then there are

2^M bundles of items in which the agents may place bids. The agents may not have enough computing resources to be able to compute values for each bundle. Instead they must decide on which bundles to *focus their attention.* However, this can not be done in a greedy fashion. It does not work for each agent to simply compute the valuations for bundles that will have the highest utility to the agent in isolation. The agents must take into consideration what bundles other agents will express interest in. Even if agents are not in direct competition for the same bundles of items, they may be interested in overlapping bundles, that is bundles that share some of items. By deliberating on other agents' problems, an agent may be able to determine whether it is likely to win a bundle of items or not. It can then focus its attention on deliberating on the valuation problems for bundles in which it is more likely to be able to win.

In the single item Vickrey auction, if agents had free but limited computing resources then there was no incentive for them to compute on any other problem other than on their own value problem (Theorem 15). However, due to the relations between bundles of goods, strategic deliberation may occur as agents partially evaluate competitors' bundles in order to determine their chances of being allocated certain bundles

Theorem 19. *If agents have anytime algorithms, free but limited computing and their performance profiles are not deterministic then in a generalized Vickrey auction strategic deliberation may occur in equilibrium. This result does not depend on an assumption about independence of performance profiles.*

Proof. By example. We will use the performance profile tree representation in this proof for illustrative purposes. Other performance profile representations have the same property. Let there be two agents, α and β, and three items, $g1$, $g2$, and $g3$. The performance profiles for the agents' valuation problems are in Figure 7.3. Valuation problems that remain zero no matter how much computing is allocated to them are not shown.

Assume that both agents' deadlines, d_α and d_β, occur at $t = 3$. Assume, also, that the auction closes at $T = 3$. Agent β has a dominant strategy. In the first time step it computes on the valuation for $\{g3\}$. If $v_\beta^{g3}(1) = 1.0$ then it computes two more time step on $\{g3\}$ to obtain a valuation of 22.0. At time T it would bid its true valuation for all bundles. If $v_\beta^{g3}(1) = 0.0$ then it performs two computing steps on $\{g2, g3\}$ and obtains a valuation $v_\beta^{\{g2,g3\}}(2) = 7.0$. At time T it would bid its true valuation.

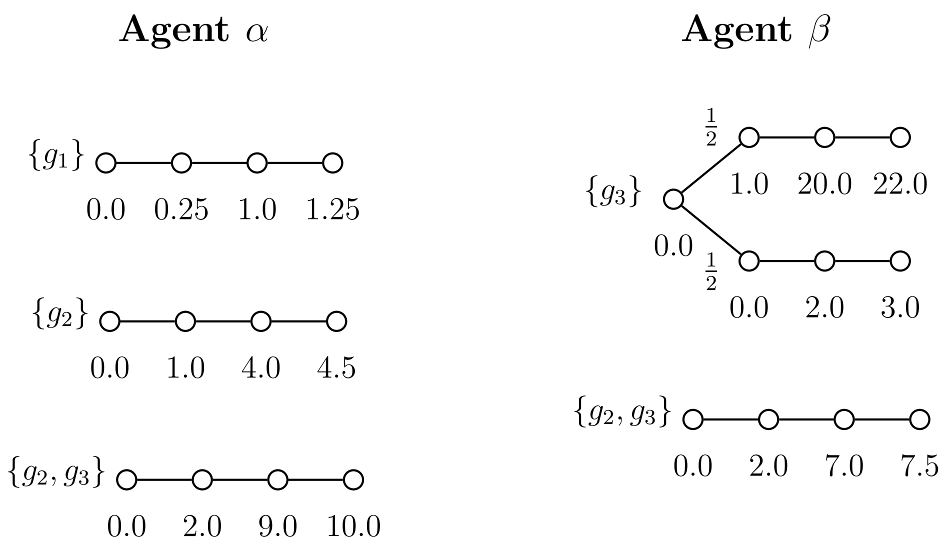


Figure 7.3: *Performance profiles for agents α and β. There is uncertainty in agent β's valuation for bundle $\{g3\}$.*

Agent α's best response is to compute the first time step on agent β's valuation problem for $\{g3\}$ (i.e. perform strategic deliberation). If after one time step, agent α determines that $v_\beta^{\{g3\}}(1) = 1.0$ then agent α computes two time steps on its own valuation problem for $\{g1\}$. Otherwise it computes two steps on its own valuation problem for $\{g2, g3\}$.

Agent α realizes that if after one computing step, $v_\beta^{\{g3\}}(1) = 1.0$ then agent β will continue computing on the problem, obtain a valuation of 22.0, and include it in its bid. Since in any optimal allocation, the item $g3$ will be awarded to agent β, agent α could never be awarded any bundle that contains $g3$. Therefore it is better off computing on the valuation problem for $\{g1\}$ and bidding its true valuation. However, if $v_\beta^{\{g3\}}(1) = 0.0$ then agent α knows that it can win the bundle $\{g2, g3\}$ and so computes two steps on the valuation problem.

The expected utility for each agent is

$$\begin{aligned} E[u_\alpha] &= \frac{1}{2}(1.0 - 0.0) + \frac{1}{2}(9.0 - 7.0) = 1.5 \\ E[u_\beta] &= \frac{1}{2}(22.0 - 0.0) + \frac{1}{2}(0) = 11.0 \end{aligned}$$

□

In the single item Vickrey auction, if the performance profiles were deterministic and computing was free but limited, then agents would not even partake of any weak strategic deliberation. However, in the GVA, this property no longer holds. Agents may at times have incentive to base their computing choices on the deterministic performance profiles of their competitors.

Theorem 20. *If agents have anytime algorithms, free but limited computing and their performance profiles are all deterministic, then in a generalized Vickrey auction, weak strategic deliberation may occur in equilibrium. This result does not depend on any assumption about independence of performance profiles.*

Proof. By example. Let there be two agents, α and β and 3 goods, g_1, g_2, g_3. Assume that the performance profiles of the agents are defined as follows.

$$f_b^\alpha(t) = \begin{cases} 0 & \text{if } t = 0 \\ 4 & \text{if } t > 0 \text{ and } b = \{g_1\} \\ 1 & \text{if } t > 0 \text{ and } b = \{g_2\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_3\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_1, g_2\} \\ 6 & \text{if } t > 0 \text{ and } b = \{g_1, g_3\} \\ 0.5 & \text{if } t > 0 \text{ and } b = \{g_2, g_3\} \\ 2 & \text{if } t > 0 \text{ and } b = \{g_1, g_2, g_3\} \end{cases} \qquad f_b^\beta(t) = \begin{cases} 0 & \text{if } t = 0 \\ 0 & \text{if } t > 0 \text{ and } b = \{g_1\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_2\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_3\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_1, g_2\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_1, g_3\} \\ 12 & \text{if } t > 0 \text{ and } b = \{g_2, g_3\} \\ 0 & \text{if } t > 0 \text{ and } b = \{g_1, g_2, g_3\} \end{cases}$$

Assume that the deadline of both agents is at time $t = 1$. The agents must decide how to use their single computing step. Agent β has a dominant strategy which is to compute one step on bundle $\{g_2, g_3\}$ and submit a bid equal to the computed value. If agent α ignored agent β then it would want to compute on bundle $\{g_1, g_3\}$ and submit a bid equal to 6. However, agent α would not be awarded this bundle since it shares an item (g_3) with agent β's desired bundle which will be allocated to agent β. Therefore, agent α is best off using information from agent β's performance profile and computing one step on the value problem for bundle $\{g_1\}$ and then bidding its computed value. $\square$

7.3 Algorithm Choice

So far, a basic assumption has been that each agent only has one algorithm for each problem in which it is interested. However, one can imagine situations where agents have access to several different algorithms. In such settings, agents are faced not only with the decision of how to allocate their computing resources across problems, but must also decide how to allocate their computing resources between algorithms for the same problem. It is possible that the choice of algorithm for a problem may depend on what algorithms other agents have chosen and what values that the agents have obtained.

Clearly, in situations where agents incur a cost for computing, strategic deliberation can occur when agents have multiple algorithms for the same problem. However, in settings where agents have limited computing resources, if agents have multiple algorithms, then it is possible that strategic deliberation can occur.

Consider the following example. Assume there are two agents, α and β, participating in a Vickrey auction who have limited computing resources with deadlines D_α and D_β and are computing to improve their values. Agent α has two algorithms for its valuation problem. Algorithm 1 performs well on some types of problems, while Algorithm 2 performs well on other types of problems. However, for the first k steps, both algorithms perform identically (returning a value of 0) so it is impossible for the agent to determine which is the best algorithm to use until after computing k steps. Agent β has only one algorithm, but it has the property that it always performs well on problems that Algorithm 1 performs well on, and performs poorly on problems that Algorithm 2 performs well on. An agent using this algorithm of agent β learns after only $l < k$ steps whether the algorithm is performing well or not. Additionally, to further simplify the problem, we assume that the agents have very different uses for the item and so the solution found by the algorithm(s) of one agent is not usable by the other agent. Instead, the results from one agent's algorithm can only be used as a signal about the problem instance being computed on, and not as a solution for the other agent.

If agent α only computes using its own algorithms then, in the worst case, it will have to compute $k + 1$ steps before it computes a value which is greater than 0. However, by first computing l steps using agent β's algorithm, agent α can determine the type of problem it is facing. It can, therefore, choose the appropriate algorithm for the problem, resulting in a higher expected utility. Using a similar example, it is possible to show that strategic deliberation can also occur if agents computed to refine their values. We obtain the following theorem.

Theorem 21. *Assume agents have limited computing resources and compute to improve or refine their values. If agents have a choice between multiple algorithms for solving their valuation problems, then strategic deliberation may occur in equilibrium.*

7.4 (Non)strategic Equivalence of the Vickrey and Ascending Auction

In this chapter we have shown that interesting strategic behavior, namely strategic deliberation, can occur in both the Vickrey and the ascending auctions. In the classical, fully rational setting, the Vickrey auction and the ascending auction are strategically equivalent, in that it is possible to construct an isomorphism between the strategy spaces of the agents participating in the auctions.

If the agents have computational limitations then it is unlikely that this isomorphism still exists. Due to the structure of the auctions, the agents are able to deduce different information about each other as the auction runs. In particular, in the ascending auction an agent may be able to deduce information about an competitor agent without having to do any computing, while in the Vickrey auction, to get the same information, the agent would have to strategically compute.

Theorem 22. *The ascending and Vickrey auctions are not strategically equivalent if agents are computationally limited. The occurrence of strategic deliberation in the Vickrey auction does not imply that there will be strategic deliberation in an ascending auction.*[4]

Proof. We prove this theorem by providing an example of performance profiles and cost functions where strategic deliberation occurs in equilibrium in the Vickrey auction, but does not occur in equilibrium in an ascending auction.

Assume that there are two agents, agent 1 and agent 2, with performance profiles as shown in Figure 7.4. Assume that the cost functions of the agents' are

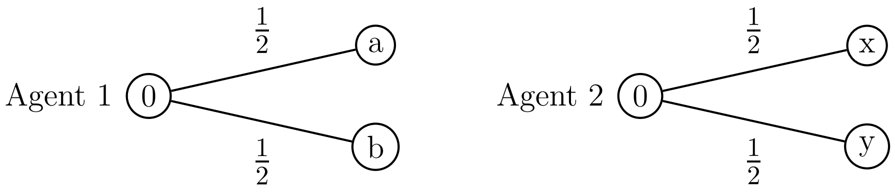
$$\text{cost}_1((t_1, t_2)) = 0$$

and

$$\text{cost}_2((t_1, t_2)) = t_1 + 3t_2.$$

In both the Vickrey and ascending auctions, agent 1 has a (weakly) dominant strategy which is to compute only on its own problem to obtain value a or b and then bid as though it was a fully rational agent participating in the auction.

[4]This result does not rely on any independence assumptions.



$$a > x > y > b$$

Figure 7.4: *Performance profiles for agent 1 and agent 2. In an ascending auction, agent 2 would never compute on the problem of agent 1 since it can obtain this information in a timely manner through the auction mechanism.*

In an ascending auction, agent 2 has no incentive to compute on the problem of agent 1. Instead, agent 2 can wait until the price has risen to $p = b$. If the value of agent 1 is b then agent 1 will withdraw from the auction, leaving the item for agent 2 to get it at price b. If, however, at price $b + \epsilon$ agent 1 is still in the auction, then agent 2 can deduce that agent 1 values the item at a which is higher than what agent 2 will ever value it. Therefore, agent 2 still has no incentive to pay a cost to compute on the problem of agent 1.

In a Vickrey auction, however, agent 2 does not have the opportunity to learn the value of agent 1 for free. Instead, it has incentive to compute first on the problem of agent 1, in order to determine if the value of agent 1 is high, under the following conditions:

$$0 < -1 + \frac{1}{2}\left[\frac{1}{2}(x-b) + \frac{1}{2}(y-b) - 3\right]$$

and

$$-3 + \frac{1}{4}\left[(x-b) + (y-b)\right] < -1 + \frac{1}{2}\left[\frac{1}{2}(x-b) + \frac{1}{2}(y-b) - 3\right].$$

These two conditions hold when the values of a, b, x, y are as follows:

$$5 < x \leq 10,\ 10 - x < y < x,\ 0 < b < \frac{1}{2}(x + y - 10),\ a > x$$

or

$$x > 10,\ 0 < y < 10 - x,\ 0 < b < y,\ a > x$$

or

$$x > 10,\ x - 10 \leq y < x,\ 0 < b < \frac{1}{2}(x + y - 10),\ a > x.$$

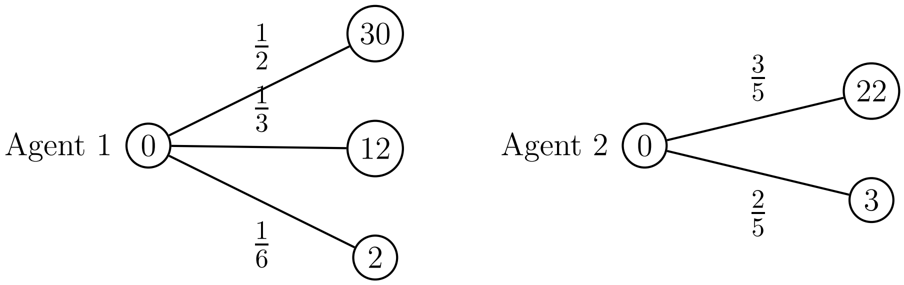


Figure 7.5: *Performance profiles for agent 1 and agent 2. While strategic deliberation can occur in the ascending auction, it does not imply that the same instances (performance profiles and cost functions) will lead to strategic deliberation in the Vickrey auction.*

□

The ascending auction is not strategically equivalent to the Vickrey auction. However, the counterexample presented in the above theorem seems to suggest that, with respect to strategic deliberation, the ascending auction is a more robust mechanism in that strategic deliberation occurs less often in it compared to the Vickrey auction. However, as we show in the next theorem, this is not true. There exist situations where strategic deliberation occurs in the ascending auction, but not the in the Vickrey auction.

Theorem 23. *The occurrence of strategic deliberation in the ascending auction does not imply that strategic deliberation will also occur in the Vickrey auction.*

Proof. Assume that there are two agents, agent 1 and agent 2, with performance profiles as shown in Figure 7.5. Assume that the cost functions are as follows:

$$\mathrm{cost}_1((t_1, t_2)) = 0$$

and

$$\mathrm{cost}_2((t_1, t_2)) = t_1 + 2t_2.$$

Agent 1 has a dominant strategy in both the ascending and Vickrey auctions. It is best off computing only on its own problem and bidding using the fully rational agent bidding strategy given its computed value.

The strategy of agent 2 depends on which auction it is participating in. In the ascending auction, agent 2 is best off not doing anything until the price increases to $p = 2$. If agent 1 drops out of the auction at this point, then agent 2 will be allocated the item. If however, agent 1 does not withdraw from the auction, then agent 2 learns that agent 1 has either 30 or 12 as its value. Using Bayes rule to update its beliefs about what value agent 1 has computed, agent 2 places probability $\frac{3}{5}$ on the value being 30 and and $\frac{2}{5}$ on the value being 12. Agent 2 is then best off computing one step on the problem of agent 1 and then computing on its own problem only if the value of agent 1 is 12 (with expected utility of 0.6), as opposed to doing nothing (expected utility of 0) or computing only on its own problem (expected utility of 0.4).

