Open Mechanism Design: Ensuring and Verifying the Strategyproofness of Mechanisms in Open Environments

A dissertation presented

by

Laura Suh Young Kang

to

The School of Engineering and Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Harvard University

Cambridge, Massachusetts

September 2008

Thesis advisor

David C. Parkes

Author

Laura Suh Young Kang

Open Mechanism Design:
Ensuring and Verifying the Strategyproofness of Mechanisms in Open Environments

Abstract

The wide-spread availability of high-speed internet access has brought about a migration of computation from local company-owned servers and personal computers to shared resources or on-demand platforms. Rather than performing computation on local machines, more organizations are utilizing pooled computational resources, e.g., *grid computing*, or software provided as an on-demand service, e.g., *cloud computing*. These environments are *open* in that no single entity has control or full knowledge of outcomes. Entities are owned and deployed by different organizations or individuals, who have conflicting interests. These entities can be modeled as self-interested agents with private information. The design of systems deployed in open environments must be aligned with the agents' *incentives* to ensure desirable outcomes. I propose *open mechanism design*, an open infrastructure model in which anyone can own resources and deploy mechanisms to support automated decision making and coordination amongst self-interested agents. This model allows for a decentralized control structure, respecting the autonomy of resource owners and supporting innovation and competition. Each mechanism can adopt its own design goals. This vision of an open infrastructure to promote automated and optimal decision mak-

ing between multiple parties encompasses and expands on much of the thinking that underlies agent-mediated e-commerce and on-demand computing systems. The role of the infrastructure in an open setting – as it applies to resource allocation mechanisms – is to ensure or verify the property of *strategyproofness*, namely, whether a self-interested agent can maximize her utility by simply reporting information about her preferences for different resource allocation truthfully. I present two approaches, with the role of the infrastructure slightly different in each. The first approach considers passive verification of the strategyproofness of mechanisms, while the second approach considers active enforcement of strategyproofness in decentralized auctions for dynamic resource allocation. I present monotonic resource estimation and pricing algorithms that can be used to ensure strategyproofness of a mechanism and empirical results from simulations using data collected from the Crimson Grid.

Contents

List of Figures

List of Tables

Citations to Previously Published Work

Large portions of Chapters 4 and 5 have appeared in the following papers:

"Passive Verification of the Strategyproofness of Mechanisms in Open Environments", L. Kang and D. C. Parkes, Proceedings of the 8th International Conference on Electronic Commerce, ACM Press, August 2006;

"A Decentralized Auction Framework to Promote Efficient Resource Allocation in Open Computational Grids", L. Kang and D. C. Parkes, Proceedings of the Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon+IBC 2007), San Diego, 2007.

Discussion on the *EGG* project can be found in the following paper:

"EGG: An Extensible and Economics-inspired Open Grid Computing Platform", J. Brunelle, P. Hurst, J. Huth, L. Kang, C. Ng, D. Parkes, M. Seltzer, J. Shank, S. Youssef, Proceedings of 3rd International Workshop on Grid Economics and Business Models (GECON'06), Singapore, 2006.

Acknowledgments

This dissertation would not have been possible without the support and guidance of numerous people. I owe my gratitude to all those who have made my doctoral study one that I will cherish forever.

My most sincere gratitude goes to the greatest advisor ever, Prof. David Parkes, for giving me the opportunity to begin and complete this journey. He encouraged my every step and inspired me to keep focused. His advice and insightful comments helped me greatly at all stages of research and writing.

I also offer great thanks to the other members of my dissertation committee, Prof. Margo Seltzer and Prof. Avi Pfeffer. Special thanks to Susan Wieczorek for ensuring that all my paperwork was complete and deadlines were met. She was knowledgeable and helpful whenever I had a concern or question. I would also like to thank Dr. Joy Sircar for allowing me to collect data on the Crimson Grid.

I would like to express my gratitude to all my friends for making my graduate school experience so enjoyable and to the members of the EconCS Research Group and my colleagues at Harvard University for collaboration and fruitful discussion.

I offer my deepest admiration and gratitude to my parents, In Seok and Ae Sun, and my siblings, Erin, Christine, and Sean, for their patience, support, prayer, and unconditional love.

Finally, I thank God for giving me the strength and resources to complete my thesis.

Now to him who is able to do immeasurably more than all we ask or imagine, according to his power that is at work within us (Ephesians 3: 20).

Dedicated to my father, Dr. In Seok Kang.

Chapter 1

Introduction

1.1 Motivation

The wide-spread availability of high-speed internet access has brought about a migration of computation from local company-owned servers and personal computers to shared resources or on-demand platforms. Rather than performing computation on local machines, more organizations are utilizing pooled computational resources, e.g., *grid computing*, or software provided as an on-demand service, e.g., *cloud computing*. These trends have increased the ease of collaboration and sharing and reduced the cost of installation and maintenance for end-users. These trends have influenced e-commerce as many services offer means for outsourcing customer support, marketing, and sales, further lowering the barriers to entry. They have also offered a way to solve computationally intensive problems such as climate modeling, earthquake simulation, protein folding, and financial modeling. These trends have not only affected intra-organizational resource sharing, but also offer means of using IT resources opti-

mally within an organization (for example, Crimson Grid [76]). Amazon.com, IBM, and Google have all introduced cloud computing platforms. Well-known grid projects include *Enabling Grids for E-sciencE (EGEE)* [4], *World Community Grid* [1], and *SETI@home* [38]. Many infrastructures, e.g., Berkeley Open Infrastructure for Network Computing (BOINC) [3] and gLite [4], have been developed to act as middlemen between the end-users and geographically distributed computational resources, taking charge of scheduling and resource allocation, and ensuring good properties for the end-users.

These infrastructures provide an attractive application area for market mechanisms because of the distributed nature of the ownership and use of resources in these systems and the presence of users and resource owners with conflicting interests. In market mechanisms, prices coordinate decision making both within and between organizations.

These environments are *open* in that no single entity has control or full knowledge of outcomes. Entities are owned and deployed by different organizations or individuals, who have conflicting interests. These entities can be modeled as self-interested agents with private information, e.g. user's value or resource state of a machine. The design of systems deployed in open environments must be aligned with the agents' *incentives* to ensure desirable outcomes. The goal in these environments is to promote automated and optimal decision making between multiple parties. In open environments, cooperation and trust cannot be assumed. Infrastructures should allow for autonomy for resource owners, respecting their policy requirements, and support competition and innovation.

The study of *Mechanism Design* from microeconomic theory provides useful tools for designing and analyzing systems with self-interested agents. However, the classic model of mechanism design assumes a single mechanism with a central planner with full enforcement power and agreement on design goals. In environments with