In the Vickrey auction, agent 2 is best off computing only on its own problem and submitting a bid equal to its computed value. If it follows such a strategy it has an expected utility equal to 3.067, as opposed to doing nothing (expected utility of 0) or computing on the problem of agent 1's first in order to learn whether agent 1 has a value of 30 (expected utility of 2.067). $\square$

The two previous theorems show that with respect to strategic deliberation, it is not possible to claim that either the ascending or Vickrey auction is more robust than the other. While strategic deliberation is prevalent in both auction mechanisms, the existence of strategic deliberation, given an instance defined by the agents, does not imply that strategic deliberation will also occur in the other auction mechanism.

7.5 Experiments

In this chapter we have studied the occurrence of strategic deliberation in different auction mechanisms. While we have shown that for all auctions of interest (the ascending auction, the Vickrey auction, and the first-price, sealed-bid auction) strategic deliberation was prevalent, depending on the performance profiles of the agents. However, in showing the existence of strategic deliberation, we relied on hand crafted examples, and it was not clear whether strategic deliberation was an actual issue or an artifice of the analysis.

Using performance profile trees as our deliberation control procedure, we conducted a series of experiments to explore the effect that limited deliberation resources has on agents' strategies. After generating performance profiles using data from different real-world application domains, we used Gambit [73], a popular solver for finding game-theoretic equilibria, to find and categorize all Nash deliberation equilibria.

7.5.1 Application Scenarios

We conducted our experiments using data from two different scenarios; vehicle routing and single-machine manufacturing scheduling. We treated the domain problem solvers as black boxes. We outline the main properties of each domain, but the full details are found in Chapter 3.

Vehicle Routing

In the real-world vehicle routing problem (VRP) in question, a dispatch center is responsible for a certain set of tasks (deliveries) and has a certain set of resources (trucks) to take care of them. Each truck has a depot, and each delivery has a pickup location and a drop-off location. The dispatch center's problem is to minimize transportation cost (driven distance) while still making all of its deliveries and honoring the following constraints:

- each vehicle has to begin and end its tour at its depot, and
- each vehicle has a maximum load weight and maximum load volume constraint.
- each delivery has to be included in the route of some vehicle.

To generate data for our experiments, an iterative improvement algorithm was used for solving the VRP. The problem instances were generated using real-world data collected from a dispatch center that was responsible for 15 trucks and 300 deliveries. We generated two independent sets by randomly dividing the deliveries in half. To generate 1000 instances for each set used to build PPTrees, we randomly selected (with replacement) 60 deliveries.

Manufacturing Scheduling

The second domain is a single-machine manufacturing scheduling problem with sequence-dependent setup times on the machines, where the agent's objective is to minimize

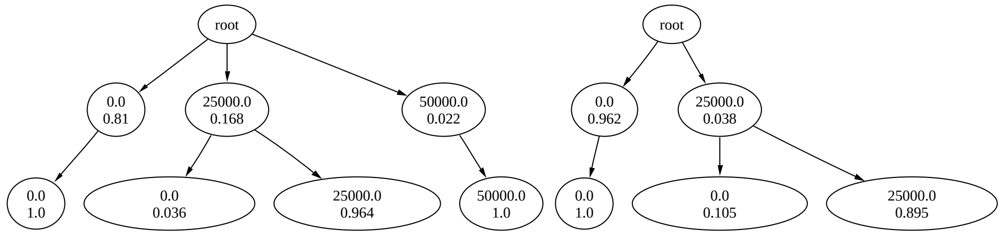


Figure 7.6: *Performance profile trees for the scheduling domain Each node in a tree contains two numbers. The first number is the solution quality and the second number is the probability of reaching the node, given that its parent was reached.*

weighted tardiness,

$$\sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(f_j - d_j, 0),$$

where T_j is the tardiness of job j, and w_j, f_j, d_j are the weight, finish time, and due-date of job j.

In our experiments, we used a state-of-the-art scheduler developed by others [19] as the domain problem solver. It is an iterative improvement algorithm that uses a scheduling algorithm called Heuristic Biased Stochastic Sampling [16]. We treated the domain problem solver as a black box without any modifications.

The problem instances were generated according to the standard benchmark outlined in [68]. The due-date tightness factor was set to 0.3 and the due-date range factor was set to 0.25. The setup time severity was set to 0.25. These parameter values are the ones used in standard benchmarks [68]. Each instance consisted of 100 jobs to be scheduled. We generated two independent sets of instances by using different random number seeds.

7.5.2 Generating Performance Profiles

We used independent sets of 1000 instances per set from both application domains to generate performance profile trees for the agents. Due to limitations in Gambit [73], the software used for the game-theoretic analysis, we were required to use coarse discretizations on both solution quality and time steps. Each tree had depth two (an agent could compute for two time steps on a single problem). The solution quality

Figure 7.7: *Performance profile trees for the trucking domain . Each node in a tree contains two numbers. The first number is the solution quality and the second number is the probability of reaching the node, given that its parent was reached.*

was coarsely discretized in order to reduce the branching factor of the PPTrees so that the problems were feasible for Gambit. Figures 7.6 and 7.7 show sample performance profile trees produced when the solution quality was uniformly discretized into buckets of size 25000 for scheduling and size 10000000 for the trucking domain.[5] Later in this chapter we discuss the implications caused by using such a coarse discretization, whether we believe the discretization influenced the observed results in a marked way, and whether we believe there is any approach in which more refined performance profiles can be used for such analysis.

7.5.3 Cost Functions

In our model there are costs associated with deliberating. The costs for agent i are represented by a function $c_i(t_1, t_2) = K_i^1 t_1 + K_i^2 t_2$ where t_j is the amount of time agent i deliberated on the valuation problem of agent j, and $K_i^1, K_i^2 \geq 0$ are predefined constants. The cost functions of the agents are either *symmetric* or *asymmetric*.

Definition 41. *A cost function $c_i(t_1, t_2) = K_i^1 t_1 + K_i^2 t_2$ is* symmetric *if $K_i^1 = K_i^2$. A cost function $c_i(t_1, t_2) = K_i^1 t_1 + K_i^2 t_2$ is* asymmetric *if $K_i^1 \neq K_i^2$.*

Symmetric costs naturally model situations where agents *compute* on problems in order to determine valuations. For example, an agent may pay some amount K for each CPU cycle used running an algorithm. Asymmetric cost functions naturally

[5]We experimented with other feasible uniform discretizations. The results were all similar to the results reported in this paper. Therefore, due to space considerations we do not include them.

model information gathering scenarios as it is not unreasonable that there are different costs associated with gathering information from different sources.

7.5.4 The Reverse Vickrey Auction

We provide a motivating example. A company wishes to contract out a set of 100 tasks as a sole-source contract, i.e., the entire set is allocated to one manufacturer. The company runs a reverse Vickrey auction to allocate the set to one of two possible manufacturers. Each task in the set has a deadline, and a task-specific penalty for each unit of time that the task is late. The manufacturer has to pay the total weighted tardiness to the company as a penalty. If agent i wins the auction, has obtained a valuation, v_i, by deliberating, and has incurred cost c_i while doing so, then its utility is

$$u_i = x - v_i - c_i$$

where $x = \min(b_j, R)$ where b_j is the second lowest bid (if any) and R is the company's reserve price, i.e., the company's maximum willingness to pay to get the task set manufactured. If agent i does not win, its utility is

$$u_i = -c_i.$$

In our experiments there were two agents, agent 1 and agent 2. Each agent had different valuation problems and thus different performance profiles. The performance profile trees were common knowledge. At each time step an agent had a choice of actions available to it: it could deliberate on its own valuation problem, on its competitor's problem, or it could choose not to deliberate. Once both agents stopped deliberating (after at most two time steps), they submitted bids to the auctioneer.

7.5.5 Representing Equilibria

We denote a (deliberation) strategy by $A_{i_{B_i}}$ where A_i is the deliberation action taken by agent i in the first step, and B_i is the *set* of deliberation actions taken by agent i in the second step, *conditional on the results observed after the first step.* We do not denote the bidding actions since agents are motivated to always submit their value obtained by deliberating.

We represent the deliberation actions as follows:

- N is the action of not deliberating,
- O is the action of deliberating for one step on its own problem, and,
- Z is the action of deliberating for one step on its opponent's problem.

Agents may play mixed strategies, that is, they randomize between multiple actions. For example, at a given time step, an agent may decide to randomize between not deliberating (N) and deliberating on its own problem (O). We denote this by $m(O, N)$.

Consider the following example. Assume that there exists an equilibria where agent 1 randomizes between deliberating on its own problem and the problem of agent 2, and then either tops deliberating of deliberates for one step on its own problem in the second stage. Agent 2 randomizes between all three actions, and then stops. This is denoted by

$$(m(O, Z)_{(O,N)}, m(O, Z, N)_N).$$

7.5.6 Symmetric Cost Functions

We first investigate what happens if agents have symmetric, but potentially different, cost functions. For both application domains we ran a series of reverse Vickrey auctions, varying the reserve prices and cost functions of the agents. Table 1 presents a complete taxonomy of all Nash (deliberation) equilibria in the scheduling domain, when the reserve price was taken from the set $\{25000, 50000, 100000\}$. These price choices span the interesting range for the scheduling domain. Reserve prices cap the potential utility of the agents. At the bottom end the agents' utilities are capped so that there is a potential that even if both agents deliberate on their own problems, neither agent will win the auction. At the high end the reserve price has little influence, since the cap is set high enough such that as long as agents deliberate, an agent will win the auction. The cost functions of the agents were of the form $c_i(t_1, t_2) = K_i(t_1 + t_2)$ for $K_i \in \{10, 100, 12500, 25000, 50000\}$. These value choices span the interesting range for the scheduling domain: at the bottom end the values are low enough so that deliberation on problems is a potential strategy, while at the top end there is the potential that the deliberating cost is higher than any possible utility that could be achieved.[6]

[6]We experimented with other reserve prices and cost functions. The results of these experiments were similar to the ones presented in the paper.

There are several observations. First, we did not observe any equilibria where strategic deliberation occurred. Second, the cost functions of the agents influenced their strategic behavior. For example, when $K_1 = 100$ then agent 1 always spent the first time step deliberating on its own problem. However, the action it took in the second time step depended on the actions of agent 2. Third, multiple and mixed equilibria appeared when the costs were high enough (12500), and disappeared when the costs became too high (compared to the reserve price). Finally, reserve prices influenced the deliberation equilibria. For example, the equilibria that occurred when $c_1(t_1, t_2) = 50000 \cdot (t_1 + t_2)$ and $c_2(t_1, t_2) = 25000 \cdot (t_1 + t_2)$ changed according to the reserve price. When the reserve price was $R = 25000$ then there was a single equilibrium: neither agent deliberated on any problem. When $R = 50000$, there was a different, single equilibrium. Finally, when $R = 100000$ there were multiple equilibria: two pure equilibria where one agent deliberated on its own problem for one time step while the other did not deliberate at all, and a mixed equilibrium where both agents randomized between not deliberating and deliberating on their own problem in the first step.

7.5.7 Asymmetric Cost Functions

We conducted experiments with asymmetric cost functions. Tables 2 and 3 present results from the vehicle routing domain, where the reserve price, R, was set to $5 \cdot 10^7$, *i.e.*, high enough so that it did not introduce additional strategic considerations, and where $K_i^j \in \{1000, 2.5 \cdot 10^6, 5 \cdot 10^6, 1 \cdot 10^7\}$. These values were chosen so as to span the interesting range for the routing domain.

We only present results where cost functions had the form $c_i(t_1, t_2) = \sum_{j=1}^{2} K_i^j \cdot t_j$ where $K_i^i \geq K_i^k$. If the cost of deliberating on a competitor's valuation problem is higher than the cost of deliberating on the agent's own problem, then clearly no strategic deliberation will occur in equilibrium.

Several interesting phenomena were observed. First, when the cost of deliberating on the other agent's valuation problem was low, strategic deliberation occurred in equilibrium. This is illustrated by Table 2. Almost every mixed Nash equilibrium involved agent 1 deliberating on the problem of agent 2 with positive probability. However, when $K_1^2 = 2.5 \cdot 10^6$, then agent 1 never deliberated on the valuation problem of agent 2. The same behavior was observed for agent 2 (see Table 3). Second, if K_i^j was high enough then no strategic deliberation occurred, irrespective of the value of

K_i^i (Table 3). Third, the other agent's actions also play an important role. That is, agents' do not have dominant strategies which are parameterized by their own cost functions, instead, complex strategic behavior occurs. This can be observed from Table 3.[7]

How much asymmetry is required in order for strategic deliberation to occur? The experiments reported in Tables 2 and 3 involved gross asymmetries. It was not clear whether large differences in deliberating costs are required in order to force strategic deliberation, or whether just an ϵ-difference in the cost constants could produce interesting strategic behavior. Using the scheduling domain as our sample application, we fixed the cost function of agent 2 as $c_2(t_1, t_2) = 100 \cdot (t_1 + t_2)$ and set $c_1(t_1, t_2) = 100 \cdot t_1 + K_1^2 \cdot t_2$. We decreased K_1^2 by increments of 0.5. When $K_1^2 \geq 97.0$ the unique Nash (deliberation) equilibrium was (O_N, O_N). However, for $K_1^2 < 97.0$ the unique Nash (deliberation) equilibrium was $(Z_{(O,N)}, O_N)$. It appears as though we can conclude that gross asymmetries are not required in order for strategic deliberation to occur, however there should be more than an ϵ-difference between the costs associated with deliberating on different problems.

7.5.8 Discussion

From studying the results from both the symmetric and asymmetric cost function experiments, there are some interesting points that arise. First, in order for strategic deliberation to occur, there must be a certain amount of asymmetry between the problems of the agents. This asymmetry may arise due to differences in cost functions, or the relative difficulty of different agents' valuation problems. For example, in a vehicle routing problem, it might be very easy to find a short route in one problem instance, yet not be so easy in a different problem instance. Even if there is no strategic deliberation (as in the symmetric cost function setting) agents' optimal strategies depend on what strategies the other agents are following: dominant strategy equilibria do not always exist. For example, there may exist situations where an agent may decide to not compute on any problem in order to avoid incurring a cost, given that a competitor intends to solve its own valuation problem.

Second, the (reverse) Vickrey auction is a very simple auction mechanism and when agents are fully rational, they have dominant strategies. In spite of its simplicity,

[7]Using PPTrees generated from data from the scheduling domain, we repeated the experiments and observed similar equilibria. Due to space limitations we do not present the results in this paper.

in recent work it has been shown that the information structures which appear in the Vickrey auction and drive the theoretical strategic deliberation results, also appear in many multi-stage auctions such as the English auction and variants [65]. Thus, while our experiments focus only on one auction type, we believe that the conclusions are applicable across a wide range of auction mechanisms.

A criticism of these experiments is that in order to make the games small enough so that the software, Gambit, could actually load and solve them, we had to use a very coarse discretization of both time and solution quality. This meant that a lot of information about the algorithms' performance was lost as potentially very different runs of the algorithms were treated identically. We do not believe that the observed strategic deliberation was an artifice caused by the discretization choices. If anything, by using a finer discretization we believe that there would have been increased asymmetry in the performance profile trees, which, we know from the theoretical analysis, can lead to the occurrence of strategic deliberation. We did not observe any strategic deliberation in the symmetric cost function experiments, and this may be due to the coarse discretization. As just mentioned, the discretization removed a lot of useful information from the performance profiles, particularly removing possible asymmetries. Therefore, we are not willing to conclude that strategic deliberation will never occur in equilibrium when there are symmetric cost functions.

Gambit is a state-of-the-art general game solver, and unless there is a significant breakthrough in the algorithms for finding equilibrium, we believe that we will always be faced with the problem of having to work with coarse discretizations in the performance profile trees. However, there are a few glimmers of hope. First, for some sets of performance profile trees there may be enough structure so that the game description could be reduced, making the equilibrium finding problem easier. Second, it may be possible to design auction mechanisms that explicitly take into account the computational limitations of the agents and guaranteeing good equilibrium behavior. This would eliminate the need of using such tools as Gambit to find all equilibria, since the mechanism would provide the correct incentives for the agents to play a particular equilibrium.

7.6 Summary

Auctions provide efficient and distributed ways of allocating goods and tasks among agents. For rational agents, bidding strategies have been well studied in the game theory literature. However, not all agents are fully rational. Instead they may have computing limitations which curtail their ability to compute valuations for the items being auctioned. This adds another dimension to the agents' strategies as they have to determine not only the best bid to submit, but how to use their computing resources in order to determine their valuations and gain information about the valuations of the other agents participating in the auction.

We investigated agents strategic behavior under different computational settings and auction mechanisms. We showed that in most standard auction settings, in equilibrium agents are willing to use their deliberation resources in order to compute on valuation problems of competitors, even if this means that their knowledge about their own true valuation is lessened. In particular, we showed that strong strategic computing does not depend on the type of anytime algorithm, nor in general does it depend on the type of performance profile. The critical property is whether or not there is uncertainty as to what values will result from further computing. In particular we have shown that if all agents' performance profiles are deterministic then

- No strong strategic computing occurs. This is true for all auctions studied (first-price sealed-bid, Vickrey, ascending and GVA), for both types of anytime algorithms (contract and interruptible) and for both costly and limited computing.
- If computing is costly, then weak strategic computing may occur in all auctions.
- If computing is limited, weak strategic computing may occur in the first-price sealed-bid auction and GVA. Weak strategic computing will not occur in the ascending and Vickrey auctions. This is true for both contract and interruptible algorithms.

If agents' performance profiles are not deterministic (there is uncertainty as to what values will result from computing) then the following results were derived. This is true for both contract and interruptible algorithms.

- If computing is limited then in the ascending and Vickrey auction neither strong nor weak computing occurs in equilibrium. This is true for both contract and interruptible algorithms.

- If computing is limited then in the first-price sealed-bid and GVA strong strategic computing may occur in equilibrium. This is true for both contract and interruptible algorithms.

- If computing is costly, strong strategic computing may occur in all auctions studied. This is true for both contract and interruptible algorithms.

These results are summarized in Tables 7.5 and 7.6.

We also conducted a series of experiments to study whether strategic deliberation occurs in practice or was merely an artifice of the analysis. We ran reverse Vickrey auctions with bidding agents who had limited deliberation resources and were provided with fully normative deliberation control methods. We observed no strategic deliberation when agents had *symmetric* cost functions.[8] However, if the cost functions of the agents were *asymmetric* then strategic deliberation occurred in equilibrium. This supports our position that when designing electronic markets and other multiagent systems, it is of both theoretical and practical importance to consider the deliberation actions of agents.

[8]This is not proof that it never occurs in practice. If more deliberation actions were allowed, then we may observe it experimentally.

R=25000	100	1000	12500	25000	50000
100	$O_N, O_{(O,N)}$	O_N, O_N	$O_{(O,N)}, N$	$O_{(O,N)}, N$	$O_{(O,N)}, N$
1000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(N,O)_N$ O_N, N	O_N, N	O_N, N
12500	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(N,O)_N$ O_N, N	O_N, N	O_N, N
25000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N	N, N	N, N
50000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N	N, N	N, N
R=50000					
100	$O_N, O_{(O,N)}$	O_N, O_N	$O_{(O,N)}, N$	$O_{(O,N)}, N$	$O_{(O,N)}, N$
1000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	O_N, N
12500	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	O_N, N
25000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	O_N, N
50000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N	N, O_N	N, N
R=100000					
100	$O_N, O_{(O,N)}$	O_N, O_N	$O_{(O,N)}, N$	$O_{(O,N)}, N$	$O_{(O,N)}, N$
1000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N
12500	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N
25000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N
50000	$N, O_{(O,N)}$	$N, O_{(O,N)}$	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N

Table 7.2: *All Nash equilibria for the scheduling domain with symmetric cost functions. The rows are different cost functions for agent 1 and the columns are different cost functions for agent 2. Each cell contains all Nash equilibria given the agents' cost functions and the reserve price R.*

(1000,1000)	1000	2.5E6	5E6	1E7
1000	O_N, O_N	O_N, N	O_N, N	O_N, N
2.5E6	N, O_N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,Z)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,Z)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N
5E6	N, O_N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,Z)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,Z)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N
1E7	N, O_N	N, O_N $*\mathbf{m(O,N)_{N}}, \mathbf{m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(O,N)_{N}}, \mathbf{m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $*\mathbf{m(O,N)_{N}}, \mathbf{m(Z,N)_{(O,N)}}$ $*\mathbf{m(O,Z,N)_{(O,N)}}, \mathbf{m(O,Z,N)_{(O,N)}}$ $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N

(1000,2.5E6)				
1000		O_N, N	O_N, N	O_N, N
2.5E6		N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N
5E6		N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N
1E7		N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(O,Z)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N	N, O_N $*\mathbf{m(Z,N)_{(O,N)}}, \mathbf{m(O,N)_{N}}$ O_N, N

Table 7.3: *Nash (deliberation) equilibria for the routing domain with asymmetric cost functions. The reserve price is* $R = 5 \cdot 10^7$. *The cost function of agent 1 is* $c_1 = K_1^1 \cdot t_1 + 1000 \cdot t_2$. *he entry in the upper left hand corner cell of each subtable specifies* (K_1^2, K_2^1) *The values for* K_1^1 *are listed in the rows, and the values for* K_2^2 *are listed in the columns. The subtables* $(1000, 5E6)$ *and* $(1000, 1E7)$ *were identical to subtable* $(1000, 2.5E6)$.

(2.5E6, 1000)	1000	2.5E6	5E6	1E7
2.5E6	N, O_N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N
5E6	N, O_N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ $0_N, N$	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N
1E7	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N	N, O_N $\ast\mathbf{m(O,N)_N, m(Z,N)_{(O,N)}}$ O_N, N

(2.5E6,2.5E6)				
2.5E6		N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N
5E6		N, O_N $m(O,N)_{(O,N)}, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_{(O,N)}, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_{(O,N)}, m(O,N)_N$ O_N, N
1E7		N, O_N $m(O,N)_N, m(O,N)_N$ O_N, N	N, O_N $m(O,N)_N, m(O,N)_N$ N, O_N	N, O_N $m(O,N)_N, m(O,N)_N$ N, O_N

Table 7.4: *Nash (deliberation) equilibria for the routing domain with asymmetric cost functions. The reserve price is* $R = 5 \cdot 10^7$. *The cost function of agent 1 is* $c_1 = K_1^1 \cdot t_1 + 2.5 \cdot 10^6 \cdot t_2$. *he entry in the upper left hand corner cell of each subtable specifies* (K_1^2, K_2^1) *The values for* K_1^1 *are listed in the rows, and the values for* K_2^2 *are listed in the columns. The subtables* $(2.5E6, 5E6)$ *and* $(2.5E6, 1E7)$ *were identical to subtable* $(2.5E6, 2.5E6)$.

	Auction mechanism	Counterspeculation by rational agents?	Strategic computation? Limited computation	**Costly computation**
Single item	1st price sealed bid	yes	weak only	weak only
	Vickrey	no	no	weak only
	Ascending	no	no	weak only
Multiple items	GVA	no	weak only	weak only

Table 7.5: *Strategic deliberation does not occur in equilibrium when agents have deterministic performance profiles. However, agents do not have dominant strategies. Instead, when ever there is a cost associated with computing, an agents strategy will depend on the performance profiles of the other agents (i.e. weak strategic deliberation occurs.)*

	Auction mechanism	Counterspeculation by rational agents?	Strategic computation? Limited computation	**Costly computation**
Single item	1st price sealed bid	yes	yes	yes
	Vickrey	no	no	yes
	Ascending	no	no	yes
Multiple items	GVA	no	yes	yes

Table 7.6: *If there is uncertainty in the results from computing, then strategic deliberation occurs in equilibrium when ever there is a cost associated with computing. Additionally, in the 1st-price sealed-bid auction and the GVA strategic deliberation occurs even when computing is limited by deadlines.*

Chapter 8

Mechanism Design for Computationally Limited Agents

In this chapter we study the problem of designing mechanisms for computationally limited agents. Instead of looking at the properties of mechanisms that were designed for fully rational agents, we ask the question:

> *"Is it possible to design mechanisms that have desirable properties for computationally limited agents?"*

We propose a set of weak, intuitive properties that are desirable for mechanisms designed for such agents. In particular, we propose that mechanisms should not solve the deliberation problems for the agents, that strategic deliberation should not occur in equilibrium, and that agents should not have incentive to provide false information about their computing results. We show that no (interesting) direct-revelation mechanism satisfies these properties. Moving beyond direct-revelation mechanisms, we show that no *value-based* mechanism (that is, any mechanism where the agents are only asked to report valuations - either partially or fully determined ones) satisfies these properties.

The rest of the chapter is organized as follows. We first show that it is possible to derive a Revelation Principle for computationally limited agents, but argue that the direct mechanism produced in the proof has highly impractical properties. We propose a set of mechanism properties which, we believe, are important when the mechanism is to be used by computationally limited agents (Section 8.2). In Section

8.3 we look at the space of *interesting* mechanisms, and present our impossibility result, before concluding the chapter in Section 8.4.

8.1 A Revelation Principle

One of the fundamental tools of mechanism design is the Revelation Principle. It states that under very weak conditions, mechanism designers need to only focus on incentive-compatible direct mechanisms in order to determine which social choice functions are implementable. That is, mechanism designers can restrict their attention to mechanisms where agents reveal their *type* truthfully to the mechanism center. The problem with computationally limited agents is that it is not clear what their *types* are. In this section we propose one definition for an agent's type, and show that it is possible to derive a Revelation Principle using this definition. However, we argue, that this approach results in impractical mechanisms, both from the perspective of the agents and the mechanism center.

We propose defining the type of a computationally limited agent to be its entire set of tools and information it uses to determine its preferences over different outcomes. That is, the type of an agent i depends on its set of algorithms ($\mathcal{A}_i = \{A_i^j\}_{j=1}^m$), its set of performance profiles ($\mathcal{PP}_i = \{PP_i^j\}_{j=1}^m$), its cost function ($\mathrm{cost}_i(\cdot)$), and the set of problem instances it is computing on ($\{x_1, \ldots, x_m\} \subseteq \mathcal{I}$).[1] The type of agent i given problem instances $\{x_1, \ldots, x_m\}$ is

$$\theta_i(\{x_1, \ldots, x_m\}) = \langle \mathcal{A}_i, \mathcal{PP}_i, \mathrm{cost}_i(\cdot), \{x_1, \ldots, x_m\} \rangle.$$

Using this definition it is straightforward to derive a Revelation Principle for computationally limited agents.

Theorem 24. *Suppose there exists a mechanism $M = (S_1, \ldots, S_n, g())$ that implements the social choice function $f(\cdot)$ in dominant strategies. Then $f(\cdot)$ is truthfully implementable in dominant strategies.*

Proof. The proof follows an argument similar to that of the original Revelation Principle. Assume agent i has m problems it can deliberate on. Let $\mathcal{A}_i$ be the set

[1] We let m denote the number of problems that the agent can compute on. These problems may be its own problems or the problems of other agents.

of algorithms and all required tools for running the algorithms for agent i, $\mathcal{PP}_i$ be the set of performance profiles, $\text{cost}_i(\cdot)$ be the cost function of agent i, and let $\{x_1, \ldots, x_m\} \subseteq \mathcal{I}$ be the specific problem instances. The type of agent i is $\theta_i(\{x_1, \ldots, x_m\}) = \langle \{\mathcal{A}_i^j\}_{j=1}^m, \{PP_i^j\}_{j=1}^m, \text{cost}_i(\cdot), \{x_1, \ldots, x_m\} \rangle$.

Suppose an indirect mechanism $M = (S_1, \ldots, S_I, g(\cdot))$ implements social choice function $f(\cdot)$ in dominant strategies. Then, there exists a strategy profile

$$s^* = (s_1^*, \ldots, s_n^*)$$

such that

$$u_i(g(s_i^*(\theta_i(\{x_1, \ldots, x_m\})), s_{-i}(\theta_{-i}))) - \text{cost}_i(s_i^*(\theta_i(\{x_1, \ldots, x_m\}))$$

$$\geq$$

$$u_i(g(s_i'(\theta_i(\{x_1, \ldots, x_m\})), s_{-i}(\theta_{-i}))) - \text{cost}_i(s_i'(\theta_i(\{x_1, \ldots, x_m\})))$$

for all s_i' and s_{-i}.

Alter the mechanism in the following way. Introduce a mediator who announces to each agent i: "Tell me your true type, and when you say your type is $\theta_i(\{x_1, \ldots, x_m\})$, the strategy $s_i^*(\theta_i(\{x_1, \ldots, x_m\}))$ will be executed. This includes computing on the problems as specified by s_i^*. An amount $\text{cost}_i(s_i^*(\theta_i(\{x_1, \ldots, x_m\}))$ will be charged for executing the strategy."

If $s_i^*(\theta_i(\{x_1, \ldots, x_m\}))$ is the optimal strategy for agent i for each $\theta_i(\{x_1, \ldots, x_m\})$ in the initial mechanism M, for any strategy chosen by the other agents, then agent i will find telling the truth to be a dominant strategy in this new mechanism. Therefore, there exists an incentive-compatible direct mechanism that implements social choice function $f(\cdot)$. $\square$

Using an identical argument it is possible to derive a Bayes-Nash Revelation Principle.

Theorem 25. *Suppose there exists a mechanism $M = (S_1, \ldots, S_n, g())$ that implements the social choice function $f(\cdot)$ in Bayesian Nash equilibrium. Then $f(\cdot)$ is truthfully implemented in Bayesian Nash equilibrium.*

At one level Theorems 24 and 25 imply that mechanism design for computationally limited agents can be reduced to mechanism design for fully rational agents. In theory, it is possible to simply allow the agents to reveal, in a single step, the information needed by the mechanism to implement the appropriate social choice function. However, the Revelation Principles derived in Theorems 24 and 25 make unrealistic assumptions about the communication capabilities of the agents as well the computational capabilities of the mechanism center itself. This is in addition to the computational issues with the Revelation Principle in classical settings [27].

First, the derived Revelation Principles assume that it is possible for the agents to reveal their types to the mechanism in a single step. This type revelation would require an agent to report all the details of all its algorithms (as well as any additional tools used like random number generators), submit its entire set of performance profiles, and fully describe its cost function. It is unrealistic to assume that an agent would be capable of fully communicating all this information in most situations.

The second concern with the Revelation Principles for computationally limited agents is with respect to the mechanism center itself. The theorems assume that, once given all information by the agents, the mechanism center has enough computing resources of its own that it can find the optimal computing policies for all agents. Additionally, the mechanism must compute the outcome of the mechanism, which can possibly a computationally difficult problem in and of itself. Clearly, this assumption about the mechanism center's computing resources is impractical and unrealistic.

8.2 New Properties for Mechanisms

We believe that mechanisms for computationally limited agents should have good computing properties in addition to good economic properties. In this section we propose a set of properties which we argue are desirable.

Property 1 (Non-Deliberative). *A mechanism should be* non-deliberative*. That is, the mechanism should not compute on or solve the agents' individual valuation problems.*

If a mechanism is non-deliberative the agents' are responsible for solving their own valuation problems. The mechanism's primary concern is determining an outcome given the agents' strategies. We believe that in many settings it is unreasonable to assume that the mechanism is capable of both determining optimal deliberation policies for agents and computing the outcome given the policies. The problem with the mechanisms presented in Theorems 24 and 25 is that they were not non-deliberative.

Property 2 (Deliberation-Proof). *A mechanism should be* deliberation-proof*. That is, agents should have no incentive to strategically deliberate.*

Recall that strategic deliberation is defined to be the act of actively using computing resources in order to get information about the valuations of the other agents. We believe that strategic deliberation places too high of a strategic overhead on agents. A well designed mechanism reduces the amount of strategizing required by agents to act optimally and a deliberation-proof mechanism eliminates the need of strategic deliberation in order for an agent to act optimally.

Many commonly used auction mechanisms are not deliberation-proof. In Chapter 7, we showed that the Vickrey, ascending, first-price sealed-bid, and generalized Vickrey auction are not deliberation-proof.

Property 3 (Non-Deceiving). *A mechanism should be* non-deceiving*. Assume that v_i is the true computed value of agent i. A mechanism is non-deceiving if the agent never has incentive to send a report to the mechanism such that if any agent had seen the report, their belief that agent i's value is v_i would be 0.*

A non-deceiving mechanism does not require that an agent directly reveal its actual computing results. It just ensures that an agent does not lead other agents to believe that its true value is not possible. For example, assume that an agent had secretly computed a value v. A mechanism would be deceiving if the agent had incentive to report that its value was strictly greater than v. The mechanism would not be deceiving if the agent had incentive to report that its value was greater than some y, $y < v$.

Since we desire that mechanisms be non-deliberative, we restrict our attention to *value-based mechanisms* for the rest of the chapter.

Definition 42 (Value-based Mechanism). *A value-based mechanism, $M = (S_1, \ldots, S_N, g())$, is a mechanism where each agent i's strategies are restricted so that the only allowable messages are functions of (partially) determined valuation functions and where $g(\cdot)$ is a function only of the agents declared values.*

Value-based mechanisms are non-deliberative. The mechanism is not given any of the tools required for it to actively compute on the agents' valuation problems. Examples of value-based mechanisms include sealed-bid auctions where agents submit numerical bids, and ascending and posted-price auctions where agents answer yes or no to the query of whether they would be willing to buy an item at a specified price.

Mechanisms can be broadly classified into two groups, sensitive and non-sensitive.

Definition 43 (Sensitive). *A mechanism is* sensitive *to agents' strategies if for each agent i there exist strategies s_i', s_i'', $s_i' \neq s_i''$ such that for strategy profiles*

$$s' = (s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n),$$

$$s'' = (s_1 \ldots, s_{i-1}, s_i'', s_{i+1}, \ldots, s_I),$$

it is the case that

$$g(s') \neq g(s'').$$

Most mechanisms of interest are sensitive. For example, efficient mechanisms such as all VCG mechanisms are sensitive since the final outcomes depends on the information provided by the agents. Non-sensitive mechanisms are ones in which agents' strategies do not influence the outcome. Examples of non-sensitive mechanisms are dictatorial mechanisms which always choose the preferred outcome of a specific agent and completely random mechanisms which choose outcomes in a completely random fashion.

The next theorem makes the observation that if the agents' strategies do not influence the outcome of the mechanism, then there is no incentive for an agent to invest computing resources into competitors' valuation problems.

Theorem 26. *Assume mechanism $M = (S_1, \ldots, S_n, g(\cdot))$ is value-based and non-sensitive. Then M is non-deliberative, non-deceiving and deliberation-proof.*

Proof. Suppose a mechanism $M = (S_1, \ldots, S_n, g(\cdot))$ is value-based and non-sensitive. Clearly, the mechanism is non-deliberative. Since it is value-based, the mechanism is not provided by the set of tools it would require to solve the agents' deliberation problems. Let s_i' and s_i'' be two strategies for agent i such that all deliberation actions of specified by the two strategies are the same, but s_i' is non-deceiving and s_i''

is deceiving. Therefore, $\text{cost}_i(s_i') = \text{cost}_i(s_i'')$. Since the mechanism is non-sensitive, $g(s_i', s_{-i}) = g(s_i'', s_{-i})$ and so

$$u_i(g((s_i', s_{-i})) - \text{cost}_i(s_i') = u_i(g(s_i'', s_{-i})) - \text{cost}_i(s_i'')$$

and so following a non-deceiving strategy is weakly dominant for agent i.

A similar argument holds when showing that the mechanism is deliberation-proof. Let s_i be a strategy such that agent i deliberated only on its own problems. Further assume that it devotes some amount r_i on deliberating. Let s_i' be any strategy where agent i deliberates r_i on its own problems and r_{-i} on other agents' problems. Since the mechanism is not sensitive,

$$g(s_i, s_{-i}) = g(s_i', s_{-i})$$

for all s_{-i}. Therefore,

$$u_i(g(s_i, s_{-i})) - \text{cost}_i(s_i) \geq u_i(g(s_i', s_{-i})) - \text{cost}_i(s_i')$$

since the cost function is additive and nondecreasing. This holds for any r_i. Therefore, agent i is always better off by not deliberating on other agents' problems. □

This theorem shows us that it is possible to design a mechanism with our three desired properties. However, at the same time it is not particularly useful as the class of mechanisms which we have shown to exhibit these properties are not useful in most settings. In the rest of this chapter we focus solely on mechanisms which have more practical applications, the sensitive mechanisms.

8.3 Sensitive Mechanisms

In this section we study sensitive mechanisms in order to understand whether it is possible to design mechanisms which are non-deliberative, deliberation-proof and non-deceiving. We start by studying the properties of direct mechanisms.

Theorem 27. *There exists no value-based sensitive direct mechanism that is deliberation-proof across all problem instances. (An instance is defined by the agents performance profiles, cost functions, and current problem instance.)*

Proof. To prove the theorem, we need only show that there exist instances, defined by the performance profiles, where in equilibrium agents will strategically compute. We will restrict ourselves to mechanisms which are individually rational. If a mechanism does not satisfy participation constraints then agents have no incentive to compute in the first place.

Let $M = (S_1, \ldots, S_n, g(\cdot))$ be any incentive compatible mechanism. Since our agents have quasilinear utilities, it must be the case the $g(\cdot) = (k(\cdot), t(\cdot))$ where $k(\cdot)$ is a mapping from strategy profiles into allocations and $t(\cdot)$ is a mapping from strategy profiles into transfer vectors.

Assume that there are two agents ($n = 2$), and assume that the agents have access to the same algorithms. Each agent i has performance profiles $PP_i^{\{i,j\}}$, and a cost function $\text{cost}_i(\cdot)$ where

$$\text{cost}_1((t_1, t_2)) = \epsilon(t_1 + t_2)$$

for some small $\epsilon > 0$, and

$$\text{cost}_2((t_1, t_2)) = t_1 + Kt_2$$

for some constant $K > 1$. The algorithms, performance profiles, and cost functions are common knowledge.

Since we are free to choose the instance, we define the performance profiles as follows. The performance profile for agent 1 is

$$PP_1((t_1, t_2)) = \begin{cases} 0 & \text{if } \sum t_i = 0 \\ v_1^h \text{ with probability } p & \text{if } t_1 > 0 \\ v_1^l \text{ with probability } 1 - p & \text{if } t_1 > 0 \end{cases}$$

The performance profile for agent 2 is

$$PP_2((t_1, t_2)) = \begin{cases} 0 & \text{if } \sum t_i = 0 \\ v_2^h \text{ with probability } q & \text{if } t_2 > 0 \\ v_2^l \text{ with probability } 1 - q & \text{if } t_2 > 0 \end{cases}$$

where v_1^h, v_1^l, v_2^h and v_2^l are chosen such that the allocation function specifies $k(v_1^h, v_2^*) = 1$, $k(v_1^*, v_2^l) = 1$ and $k(v_1^l, v_2^h) = 2$. The transfers $t_i(v_1, v_2)$ are determined so that the mechanism is incentive compatible. In particular, the transfer function of agent i can not be a function of its own declaration.

For small ϵ agent 1 has a dominant strategy which is to compute for one step on its own problem. For example if agent 2 decides not to compute then, clearly, agent 1 is best off computing on its own problem since it does not matter what value agent 2 could have obtained. If agent 2 computes on its own problem, then agent 1 is still best off computing on its own, since

$$-\epsilon + p\left[v_1^h + qt_i(v_2^h) + (1-q)t_1(v_2^l)\right] + (1-p)(1-q)\left[v_1^l + t_1(v_2^l)\right] \geq 0$$

where 0 is the utility if agent 1 does not compute and

$$\begin{aligned}-\epsilon + p\left[v_1^h + qt_i(v_2^h) + (1-q)t_1(v_2^l)\right] + (1-p)(1-q)\left[v_1^l + t_1(v_2^l)\right] \quad &\geq \\ -\epsilon q + (1-q)\left[-2\epsilon + pv_1^h + (1-p)v_1^l + t_1(v_2^l)\right]\end{aligned}$$

where the left hand side is the utility if agent 1 deliberates on agent 2's problem and then deliberates on its own problem only if agent 2's valuation is v_2^l. In a similar way it is easy to show that for small ϵ agent 1 is best off deliberating on its own problem when agent 2 deliberates on agent 1's problem first.

Given that agent 1 will compute on its own problem, agent 2 must determine its best strategy. It can determine its own true valuation for a cost of $K > 1$ but it can also determine the valuation of agent 1 at a cost of 1. Agent 2 will first determine whether agent 1 has high or low type under the following conditions;

$$-p + (1-p)\left[q(v_2^h + t_2(v_1^l)) - (1+K)\right] \geq 0,$$

that is agent 2 is better off computing on agent 1's problem than not computing at all, and

$$-p + (1-p)\left[q(v_2^h + t_2(v_1^l)) - (1+K)\right] \geq -K + (1-p)q\left[v_2^h + t_2(v_1^l)\right],$$

that is agent 2 is better off computing on agent 1's problem than computing on its own problem. This means that agent 2 will strategically deliberate if K is set such that

$$\frac{p + (1-p)q}{1-(1-p)q} \leq K \leq q(v_2^h + t_2(v_1^l)) - \frac{1+p}{1-p}$$

where we are free to choose p, q, v_2^h, v_1^l. $\square$

We knew that the Vickrey auction and the generalized Vickrey auction were not strategy-proof from the results in Chapter 7. This proof shows that the problem is not with the payment rules or how the allocation is decided, but instead lies with the fact that the mechanisms are executed in a single-shot and do not provide the agents with any opportunity to gather information about their competitors, other than through strategic-deliberation. The proof also provides a way of finding instances for which strategic-deliberation occurs. In particular it provides a technique for constructing sets of performance profiles which induce strategic-deliberation.

While direct mechanisms are not deliberation-proof in general, they do allow us to study why strategic deliberation occurs. First, by learning the valuations of others, an agent may be able to tailor the announcements in such a way so as to increase their chances of being included in the final allocation. However, by designing mechanisms which are incentive-compatible this issue disappears. A more insidious problem is the following. Agents may be asymmetric due to cost functions, algorithms or problem instances. It may be more difficult to deliberate on some problems as opposed to others. If an agent finds itself in a situation where it is easier to deliberate on a competitor's problem compared to its own problems, then it may strategically deliberate in order to determine if it is worth while to actually do (or continue doing) its own deliberation. For example, in a Vickrey auction, if agent 1 can learn for a low cost that agent 2 has achieved a very high valuation after deliberating then it may be in agent 1's best interest to not waste additional deliberation resources on determining its own problems.

To avoid strategic deliberation, the computationally limited agents must be provided with enough information by the mechanism so that they can determine whether to devote resources to their own problems or not.

Many indirect mechanisms reveal information and this information can be used by the agents to help determine which are the best computing and non-computing actions to take. For example, in a single item ascending auction, as the price rises the mechanism may reveal to all agents the number of agents remaining in the auction. The agents can use this information to deduce valuation information about the remaining agents.

To explicitly model the information that is revealed to the agents by the mechanism we introduce a *feedback game*. A feedback game (M, F) is the extensive form game induced by mechanism M (this includes the computing actions of the agents) coupled with a feedback function. At each stage of the game, the feedback function maps all messages that are sent to the mechanism at that stage to the information that would have been revealed to the agents through the mechanism. To illustrate, we provide two examples. Let M be a direct mechanism. At each stage agents have the choice of taking a computing action, and at some stage t' agents submit bids to the auction. The corresponding feedback function is

$$F(t, \omega(t)) = \begin{cases} \emptyset & \text{if } t \neq t' \\ (x, p) & \text{if } t = t' \end{cases}$$

where $\omega(t)$ is the vector of agents' messages, and (x, p) specifies the final allocation (x) and the prices paid by the agents (p).

In an ascending auction more information is revealed by the mechanism. The feedback function for an ascending auction is

$$F(t, \omega(t), p) = \{i| \text{ agent } i \text{ is still in the auction at price } p\}.$$

Introducing an explicit feedback function does not change the original mechanism. However, it provides a tool to the mechanism designer for reasoning about what information is available to agents and how the agents are doing belief revision given the information reported by the feedback function.

Lemma 5. *Given any mechanism M it is possible to construct a feedback function such that the equilibria in the feedback game, (M, F), are the same equilibria in the original mechanism M.*

Proof. Let $M = (S_1, \ldots, S_n, g(\cdot))$ be any mechanism and let G^M be the extensive form game that agents play when participating in mechanism M. For each stage t in the game let $\omega(t)$ represent the messages sent to the mechanism by the agents. Let $\text{info}(\omega(t))$ be the information revealed at stage t in the game G^M after agents send $\omega(t)$. Define the feedback function $F(\cdot)$ as follows

$$F(t, \omega(t)) = \text{info}(\omega(\text{t})).$$

By introducing the feedback function, nothing in the game G^M has been changed. Therefore, the feedback game (M, F) is equivalent to the game G^M. □

Using the feedback function as a tool to help in the analysis, we are able to show our final impossibility result. In general, sensitive value-based mechanisms do not satisfy the proposed desirable properties.

Theorem 28. *There exists no sensitive value-based mechanism that is*

- *non-deliberative,*
- *deliberation-proof, and*
- *non-deceiving*

across all problem instances. (An instance is defined by agents' performance profiles, cost functions).

Proof. Let mechanism M be an incentive compatible (non-deceiving) direct mechanism which implements social choice function $f(\cdot)$. Mechanism M is not deliberation-proof (Theorem 27). Therefore, it is possible to construct problem instances such that in equilibrium, agents will strategically deliberate. Using the technique in the proof for Theorem 27 construct performance profiles PP_1, PP_2 (such that in one deliberation step agents learn their true valuation) and cost functions, $\text{cost}_1((t_1, t_2))$ $\text{cost}_2((t_1, t_2))$, for 2 agents such that in the mechanism M, one agent (agent 2) has incentive to strategically deliberate while the other (agent 1) has a dominant strategy to deliberate only on its own problem.

We also use the additional minor assumptions which make the proof clearer. First, if an agent decides not to compute and thus has valuation 0, then we assume that its strategy indicates to the mechanism that $v_i = 0$. Second, we assume that $t_i(0) \leq t_i(v)\ \forall v \neq 0$. This means that if the agent has not computed and has no value, then its transfer can not be greater than if it had computed and declared a valuation. These assumptions do not change the equilibrium behavior of the agents since if an agent, for all intents and purposes, resigns from the game, declaring its valuation to be anything else but the default valuation of 0 is weakly dominant.

Since strategic deliberating occurs in equilibrium in the direct mechanism M, there exists an information set H_2 such that it is agent 2's turn to move and

$$E[u_2|H_2, \mu, ((a_1, H_2)s_2, s_1] \geq E[u_2|H_2, \mu, (a_2, H_2)s_2, s_1]$$

or

$$E[u_2|H, \mu, ((a_1, H_2)s_2, s_1] \geq E[u_2|H_2, \mu, (\emptyset, H_2)s_2, s_1]$$

where $(a, H)s$ denotes the strategy which specifies taking computing action a at information set H. In particular in the direct mechanism this information set occurs at the start of computing for agent 2.

Let $M' = (S_1, \ldots, S_n, k(\cdot), t_1(\cdot), \ldots, t_n(\cdot))$ be an indirect mechanism which also implements the social choice function $f(\cdot)$. Let s^* be the equilibrium strategy profile for mechanism M'. Since the mechanism implements the same social function as M the outcomes of the two mechanisms must be the same. Create an appropriate feedback function $F(\cdot)$ to produce feedback game $(M', F(\cdot))$ (Lemma 5).

Let v_i represent agent i's actual valuation that it could achieve if it computed on its own problem (Recall that in this example instance, agents need only compute for one step on a problem to determine the valuation. With no computation their valuation is 0). At each stage t of the game induced by the mechanism M' define $\overline{V_i}(t)$ to be a partition that agent i makes of agent j's possible computable valuations. In particular, $\overline{V_i}(t) = \{V, V'\}$ where if $v_i \in V$ then agent i has incentive to compute on its own problem and if $v_i \in V'$ then agent i is best off stopping all computing. Given the construction of the problem instance, there exists stages such that $V' \neq \emptyset$.

The feedback function F can exhibit different properties at each stage t of the game. In particular, the feedback function can be either pooling or separating.

Definition 44 (Separating). *Let $\overline{V_i}(t) = \{V, V'\}$ be a partition for agent i at stage t. A feedback function F is* separating at stage t *if for any messages $m(v_j)$, $m(v'_j)$ sent by agent $j \neq i$ such that $v_j \in V$ and $v'_j \in V'$,*

$$F(t, (m(v_i), m(v_j))) \neq F(t, (m(v_i), m(v'_j))).$$

Definition 45 (Pooling). *A feedback function $F(\cdot)$ is* pooling at stage t *if it is not separating at stage t.*

Since agent 1 has a dominant strategy we assume that it determines its own valuation and then sends signals to the mechanism. Assume that at stage $t-1$ agent 2 has partition $\overline{V}_i(t-1) = \{V, V'\}$ where $V' \neq \emptyset$. Upon observing in stage t $F(t, s_1(v_1))$ agent 2 updates its beliefs about what set the actual valuation of agent 1 is in.

Assume the feedback function F is pooling at stage t. Let $m(v^*)$ be the message that agent 1 sends to the mechanism in equilibrium and let $a^*(F(t, m(v^*))$ be the deliberation action that agent 2 takes upon observing $F(t, (v^*))$. Since F is pooling at stage t, agent 2 must base its beliefs on whether $v_1 \in V$ or $v_1 \in V'$ solely on the probability given by the performance profiles. However, this is the same beliefs that agent 2 would have in a situation where there was a direct mechanism. Therefore, agent 2 is best off strategically deliberating (since the performance profiles were chosen so as to have strategic deliberation in the direct mechanism) or doing nothing (not even sending a message) and waiting for the next stage of the game.[2]

Assume that at stage t the feedback function is separating. Again, let $m(v_1^*)$ be the signal that agent 1 sends to the feedback mechanism in equilibrium and let $a^*(F(t, m(v_1^*)))$ be the action that agent 2 takes in equilibrium upon observing $F(t, m(v_1^*))$. In perfect Bayesian equilibrium, the beliefs on the equilibrium path must be correctly derived from the equilibrium strategies using Bayes Rule. This implies that upon seeing $F(t, m(v_1^h))$ agent 2 must assign probability one to agent 1 having high type. Similarly it assigns probability one to agent 1 having low type if it observes $F(t, m(v_1^l))$. Upon observing $F(t, m(v_1^h))$ agent 2 believes that agent 1 has high type, it also believes that the mechanism will choose an outcome preferable to agent 1 (since the mechanism is implementing social choice function $f()$ which would select an outcome favorable to agent 1 when it has high type). Therefore, agent 2 is best off not computing at all since otherwise it would just incur a cost without being able to recoup the cost from getting a preferred outcome. Therefore, $a^*(F(t, m(v_1^h))$ is to do nothing, agent 1 gets its preferred outcome and pays $t_1(0)$. That is, the utility of agent 1 is

$$v_1^h + t_1(s_2(0)) - \epsilon = v_1^h + t_1(0) - \epsilon$$

If agent 2 witnessed $F(t, m(v_1^l))$ then it believes that agent 1 is low type and it has incentive to deliberate. The expected utility for agent 1 in this situation is

$$(1-q)(v_1^l + t_1(s_2(v_2^l))) - \epsilon$$

However, if agent 1 has a low type but sends a message to the mechanism $m(v_1^h)$ then agent 2 would not deliberate and the utility of agent 1 would be

$$v_1^l + t_1(s_2(0)) - \epsilon \geq (1-q)(v_1^l + t_1(s(v_2^l))) - \epsilon$$

Therefore, if the feedback function is separating, agent 1 has incentive to misreport its deliberated value to the feedback mechanism. Therefore, the mechanism is deceiving. □

[2]To avoid situations where agents wait forever, one can introduce a liveness condition which will force an agent to eventually take some action.

8.4 Summary

In this chapter we laid out mechanism design principles for computationally limited agents. We first showed that the revelation principle applies to such settings in a trivial sense by having the mechanism carry out all the computing for the agents. This is impractical, and we proposed that mechanisms should be *non-deliberative*: the mechanism should not be solving the deliberation problems for the agents. Second, mechanisms should be *deliberation-proof*: agents should not deliberate on others' valuations in equilibrium. Third, the mechanism should be *non-deceiving*: agents do not strategically misrepresent. Finally, the mechanism should be *sensitive*: the agents' actions should affect the outcome. We showed that no direct-revelation mechanism satisfies these four intuitively desirable weak properties. Moving beyond direct-revelation mechanisms, we showed that no *value-based* mechanism (that is, mechanism where the agents are only asked to report valuations - either partially or fully determined ones) satisfies these four properties.

This result is negative. It states that either we must have mechanisms which do the computing for the agents, or complex strategic (and costly) counterspeculation can occur in equilibrium. However, there is some hope. It may be possible to weaken one of the properties slightly, while still achieving the others. For example, it may be possible to design multi-stage mechanisms that are not value based; the mechanism could help each agent decide when to hold off on computing during the mechanism (and when to compute on one's own valuation for different bundles of items in a combinatorial auction). In another direction, by relaxing strategic deliberation and compensating agents appropriately, it may be possible to design mechanisms where agents who can solve problems cheaply and efficiently do so for all agents.

Chapter 9

The Social Cost of Selfish Computing

In this chapter we study the system-wide impact of computationally limited agents. We introduce a way of measuring the negative impact on the system as a whole when agents are free to choose deliberation strategies according to their own self-interest (i.e. selfishly). Our *miscomputing ratio* isolates the effect of computing from the other strategic actions that the agents might take. It compares the social welfare in the situation where all agents are allowed to freely choose their own strategies against the optimistic measure of social welfare where a global deliberation controller dictates the deliberation policies of agents so as to maximize social welfare, while leaving the agents free in their choice of which other actions to take. Our miscomputing ratio, like the price of anarchy [55, 99], provides a measure of how bad things can get if agents are free to act in their own self-interest.

We apply the miscomputing ratio to a Vickrey auction. We chose the Vickrey auction as our example application for two reasons:

- We have shown that if agents are allowed to compute freely then strategic deliberation can occur in equilibrium.
- The bidding rules allow for a clear distinction between deliberation actions and bidding actions.

We show that even in very simple settings, the equilibrium outcome can be far worse than in the case where a global controller specifies deliberation policies, but that by the careful use of small cost functions, it is possible to provide the appropriate incentives so that the right agents compute on the right problems.

The rest of the chapter is organized as follows. In the next section we introduce and define the miscomputing ratio. We then apply the miscomputing ratio to a scenario where computationally limited agents are participating in a Vickrey auction. We provide an equilibrium analysis, and a discussion of the role of cost functions and how they should be used. We then conclude with a summary of the chapter along with a discussion of different settings where the results of this chapter could be applied.

9.1 The Social Cost of Selfish Computing

A natural question to ask is whether the restrictions on computing resources results in a loss of efficiency. However, efficiency is difficult to compare in such settings. For example, 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 computing resources on a different problem, thus causing a different agent to have the highest valuation and win the auction. This outcome is still efficient *given how agents computed*, but it overlooks the computational issues in an unsatisfying way. This suggests that Pareto efficiency may not always be the right measure to use in the context of computationally limited agents. Is there an alternative measure?

Instead of looking at efficiency, we propose to use social welfare as the measure. We want to know how letting agents freely choose their own computing strategies impacts the social welfare of the set of all bidders. In particular, we 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, and given this information, specifies exactly how each agent must use its computational resources. In the bidding stage agents are free to bid as they wish, but their goal is still to maximize their own utility.

Definition 46. *Let o^* be the outcome that is reached if the global controller dictates computing policies to all agents, and agents are free to bid as they wish.*

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

Definition 47 (Worst Case Nash). *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 48 (Miscomputing Ratio). *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.

We actually study the impact of selfish computing in two slightly different settings. In the first setting we are only interested in the impact on the strategic agents (i.e. the bidding agents). In this case, if N is the set of bidding agents then

$$SW(o) = \sum_{i \in N} u_i(o).$$

We denote the miscomputing ratio in this setting as R_N. In the second setting we study the impact of computation on *all* agents in the game, including the auctioneer. The social welfare of an outcome is computed as

$$SW(o) = \sum_{i \in N} u_i(o) + u_{\text{auc}}(o)$$

where $u_{\text{auc}}(o)$ is either the amount that was paid to the auctioneer by the winning bidder, or else the value the auctioneer has for the item up for auction in the case where no one wins (and so the auctioneer keeps the item). Under this social welfare measure, the miscomputing ratio is denoted by $R_{N \cup \{auc\}}$.

9.2 The Vickrey Auction and the Miscomputing Ratio

In this section we present the results. In particular, we use the single item Vickrey auction as an illustrative example of the miscomputing ratio. We chose the Vickrey auction as our application since it has a straightforward bidding strategy (agents have incentive to bid truthfully, given their computed valuations), and because it is representative of all auction mechanisms in that the mechanism provides incentives for agents to strategically deliberate.

The rest of this section is organized as follows. We first outline the assumptions made, and prove a result concerning agents' strategies that is used throughout. We then do a Bayes-Nash equilibrium analysis of the Vickrey auction with computationally limited agents. Using this equilibrium analysis, it is possible to determine the miscomputing ratio under different conditions. We show that sometimes it is best to just allow all agents to compute freely, while at other times cost functions can be used to motivate all agents to compute "correctly".

9.2.1 Agents' Nondominated Strategies

Assume there is a set of bidding agents, N, who are competing in a Vickrey auction being run by auctioneer agent, *auc*. Before a bidding agent can submit a bid, it must first use some of its resources to determine its valuation. For ease of exposition, we assume that each agent i has a deterministic algorithm, v_i, and performance profiles and that these are common knowledge.[1] While this assumption does make the analysis simpler in that there is less uncertainty in what possible values have been computed, agents with deterministic performance profiles can still suffer from miscomputing.

Each agent has to use some of its resources (time) while computing. We model this expenditure by assuming that each agent, i, has a cost function. The cost function of agent i, cost_i, is private. However, the cost function is drawn from some set $\mathcal{C}_i$ by distribution $f_i : \mathcal{C}_i \mapsto [0, 1]$ where f_i is common knowledge.

An agent's utility depends on whether it has computed, what value it has obtained by computing, and whether it has won the auction. If an agent does not compute

[1]This assumption is not necessary. We can also assume the more general situation where v_i is drawn with distribution $g_i()$ from the set of algorithms V_i, where $g_i()$ is common knowledge.

then it does not have a valuation for the item, and so we state that its utility is 0. Assume an agent computes for t_{own} on its own problem and t_{total} on all problems. If it does not win the auction, then its utility is $u_i = -\text{cost}(t_{\text{total}})$. If it does win the auction, and the second highest bid is b, then its utility is $u_i = v(t_{\text{own}}) - \text{cost}(t_{\text{total}}) - b$.

Given these assumptions, we can now determine agents nondominated strategies.

Theorem 29. *Assume that agent $i \in N$ has a deterministic algorithm v_i and some cost function $\text{cost}_i(\cdot)$. Then, agent i has only two possible nondominated strategies. It will either not compute at all, or it will compute for t_i^* steps on its own problem where*

$$t_i^* = \arg\max_t \{v_i(t) - \text{cost}_i(t)\}.$$

If agent i does compute, then it submits a bid equal to $v_i(t_i^*)$.

Proof. First, the argument that an agent is best off submitting a bid equal to the value that it has computed is identical to the proof that bidding truthfully is optimal for rational agents. Thus, we omit this part of the proof. If an agent does not compute at all, then it cannot submit a bid and be allocated the item. We use this as the default situation, and assume that an agent i will have utility $u_i = 0$.

Second, agent i is best off never computing on another agent's valuation problem. Since the algorithms are deterministic and common knowledge, agent i already knows what $v_j(t)$ is for any agent j and any time t. Therefore, by computing on agent j's problem it gains no new knowledge, yet incurs a cost.

Finally, if the agent does compute for t time steps then its utility is

$$u_i = \begin{cases} v_i(t) - \text{cost}_i(t) - b & \text{if } v_i(t) > b = \max_{j \neq i}\{b_j\} \\ -\text{cost}_i(t) & \text{otherwise} \end{cases}$$

If agent i won the auction, then it is best off computing so as to maximize its utility, that is computing for t_i^* steps on its problem. If agent i did not win, then it is best off not having computed at all ($u_i = 0$). ∎

9.2.2 When Should an Agent Compute?

From Theorem 29 we know that agent i should either compute so as to maximize $v_i(t) - \text{cost}_i(t)$ or not compute at all. Which strategy it should follow depends on

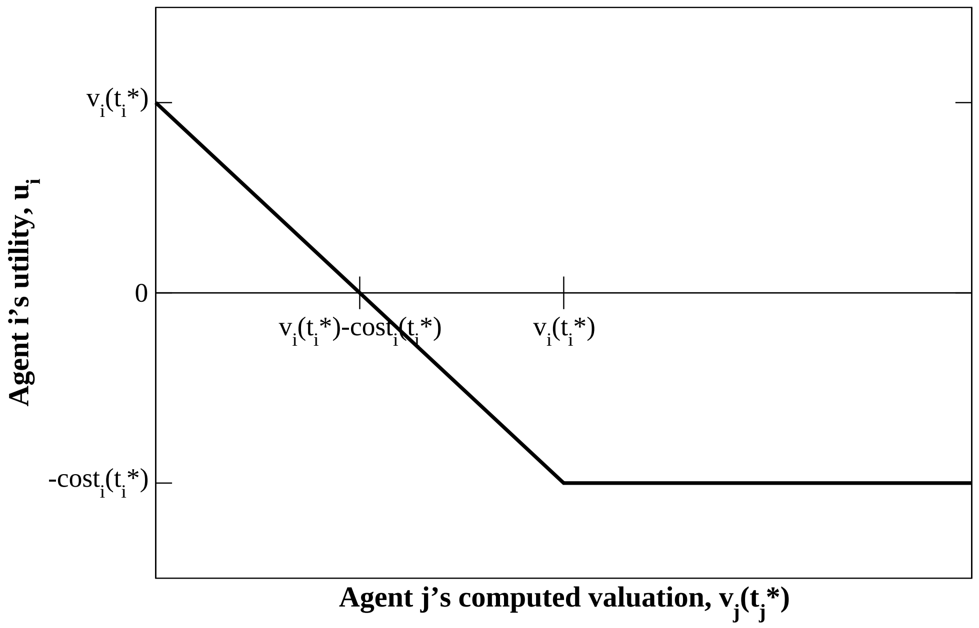


Figure 9.1: *Assuming that both agent i and agent j have computed, the utility of agent i is a decreasing function of the computed valuation of agent j. As agent j's computed valuation increases, agent i's utility decreases. Agent i's utility is 0 when $v_j(t_j^*) = v_i(t_i^*) - \text{cost}_i(t_i^*)$. When $v_j(t_j^*) \geq v_i(t_i^*)$, the utility of agent i is equal to $-\text{cost}_i(t_i^*)$.*

what other bidding agents are doing. Figure 9.1 illustrates how an agent's utility depends on both its computational actions and the bids of other agents.

So far we have determined how much an agent should compute, if it decides to, but we still do not know under what conditions an agent should decide to compute. If agent i knew every other agents' computed valuations and thus what they would bid, it would be able to optimally decide whether to compute or not. While an agent does not have this information available to it, it does have probabilistic information about the other agents' cost functions. It can use this information to derive a distribution over every other agents' possible computed valuations. Recall that $\mathcal{C}_j$ is the set of possible cost functions for agent j and $f_j(\text{cost}_j(\cdot))$ is the probability that agent j has cost function $\text{cost}_j(\cdot)$. The distribution, $\overline{f}_j$, over the valuations that agent j may have computed is

$$\overline{f}_j(x) = \int_{\mathcal{C}_j} f_j(\text{cost}_j)\chi_{\text{cost}_j}(x)dc_j$$

where

$$\chi_{\text{cost}_j}(x) = \begin{cases} 1 & \text{if } x = \max_t [v_j(t) - \text{cost}_j(t)] \\ 0 & \text{otherwise.} \end{cases}$$

For ease of presentation we assume that there are just two bidding agents, i and j.[2] Let $s_j(x)$ represent the strategy of agent j if $v_j(t_j^*) = x$. That is

$$s_j(x) = \begin{cases} 1 & \text{if } j \text{ computes when } x = v_j(t_j^*) \\ 0 & \text{if } j \text{ does not compute when } x = v_j(t_j^*) \end{cases}$$

Let $P_j(x)$ be the probability that $s_j(x) = 0$.

The expected utility from computing for agent i depends on what agent j decides to do. If agent j does not compute, then, by the rules of the Vickrey auction, agent i will win the auction, and will pay nothing for the item. Its utility will be $u_i = v_i(t_i^*) - \text{cost}_i(t_i^*)$. If agent j does compute, then the utility of agent i depends on whether the computed valuation of agent j is greater than its own. If $v_j(t_j^*) < v_i(t_i^*)$, then the utility of agent i is $u_i = v_i(t_i^*) - \text{cost}_i(t_i^*) - v_j(t_j^*)$. However, if $v_j(t_j^*) > v_i(t_i^*)$ then $u_i = -c_i(t_i^*)$. This is captured in the following equation:

$$\begin{aligned} u_i \;=\; & \underbrace{\int_0^\infty P_j(x)\overline{f}_j(x)(v_i(t_i^*) - \text{cost}_i(t_i^*))dx}_{j \text{ does not compute}} \\ & + \underbrace{\int_0^{v_i(t^*)} (1 - P_j(x))\overline{f}_j(x)(-\text{cost}_i(t_i^*))dx}_{j \text{ computes and } v_j(t_j^*) \geq v_i(t_i^*)} \\ & + \underbrace{\int_{v_i(t_i^*)}^\infty (1 - P_j(x))\overline{f}_j(x)(v_i(t_i^*) - \text{cost}_i(t_i^*) - x)dx}_{j \text{ computes and } v_j(t_j^*) < v_i(t_i^*)} \end{aligned}$$

[2]The multiple agent setting is not conceptually more difficult. The techniques and logic used to find the equilibrium is the same as the two agent setting. However, one must keep track of multiple probability distributions which makes the notation cumbersome. For this reason, we present the two agent case.

Agent i will only compute if the above equation is greater than its utility from not computing ($u_i = 0$). Therefore, agent i will only compute under the condition

$$\begin{aligned}\text{cost}_i(t_i^*) \quad &\leq \quad v_i(t_i^*)\left[1 + \int_{v_i(t_i^*)}^{\infty} P_j(x)\overline{f}_j(x)dx\right] \\ &\quad - \int_0^{v_i(t_i^*)} (1 - P_j(x))\overline{f}_j(x)xdx\end{aligned}$$

This is a cutoff equilibrium. Agent i will compute only when its cost for computing is below a certain threshold.

9.2.3 Examples

With the analysis in the previous section, we are able to determine when an agent should compute and when it should not.

Free Computation with Deadlines

Assume that all agents have cost functions of the following form

$$\text{cost}_i(t) = \begin{cases} 0 & \text{for } t \leq D_i \\ \infty & t > D_i. \end{cases}$$

This corresponds to the situation where agents have free computation but must stop computing by some deadline. We will assume that the uncertainty between the agents about the cost functions is caused by having the deadlines be private information. For each agent i, $t_i^* = D_i$ since $D_i = \arg\max_t\{v_i(t) - \text{cost}_i(t)\}$. Clearly

$$v_i(D_i)\left[1 + \int_{v_i(D_i)}^{\infty} P_j(x)\overline{f}_j(x)dx\right] \geq v_i(D_i).$$

Since

$$\int_0^{v_i(D_i)} (1 - P_j(x))x\overline{f}_j(x)dx \leq v_i(D_i)$$

it is always the case that for $t_i^* = D_i$

$$
\begin{aligned}
0 = \mathrm{cost}_i(t_i^*) \quad \leq \quad & v_i(t_i^*)\left[1 + \int_{v_i(t_i^*)}^{\infty} P_j(x)\overline{f}_j(x)dx\right] \\
& - \int_0^{v_i(t_i^*)} (1 - P_j(x))\overline{f}_j(x)xdx.
\end{aligned}
$$

That is, agent i will compute on its own problem until its deadline D_i.

Constant Cost Function

Assume that agents i and j share an algorithm, v, such that for all t, $v(t) = V$. Assume that the agents have constant cost functions, $\mathrm{cost}_i(t)$, $\mathrm{cost}_j(t)$ such that for all t, $\mathrm{cost}_i(t) = K_i$ and $\mathrm{cost}_j(t) = K_j$, where K_i, K_j are drawn uniformly from the interval $[0, K]$ for some constant K. Each agent knows its own cost function, but only knows that its competitor's cost function is drawn uniformly from $[0, K]$.

If agent i decides to compute on its problem, its utility depends on whether or not agent j also decided to compute on its problem. Assuming that agent i computed, its utility is

$$
u_i = \begin{cases} V - K_i & \text{if agent } j \text{ did not compute} \\ -K_i & \text{if agent } j \text{ did compute} \end{cases}
$$

since, if both agents computed then the item could be allocated to either of them but they would have to pay V. In equilibrium, agent i will only compute when its expected utility from computing is greater than not computing. That is

$$
0 \leq \int_0^K P_j(x)(V - K_i)dx + \int_0^K (1 - P_j(x))(-K_i)dx
$$

As before, this is a cutoff equilibrium. Let $\hat{K}_i$ and $\hat{K}_j$ be the costs at which each agent switches from a strategy involving computing to one where it does not compute. Then, in equilibrium,

$$
\hat{K}_i = \int_0^K \frac{\hat{K}_j}{K}(V - \hat{K}_i)d\hat{K}_j + \int_0^K (1 - \frac{\hat{K}_j}{K})(-\hat{K}_i)d\hat{K}_j.
$$

This reduces to

$$
\hat{K}_i = \frac{VK}{2(K+1)},
$$

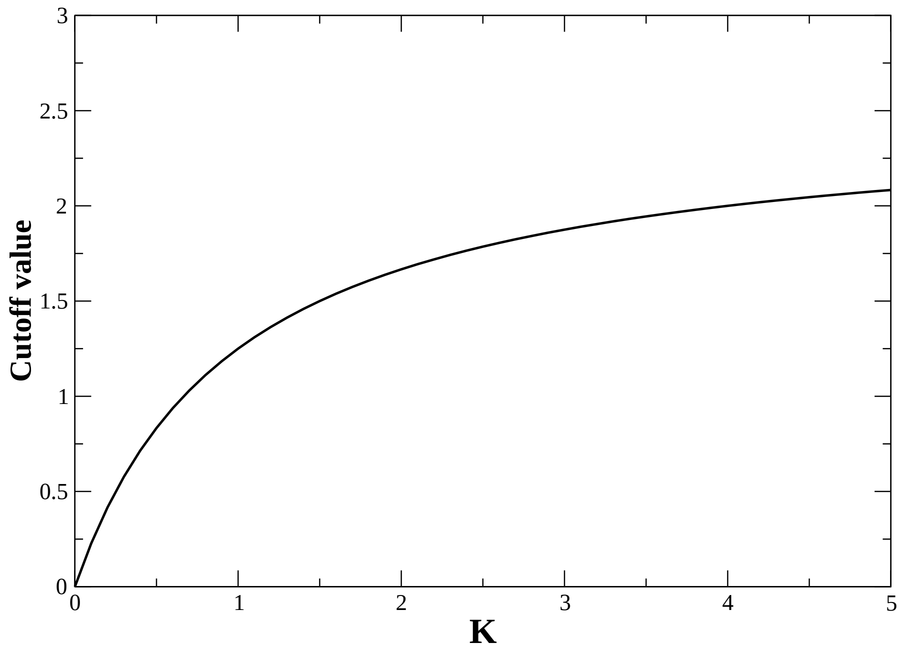


Figure 9.2: *The cutoff values for agent i as a function of K. For each value of K, if agent i's cost of computation falls below the line then it will compute. Otherwise, it does not. In this example, $V = 5$.*

that is, agent i will compute whenever its cost of computing of less than $VK/2(K+1)$. The same cutoff equilibrium holds for agent j also. Figure 9.2 displays how the cutoff changes as K approaches V.

9.2.4 Applying the Miscomputing Ratio

Given the derived tool set, it is possible to estimate what the miscomputing ratio is for any Vickrey auction. By using the Bayes-Nash equilibrium we can figure out, in expectation, what the miscomputing ratio will be given information about agents cost functions.

First, we have learned that if agents have free but limited computation, they will compute on their own problems only. What is the miscomputing ratio is this setting?

Theorem 30. *Let N be the set of bidding agents in a Vickrey auction. Assume that $|N| \geq 2$, and that each agent, i, has free computation and deadline D_i. Then, if the auctioneer is included in the social welfare measure, the miscomputing ratio is*

$R_{N\cup\{auc\}} = 1$.

Proof. To maximize social welfare, the global controller would select agent i such that $v_i(D_i) = \max_{j\in I} v_j(D_j)$, and only allow this agent to compute for D_i time steps. All other agents would be forbidden to compute. Agent i would submit a bid equal to $v_i(D_i)$ and all other agents would submit a bid equal to 0. The social welfare of this outcome, o^*, is $SW(o^*) = v_i(D_i)$. In equilibrium, as we have seen in Chapter 7, every agent would compute on its own problem until it reached its own deadline. Each agent would then submit a bid of an amount equal to its computed valuation. Agent i where $v_i(D_i) = \max_{j\in N} v_j(D_j)$ would be allocated the item. However, the price agent i would have to pay is equal to

$$v_k(D_k) = \max_{j\in N\setminus\{i\}} v_j(D_j).$$

The auctioneer's utility from this outcome is $u_{auc} = v_k(D_k)$. The social welfare is

$$SW(NE(o)) = (v_i(D_i) - v_k(D_k)) + v_k(D_k) + 0.$$

Therefore, the miscomputing ratio is

$$R_{N\cup\{auc\}} = \frac{v_i(D_i)}{(v_i(D_i) - v_k(D_k)) + v_k(D_k)} = 1.$$

□

What happens if we do not include the auctioneer in the computation of the ratio? It turns out that with limited computing, the miscomputing ratio can be arbitrarily bad.

Theorem 31. *Let N be the set of bidders in a Vickrey auction ($|N| \geq 2$). Assume that each bidder i has free but limited computing with deadline D_i. Then, the miscomputing ratio R_N can be infinity.*

Proof. Each agent has a dominant strategy which is to deliberate only on its own valuation problem until its deadline and to submit a bid equal to the valuation that it has obtained. That is, agent i submits a bid of $v_i(D_i)$. Without loss of generality, assume that $v_1(D_1) \geq v_2(D_2) \geq v_j(D_j)$ for all $j \neq 1, 2$. In equilibrium, agent 1 will win the auction and pay an amount of $v_2(T)$. Therefore, agent 1's utility is $u_1 = v_1(D_1) - v_2(D_2)$. Set $u_1 = \epsilon$. The utility for all other agents is $u_i = 0$ for $i \neq 1$. Therefore,

$$SW(o(\text{NE})) = \sum_{j \in N} u_j = \epsilon.$$

In order to maximize social welfare, the global controller would prohibit all agents expect for agent 1 from deliberating. Agent 1 would compute on its valuation problem until time D_1 and submit a bid of $v_1(D_1)$ while all other agents would submit a bid of 0. Agent 1 would win the item and pay an amount of 0. The utility for agent 1 is $u_1 = v_1(D_1) - 0 = v_1(D_1)$, while $u_i = 0$ for all $i \neq 1$. Therefore

$$SW(o^*) = \sum_{j \in N} u_j = v_1.$$

The miscomputing ratio, R_N, is

$$R_N = \frac{SW(o^*)}{SW(o(\text{NE}))} = \frac{v_1(D_1)}{\epsilon}.$$

As $\epsilon \to 0$ (that is, as the difference between the highest and second highest valuations decreases), $R \to \infty$. □

This is a negative result. Allowing agents to choose their computing strategies leads to an outcome that can be arbitrarily far from optimal.

In Chapter 7 we showed that in a Vickrey auction, if agents can compute for free but have deadlines then they have no incentive to strategically deliberate, while agents who incur a cost with deliberation can have incentive to deliberate on each others' valuation problems. This suggests that if there is a system designer who can control how the agents' computational capabilities are restricted, the designer should rather impose limits than costs. However, it turns out that computing costs can be adjusted so that the optimal miscomputing ratio ($R = 1$) is reached. This would mean that charging for computing is at least as desirable as imposing limits.

Theorem 32. *Computing cost functions can be used to motivate bidders to choose strategies that maximize social welfare.*

Proof. Consider the following example. Let there be 2 agents, agent 1 and agent 2, each with a deterministic performance profile. Assume that each agent has free but limited computing resources, and that the deadlines are D_1 and D_2. Each agent has

	compute	no
compute	$v_1(D_1) - v_2(D_2) - k,\ -k$	$v_1(D_1) - k,\ 0$
no	$0,\ v_2(D_2) - k$	0,0

Table 9.1: *Normal form game representation of the Vickrey auction where agent i has cost k for computing until D_i, and infinite cost if it computes for more than D_i. Agent 1 is the row player and agent 2 is the column player. Each agent would submit a bid that is equal to its computed valuation minus the cost spent to obtain the valuation.*

a dominant strategy, which is to deliberate on their own problem and submit a bid of $v_i(D_i)$. Assume that $v_1(D_1) > v_2(D_2)$. The equilibrium outcome is to award the item to agent 1 and have agent 1 pay an amount $v_2(D_2)$. Agent 1's utility is then $u_1 = v_1(D_1) - v_2(D_2)$ while agent 2's utility is $u_2 = 0$. To maximize social welfare the global controller would forbid agent 2 to deliberate, and thus agent 1 could get the item and need not pay anything. The maximum social welfare would be $u_1 = v_1(D_1)$. Therefore

$$R = \frac{v_1(D_1)}{v_1(D_1) - v_2(D_2)}$$

Next, consider the case where a simple cost function is introduced. Define

$$\text{cost}_i(t) = \begin{cases} k \text{ if } t \leq D_i; \\ \infty \text{ if } t > D_i; \end{cases}$$

for some constant k, $0 < k \leq v_2(D_2) \leq v_1(D_1)$. Any strategy that involves deliberating on the other agent's valuation problem is dominated as the computing action incurs a cost without improving the agent's overall utility. Thus, the remaining strategies are for the agents to compute only on their own valuation problems until the cost becomes too high, or not to compute at all. The game is represented in normal form in Table 9.1.

The sole Nash equilibrium is for agent 1 to compute and submit a bid of $v_1(D_1)$ and for agent 2 to not compute. The global controller trying to maximize the social welfare would force each agent to also follow those strategies. Therefore

$$R_N = \frac{v_1(D_1) - k}{v_1(D_1) - k} = 1.$$

□

In the proof the constant k can be made arbitrarily close to zero. Therefore, the maximum social welfare generated by the global controller in the costly computing setting can be made arbitrarily close to the maximum social welfare obtainable if computing resources are free.

9.3 Summary

In this chapter we proposed a way of measuring the impact on the social welfare that allowing agents to freely choose their own deliberation strategies has. We compared the social welfare obtained if agents were allowed to choose their own deliberation strategies freely, to the social welfare obtained if there was some global controller which dictated deliberation policies onto the agents so that the social welfare was maximized. Even in the global controller scenario we allowed agents to choose their own non-deliberation actions. This allowed us to isolate the actual impact that deliberation had on the social welfare. We called this ratio the *miscomputing ratio*.

We presented a Bayes-Nash equilibrium analysis of a Vickrey auction where the bidders' strategies include deliberation actions. The equilibrium showed how each agent's cost of computing determines the agent's strategy. The model allowed us to predict the overhead caused by miscomputing. It also allowed for the *design* of cost functions for computing. When including the auctioneer in the welfare measure, free computing with a deadline is an optimal way to control the cost of computing. If the auctioneer is not included in the ratio then the outcome can be arbitrarily far worse than in the case where computations are coordinated. However, by the careful design of cost functions, it is possible to provide appropriate incentives for bidders to choose deliberation policies that result in the optimal social welfare. This suggest that if a system designer can choose how to restrict the agents' computing, imposing costs instead of deadlines may be the right approach.

Chapter 10

Related Research

In this chapter we provide an overview of some of the research which is related to the work presented in this thesis. In Chapter 2 we discussed the related work on bounded rationality, and we do not cover it again in this chapter.

10.1 Information Acquisition and Mechanism Design

In the economics and game theory literature there has been some recent work on information acquisition and revelation and mechanism design (see, for example, [3, 7, 50, 70]). This work has mainly focused on studying the incentives to acquire information in different auction mechanisms. Most work assumes that an agent can only gather information about its own valuation. Perisco compares first-price and second-price auctions and shows that if agents have affiliated valuations then agents choose to acquire more information about their own valuation in a first price auction [92]. Compte and Jehiel compare the ascending price auction with the second price auction and show that there exist situations where an agent will pay to acquire information in an ascending auction but not in the second price auction [21, 22]. Rezende also studies an ascending price auction where agents are allowed to pay to acquire information about their own valuation [95]. He shows that in such a setting, in equilibrium, a bidder's best response function has a simple characterization which is independent of other agents' strategies. Bergemann and Välimäki study general mechanisms [8]. They are interested in mechanisms where agents acquire the efficient amount of information. They show that in the private value model the VCG mechanism is efficient in the sense that agents have incentive to acquire enough information so as to maximize the social welfare. However, in a purely common value setting agents either over-acquire or under-acquire information.

Rasmusen's work is the most similar to ours [94]. It assumes that agents do not know their valuations but must invest to learn them and are also able to invest in competitors value problems. Like us, he shows that in a second price auction, an agent may base its decision on whether to learn its true valuation on another agent's valuation, but his focus is on understanding behavior such as sniping that is commonly seen in different online auctions like eBay.

It should also be noted that the computer scientists have noticed that information gathering might have some form of strategic implication. Sandholm noted that even with a simple agent model the Vickrey auction may no longer have the dominant-strategy property [107]. However, this model was different from ours, since like most of the work in economics, it assumed that an agent was faced with a single decision as to whether to learn its own value at a cost or not.

Researchers have also investigated the effects of information gathering in mechanisms other than auctions. In particular, there is a body of work which studies the role of information gathering in contracts [28–30]. The model studied consists of a principal who can offer a contract and an agent who must decide whether to accept or not. The agent being offered a contract has uncertainty about its cost of production and has the possibility of investing some fixed cost in order to learn this information. The authors that the cost of gathering the information and the timing of when the information is gathered influence the equilibria. While this work studies a two agent negotiation problem, it is substantially different from our bargaining setting as it only allows the agent being made the offer to gather information on its own problem, as well as it is in a contract setting as opposed to a pure negotiation setting.

10.2 Computational Issues in Mechanism Design

There is a tension between classic game-theoretic solutions and tractable computational solutions, which is particularly highlighted by combinatorial allocation problems such as the combinatorial auction. Understanding the tension and the repercussions of limited computation at the mechanism center has been the focus of an area of work which falls in the category of computational mechanism design. In this section we describe some of this literature. It differs from the contributions in this thesis in that the literature mainly ignores computational issues of the agents, instead focusing on problems arising in the mechanism itself.

10.2.1 VCG Mechanisms and Computational Issues

The Vickrey-Clarke-Groves (VCG) family of mechanisms are the only social-welfare maximizing mechanisms which are incentive-compatible when agents have quasi-linear utilities. However, it has been noted by several researchers that there is a problem with the VCG mechanism when it is applied to complex combinatorial settings [69, 82, 83]. In combinatorial settings, like in the combinatorial auction, the optimization problems that the VCG mechanism must solve in order to determine the allocation and transfers are $\mathcal{NP}$-complete. Using approximation algorithms or heuristics in place of the exact algorithm can lead to the loss of some of the desirable properties of the mechanism, in particular incentive-compatibility. Rational agents may have incentive to manipulate their announcements in order to try and "fix" the approximation in their favor. The observation that approximation algorithms and heuristics can interfere with the game-theoretic properties of VCG mechanisms has led to the characterization of necessary properties for approximation algorithms for the mechanism center which retain strategy-proofness of the mechanism [52, 76].

In a different direction, people have proposed studying restrictions on the preferences of the agents in order to avoid situations where the allocation computation is computationally hard. Lehmann *et al* showed that if the preferences of the agents satisfy a single-mindedness condition (each agent is only interested in a single set of items) then a simple greedy allocation algorithm will guarantee incentive-compatibility [69]. Lavi *et al* showed that in general combinatorial auction settings, the only truthful auctions have to "almost" affine maximizers [67]. This is a rather negative result as these "almost" affine maximizers are computationally hard as exact optimization.

10.2.2 Winner Determination and Combinatorial Auctions

In spite of the computational complexity issues associated with the combinatorial auction, the practical importance of the problem has driven researchers to develop specialized algorithms for solving the winner determination problem [35, 108], or have proposed formulating the winner determination problem in such a way as so to take advantage of well developed optimization techniques [31, 51].

Researchers have looked for tractable special-cases for the winner determination problem. One approach, as mentioned in the previous subsection, is to restrict the preferences of the agents and then develop special purpose algorithms [69]. Another direction has been to look for special structure in the bids which lead to tractable instances [98].

10.2.3 Indirect Mechanisms

Indirect mechanisms have recently gotten a lot of attention from the research community [4, 90, 91]. Since indirect mechanisms run through multiple stages, they allow agents to provide incremental information about their preferences. Additionally, they can solve problems without requiring an agent to reveal all of its information to the mechanism. While indirect mechanisms have been promoted as a way of solving mechanism design problems for high-dimensional problems such as combinatorial auctions, they can also have desirable properties for settings where agents have valuation complexity [89]. In particular, indirect auctions for combinatorial settings have been presented as a way to simplify the meta-deliberation problems for the agents', with the goal of providing incentives for the "right" agents to compute for the "right" amount of time on the "right" problems, in the hopes of improving efficiency. While this approach appears to be promising, as the mechanism can help focus agents' attention on to relevant problems, to date the analysis has not considered scenarios where agents may strategically-deliberate.

10.3 Preference Elicitation and Communication Complexity

Preference elicitation is the process of asking queries to determine an agent's preferences. This is a well established field in artificial intelligence (see, for example, [13, 17, 120]), and has recently emerged as an important research topic in the multiagent system/ecommerce community [24, 26]. One strand of this work has focused on the problem of finding the minimal number of queries necessary in order to come up with the optimal allocation.

Preference elicitation – the process of making queries to determine an agent's preferences – is another active research area (see for example, [10, 14, 25, 93, 125]). Much of this work has focused on the combinatorial auction problem. In a combinatorial auction agents have 2^n bundles of items that they can express preferences over (n is the number of items being auctioned). By cleverly eliciting information from the agents, it may be possible to avoid having the agents specify a preference for every bundle [23, 25]. This form of preference elicitation is indirectly related and complements the problem of having agents invest computing resources in order to figure out their preferences. By carefully structuring the (order of the) queries, the mechanism might be able to guide agents in their search for optimal deliberation policies.

A very similar area of research has been directed toward understanding the communication complexity of different mechanisms. Nisan and Segal [84] showed that for combinatorial auctions every preference elicitation protocol which guarantees that the optimal allocation will be found must use an exponential number of queries in the worst case. Blumrosen and Nisan [11] study auctions where the bidders are restricted in the amount of information they can transmit. This research complements our work as it studies other restrictions on agents' resources.

Not all the preference elicitation literature research has focused on combinatorial auctions. Voting is a way to aggregate multiple agents' preferences into a single outcome. Several researchers have studied the computational complexity of eliciting information from voters [5, 26], and have shown that knowing when a preference elicitation process should terminate is NP-complete for many voting protocols. This is an additional issue on top of the problem of agents having to determine their preferences beforehand.

10.4 Bidding Agents and Proxy Agents

Numerous researchers have studied the problem of designing bidding agents to represent users in different market settings. The simplest form of agent is the *proxy agent* [4, 91]. Proxy agents have traditionally been proposed for use in iterative mechanisms and work in the following way. The bidder tells the agent its preferences (not necessarily truthfully) for the items in the auction. The proxy agent then represents the bidder in the auction, and follows a prescribed strategy, based on the preferences reported to it. The use of proxy agents simplifies the strategic problem of the bidder, as the bidder need only be concerned with revealing its preferences, and not with the details of the complex iterative mechanism. The agents which we proposed in this dissertation are not proxy agents, as they act autonomously from the user. They are not told the user's preferences, but instead go out and actively determine them.

The term bidding agent is usually reserved for more sophisticated agents. Some bidding agents are mobile,and are general enough to move from auction to auction [49]. Other agents are specialized and are based at one specific auction. A prime motivator in the area of bidding agent design has been the Trading Agent Competition (TAC) [41, 116, 122]. Teams compete by designing agents who, given users' preferences, try to maximize their utility by participating in complex market situations involving multiple simultaneous auctions of different types. These complex domains mean that the agents are not able to game-theoretically find optimal strategies, and thus use complicated learning and optimization techniques in order to adapt to the ever changing environment.

There are several key differences between our computationally limited agents and the agents designed for settings like those found in TAC. First, the TAC agents are not computationally limited in our sense. Instead, they are explicitly told the preferences of the users they represent, and their goal is to maximize their utility by satisfying the preferences. In our model of an agent, the agent is responsible for both computing the preferences and then satisfying them. Second, the motivation behind TAC is to encourage the development of sophisticated agents which can participate in complicated market domains, consisting of multiple simple mechanisms. The motivation behind our work is to study and understand the strategic behavior of agents in different domains, and then, if possible, design the market domains so as to reduce the strategic overhead placed on the agents. Thus, in the long term our goal is to have (potentially complicated) mechanisms so that the agents can be of a simple design.

10.5 Summary

In this chapter we provided an overview of some of the work in the area of information acquisition in mechanisms, computational mechanism design and bidding agents. The work presented in this thesis differs from most of the literature discussed. It is most similar to the work on information acquisition in mechanisms, though we present a more sophisticated model of information acquisition which uncovers new strategic behavior. The literature in the area of computational mechanism design focuses on computational limitations, but generally from the perspective of the mechanism center, as opposed to the agents. Preference elicitation tries to limit the amount of information that has to be transmitted to the center, thus reducing the overhead on both agents and the center. We believe that techniques from preference elicitation can complement our work by helping agents focus their attention on the "right" problems. Finally, we discussed some of the work on bidding agent design. There has been substantial work designing agents for complex dynamic markets. These agents, however, are not computationally limited like our model. One of the philosophies espoused in this dissertation is that the strategic burden on the agents should be reduced, if possible, by good mechanism design. This is orthogonal with much of the work on bidding agents as it is usually assumed that the mechanism is given and unmodifiable.

Chapter 11

Conclusions and Future Work

This dissertation set out to study the impact that computational limitations have on the strategic behavior of agents in multiagent negotiation settings. The thesis statement was

> By using a fully normative model of bounded rationality it is possible to incorporate agents' deliberation actions into a game-theoretic setting.
>
> - This will allow one to formally study the impact that limited computing resources has on agents' strategic behavior.
> - This will provide a foundation for game theory and mechanism design for computationally limited agents.

In this final chapter we summarize the contributions of this dissertation. We also describe some new directions for future work.

11.1 Contributions

The main contributions of this thesis are:

A normative model of bounded rationality. We presented a model for a computationally-limited agent (Chapter 2). We defined a computationally limited agent as having cost functions, anytime algorithms, and performance-profile deliberation controllers. In particular, we introduced the performance profile tree, a fully normative method for determining deliberation control policies. We showed that the performance profile tree was feasible in practice and lead to better deliberation control decisions, compared to other approaches from the literature (Chapter 3).

A formal game theoretic model for computationally limited agents. We proposed incorporating the deliberation actions of agents in a game theoretic setting. In Chapter 5 we presented definitions of strategies for agents with computational limitations, and introduced the deliberation equilibrium. We also discussed new strategic behavior which may arise among computationally limited agents. In particular, we introduced the phenomenon strategic deliberation.

Analysis of different negotiation mechanisms. In Chapters 6 and 7 we analyzed different negotiation protocols, using the deliberation equilibrium as the solution concept. In Chapter 6 we studied two different bargaining protocols; an ultimatum game and an alternating-offers model. We determined the deliberation equilibria for different variations of these games, and proposed algorithms that agents can use to solve for the deliberation equilibria strategies.

In Chapter 7 we studied four types of commonly used and studied auctions mechanisms; the first-price sealed bid auction, the ascending auction, the Vickrey auction and the generalized Vickrey auction. We observed that neither the Vickrey, ascending, nor generalized Vickrey auction preserved their dominant-strategy equilibria when the bidders were computationally limited. Through a series of experiments we studied the impact that asymmetry of cost and performance profiles has on agents' strategic behavior.

Mechanism design principles for computationally limited agents. In Chapter 8 we proposed a set of desiderata for mechanisms designed for computationally-limited agents. We argued that mechanisms should have good economic properties as well as good deliberative properties. In particular, we proposed that mechanisms should not directly solve the deliberation problems of the agents (non-deliberative), that strategic deliberation should not occur in equilibrium (deliberation-proof), and that agents should not have incentive to misrepresent their (partially) computed results (non-deceiving). We showed that these desiderata are orthogonal, and that tradeoffs in design must be tolerated.

In Chapter 9 we took a different direction, and proposed a new tool for measuring the impact that strategic deliberation has on the efficiency of different mechanisms. We introduced the miscomputing ratio which measures the difference in the social welfare of settings where agents are freely allowed to choose their deliberation strategies, against settings where a social-welfare maximizing oracle dictates which deliberation policies agents must follow. This approach provided a way of isolating the deliberation policies from the rest of the strategic actions that agents may take, and allowed one to compare efficiency across different mechanisms.

11.2 Directions for Future Work

This dissertation opens up new interesting directions for future research. In this section we outline some directions which arise from work done in this thesis.

11.2.1 Overcoming the Impossibility Result:

In Chapter 8 we proposed a set of properties which we believed that mechanisms designed for computationally limited agents should exhibit. In particular, we believed that mechanisms should be *non-deliberative*, (i.e. the mechanism should not solve the deliberation problems of agents), that the mechanism should be *deliberation-proof* (i.e. the agents should not have incentive to deliberate on competitors' problems) and that agents should not have incentive to deceive others about their computed results. Unfortunately, it was shown that these properties are, together, not attainable. Therefore, at least one property must be relaxed

One possible approach for avoiding the impossibility result is to ignore the strategy-proof property and just use the mechanisms already developed for fully rational-agents. However, it might be possible to go beyond that. For example, if one agent can solve certain problems easily and cheaply it might be effective, from a social-welfare perspective, to have that agent solve problems for all agents who need solutions for those problems. There are some interesting incentives issues that need to be addressed in such a mechanism. For example, appropriate incentives will be needed to get the computing agent to solve the "right" problems, and to share the information with all agents to which it pertains.

A different approach would be to relax the non-deliberation property of the mechanism. It might be possible to design mechanisms which, given some deliberation information by the agents, is able to help guide the agents in their deliberation decisions. This approach may reduce strategic deliberation, as long as agents believe that the mechanism is making deliberation decisions for them which are in their own best interest. Some problems that need to be addressed before such an approach can be used include:

- What sort and how much information should the agents reveal to the mechanism?
- Is there a minimal amount of information that needs to be revealed to the mechanism so that strategic-deliberation disappears?
- What sort of incentives are required before agents will truthfully reveal this information?
- What type of feedback should the mechanism provide the agents?

11.2.2 Costly Preference Elicitation in Multi-Item Auctions

In Chapter 9 we introduced the miscomputing ratio as a way of measuring the impact on social welfare that allowing agents to freely compute on any problem has on the social welfare. Our analysis has focused on single item auctions, however there are many interesting problems which also arise in multi-item auctions. In particular, it has been proposed that indirect mechanisms can lead to increased social welfare in settings where there is costly preference elicitation [88]. The approach taken in this dissertation presents a formal method for studying this question.

11.2.3 Learning Features that are Important for Deliberation Control Decision Making

The performance profile tree is a fully normative deliberation control procedure, which can, in theory, store any and all information that might be of importance to the agent when it comes to making deliberation control decisions. However, in practice it is unreasonable to assume that all such information can be captured in a performance profile tree, nor is it reasonable to assume that all information is actually of use to an agent. This was illustrated in the experimental results in Chapter 3, where using additional features to solution quality did not substantially improve the deliberation control. Combining our deliberation control method with some form of learning techniques in order to determine what are important features (for example, as described in [15]) may, in practice, bring performance-profile deliberation control closer to optimal while remaining feasible.

11.3 Summary

We started this dissertation with a motiving quote from Herbert Simon, outlining a motivation for studying bounded rationality:

> "What are the economic consequences of participants using certain procedures and not others? In what respect are current economic models deficient in the assumptions they make about reasoning procedures?" [101]

This dissertation has been an attempt to answer these questions. We have demonstrated that it is possible to incorporate reasoning procedures of agents into a game-theoretic framework, and have shown that some current economic models (i.e. bargaining protocols and auction mechanisms) exhibit different properties when used by computationally limited agents as opposed to fully rational agents. It is our hope that the models and ideas presented in this thesis will serve as a foundation for future work in the study of bounded rationality.

Bibliography

[1] H. M. Amman, D. A. Kendrick, and J. Rust, editors. *Handbook of Computational Economics*. Elsevier Science, 1996.

[2] Luzi Anderegg and Stephan Eidenbenz. Ad hoc-VCG: a truthful and cost-efficient routing protocol for mobile ad hoc networks with selfish agents. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 245–259, San Diego, September 2003.

[3] Leandro Arozamena and Estelle Cantillon. Investment incentives in procurement auctions. Working paper, August 2000.

[4] Lawrence M Ausubel and Paul Milgrom. Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1, 2002. No. 1, Article 1.

[5] John J. Bartholdi, III, Craig A. Tovey, and Michael A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[6] Eric B Baum and Warren D Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.

[7] Jean-Pierre Benoit and Juan Dubra. Information revelation in auctions. Working paper, February 2003.

[8] Dirk Bergemann and Juuso Välimäki. Information acquisition and efficient mechanism design. *Econometrica*, 70:1007–1034, 2002.

[9] Kenneth Binmore. Bargaining and coalitions. In Al Roth, editor, *Game-Theoretic Models of Bargaining*, pages 269–304. Cambridge University Press, 1985.

[10] Avrim Blum, Jeffrey Jackson, Tuomas Sandholm, and Martin Zinkevich. Preference elicitation and query learning. *Journal of Machine Learning Research*, 5:649–667, 2004. Early version in COLT-03.

[11] Liad Blumrosen and Noam Nisan. Auctions with severely bounded communication. In *FOCS*, pages 406–415, 2002.

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

[13] Craig Boutilier, Ronen Brafman, Christopher Geib, and David Poole. A constraint-based approach to preference elicitation and decision making. In *AAAI Spring Symposium on Qualitative Decision Theory*, 1997.

[14] Craig Boutilier, Relu Patrascu, Pascal Poupart, and Dale Schuurmans. Regret-based utility elicitation in contraint-based decision problems. Working paper, 2004.

[15] Justin Boyan and Andrew Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1(2), 2000.

[16] J. L. Bresina. Heuristic-biased stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, (AAAI'96)*, pages 271–278, 1996.

[17] U Chajewwska, Daphne Koller, and R Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 363–369, Austin, TX, 2000.

[18] Kalyan Chatterjee and Ching Chyi Lee. Bargaining and search with inconplete information about outside options. *Games and Economic Behavior*, 22:203–237, 1998.

[19] Vincent Cicirello and Steven Smith. Amplification of search performance through randomization of heuristics. In *Principles and Practice of Constraint Programming – CP 2002*, pages 124–138, 2002.

[20] E H Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[21] Olivier Compte and Philippe Jehiel. Auctions and information acquisition: Sealed-bid or dynamic formats?, 2001. Working Paper.

[22] Olivier Compte and Phillipe Jehiel. On the virtues of the ascending price auction: New insights in the private value setting. unpublished, December 2000.

[23] Wolfram Conen and Tuomas Sandholm. Minimal preference elicitation in combinatorial auctions. In *IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms*, pages 71–80, Seattle, WA, August 2001.

[24] Wolfram Conen and Tuomas Sandholm. Preference elicitation in combinatorial auctions: Extended abstract. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 256–259, Tampa, FL, October 2001. A more detailed description of the algorithmic aspects appeared in the IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms, pp. 71–80.

[25] Wolfram Conen and Tuomas Sandholm. Partial-revelation VCG mechanism for combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 367–372, Edmonton, Canada, 2002.

[26] Vincent Conitzer and Tuomas Sandholm. Vote elicitation: Complexity and strategy-proofness. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 392–397, Edmonton, Canada, 2002.

[27] Vincent Conitzer and Tuomas Sandholm. Computational criticisms of the revelation principle. In *Conference on Logic and the Foundations of Game and Decision Theory (LOFT)*, Leipzig, Germany, July 2004.

[28] J Cremer and F Khalil. Gathering information before signing a contract. *American Economic Review*, v82, n3:566–578, 1992.

[29] J Cremer, F Khalil, and J Rochet. Contracts and productive information gathering. *Games and Economic Behavior*, v25, n2:174–193, 1998.

[30] J Cremer, F Khalil, and J Rochet. Strategic information gathering before a contract is offered. *Journal of Economic Theory*, v81, n1:163–200, 1998.

[31] Sven de Vries and Rakesh Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 2003.

[32] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, CA, 1991.

[33] Stephan Eidenbenz and Paolo Santi. Commit: A sender-centric truthful and energy-efficient routing protocol for ad hoc networks. Tech. Rep. IIT-TR01/2004, Istituto di Informatica e Telematica, Pisa - Italy, January 2004.

[34] Shaheen Fatima, Michael Wooldridge, and Nicholas Jennings. Multi-issue negotiation under time constraints. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, Bologna, Italy, July 2002.

[35] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 548–553, Stockholm, Sweden, August 1999.

[36] Alan Garvey and Victor Lesser. A survey of research in deliberative real-time artificial intelligence. *Real-Time Systems*, 6:317–347, 1994.

[37] A Gibbard. Manipulation of voting schemes. *Econometrica*, 41:587–602, 1973.

[38] Irving Good. Twenty-seven principles of rationality. In V Godambe and D Sprott, editors, *Foundations of Statistical Inference*. Toronto: Holt, Rinehart, Winston, 1971.

[39] Joshua Grass and Shlomo Zilberstein. A value-driven system for autonomous information gathering. *Journal of Intelligent Information Systems*, 14(1):5–27, 2000.

[40] J Green and J-J Laffont. Characterization of satisfactory mechanisms for the revelation of preferences for public goods. *Econometrica*, 45:427–438, 1977.

[41] Amy Greenwald. Bidding marginal utility in simultaneous auctions. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 1463–1464, August 2003.

[42] Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

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

[44] Michael Horsch and David Poole. Estimating the value of computation in flexible information refinement. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 297–304, 1999.

[45] Eric Horvitz. *Computation and actions under bounded resources*. PhD thesis, Stanford University, 1990.

[46] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.

[47] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, 1987.

[48] 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.

[49] Qianbo Huai and Tuomas Sandholm. Mobile agents in an electronic auction house. *IEEE Internet Computing*, 4(2):80–86, 2000. Special issue on Agent Technology and the Internet.

[50] Matthew Jackson. Efficiency and information aggregation in auctions with costly information. *Review of Economic Design*, 8(2), 2003.

[51] Terence Kelly. Generalized knapsack solvers for multi-unit combinatorial auctions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, New York, July 2004.

[52] 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.

[53] 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.

[54] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, March 1990.

[55] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Symposium on Theoretical Aspects in Computer Science*, 1999.

[56] Sarit Kraus. *Strategic Negotiation in Multi-Agent Environments.* MIT Press, 2001.

[57] Sarit Kraus, Jonathan Wilkenfeld, and Gilad Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75:297–345, 1995.

[58] Vijay Krishna. *Auction Theory.* Academic Press, 2002.

[59] 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.

[60] Kate Larson and Tuomas Sandholm. Costly valuation computation in auctions. In *Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, pages 169–182, Siena, Italy, July 2001.

[61] Kate Larson and Tuomas Sandholm. An alternating offers bargaining model for computationally limited agents. In *International Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002.

[62] Kate Larson and Tuomas Sandholm. Bidders with hard valuation problems. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 160–161, Bologna, Italy, July 2002. Early version: Computationally Limited Agents in Auctions. International Conference on Autonomous Agents 2001,

Workshop on Agent-based Approaches to B2B, pp. 27–34, Montreal, Canada, May 28th.

[63] Kate Larson and Tuomas Sandholm. Miscomputing ratio: The social cost of selfish computing. In *International Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, 2003. Early version appeared in the AAAI-02 workshop on Game-Theoretic and Decision-Theoretic Agents, Edmonton, Canada.

[64] Kate Larson and Tuomas Sandholm. Experiments on deliberation equilibria in auctions. In *International Conference on Autonomous Agents and Multi-Agent Systems*, pages 394–401, New York, NY, USA, July 2004.

[65] Kate Larson and Tuomas Sandholm. Strategic deliberation and truthful revelation: An impossibility result. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, New York, NY, May 2004. Short paper.

[66] Kate Larson and Tuomas Sandholm. Using performance profile trees to improve deliberation control. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 73–79, San Jose, CA, USA, July 2004.

[67] Ron Lavi, Ahuva Mu'Alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2003.

[68] Y. H. Lee, K. Bhaskaran, and M. Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52, 1997.

[69] Daniel Lehmann, Lidian Ita O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002. Early version appeared in ACMEC-99.

[70] Vlad Mares and Ronald Harstad. Private information revelation in common value auctions. Working paper, November 2002.

[71] Andreu Mas-Colell, Michael Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[72] John McCarthy. Programs with common sense. In *Proceedings of the Symposium on the Mechanization of Thought Processes*, Teddington, England, 1958.

[73] Richard McKelvey, Andrew McLennan, and Theodore Turcoy. *Gambit: Software Tools for Game Theory*. The Gambit Project, 2002.

[74] Paul Milgrom. Auctions and bidding: A primer. *Journal of Economic Perspectives*, 3(3):3–22, 1989.

[75] 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.

[76] Ahuva Mu'alem and Noam Nisan. Truthful approximate mechanisms for restricted combinatorial auctions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 379–384, Edmonton, Canada, July 2002.

[77] Roger Myerson. Optimal auction design. *Mathematics of Operation Research*, 6:58–73, 1981.

[78] Roger Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.

[79] John Nash. The bargaining problem. *Econometrica*, 18:155–162, 1950.

[80] John Nash. Equilibrium points in n-person games. *Proc. of the National Academy of Sciences*, 36:48–49, 1950.

[81] Alan Newell. The knowledge level. *Artificial Intelligence*, 18(1):82–127, 1982.

[82] 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.

[83] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001. Early version in STOC-99.

[84] Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting Lindahl prices, 2003. Working Paper (version: March 2003).

[85] Martin Osborne and Ariel Rubinstein. *Bargaining and Markets*. Academic Press, New York, 1990.

[86] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[87] Christos Papadimitriou. Algorithms, games and the Internet. In *STOC*, pages 749–753, 2001.

[88] David Parkes. Auction design with costly preference elicitation. Submitted for publication, January 2003.

[89] 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.

[90] David C. Parkes and Lyle Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 74–81, Austin, TX, August 2000.

[91] David C. Parkes and Lyle Ungar. Preventing strategic manipulation in iterative auctions: Proxy-agents and price-adjustment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 82–89, Austin, TX, August 2000.

[92] Nicola Perisco. Information acquisition in auctions. *Econometrica*, 68(1):135–148, January 2000.

[93] Radcliffe seminar on revealed and latent preferences: Economic and computational approaches. http://www.eecs.harvard.edu/˜parkes/radcliffe.html, May 2004.

[94] Eric Rasmusen. Strategic implications of uncertainty over one's own private value in auctions. Working paper, January 2003.

[95] Leonardo Rezende. Mid-auction information acquisition. Working paper, Stanford University, November 2002.

[96] Jeffrey S Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[97] Al Roth. *Axiomatic Models of Bargaining*. Springer Verlag, 1979.

[98] Michael H. Rothkopf, Aleksandar Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[99] Tim Roughgarden and Eva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, March 2002.

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

[101] Ariel Rubinstein. *Modeling Bounded Rationality*. MIT Press, 1998.

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

[103] Stuart Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 1:1–36, 1995.

[104] Stuart Russell and Eric Wefald. *Do the right thing: Studies in Limited Rationality*. The MIT Press, 1991.

[105] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[106] 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.

[107] 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.

[108] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, January 2002. First appeared as an invited talk at the First International Conference on Information and Computation Economies, Charleston, SC, Oct. 25–28, 1998. Extended version appeared as Washington Univ., Dept. of Computer Science, tech report WUCS-99-01, January 28th, 1999. Conference version appeared at the International Joint Conference on Artificial Intelligence (IJCAI), pp. 542–547, Stockholm, Sweden, 1999.

[109] 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.

[110] Tuomas Sandholm and Victor R Lesser. Leveled commitment contracts and strategic breach. *Games and Economic Behavior*, 35:212–270, 2001. Special issue on AI and Economics. Early versions appeared as *Advantages of a Leveled Commitment Contracting Protocol* in the proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 126–133, Portland, OR, 1996, and as a University of Massachusetts at Amherst, Computer Science Department technical report 95-72.

[111] Tuomas Sandholm and Nir Vulkan. Bargaining with deadlines. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 44–51, Orlando, FL, 1999.

[112] Tuomas Sandholm and Yunhong Zhou. Surplus equivalence of leveled commitment contracts. *Artificial Intelligence*, 142:239–264, 2002. Early versions appeared in ICMAS-00 and in the AAAI-99 Workshop on Negotiation: Settling conflicts and identifying opportunities.

[113] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.

[114] John Maynard Smith. The theory of games and evolution in animal conflict. *Journal of Theoretical Biology*, 47:209–221, 1974.

[115] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[116] Peter Stone and Amy Greenwald. The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research*, 2005. To appear.

[117] Hal Varian. Mechanism design for computerized agents. In *Usenix Workshop on Electronic Commerce*, New York, July 1995.

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

[119] Bernhard von Stengel. Computing equilibria for two–person games. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory*, volume 3. North–Holland, 2001.

[120] Ha Vu and Peter Haddawy. Towards case-based preference elicitation: Similarity measures on preference structures. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 193–201, 1998.

[121] Eric Weisstein. *CRC Concise Encyclopedia of Mathematics.* CRC Press, 2nd edition, 2002.

[122] Michael Wellman, Daniel Reeves, Kevin Lochner, and Yevgeniy Vorobeychik. Price prediction in a trading agent competition. *Journal of Artificial Intelligence Research*, 21:19–36, 2004.

[123] Elmar Wolfstetter. Auctions: An introduction. Technical report, Humboldt University of Berlin, 1994.

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

[125] Martin Zinkevich, Avrim Blum, and Tuomas Sandholm. On polynomial-time preference elicitation with value queries. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 176–185, San Diego, CA, 2003.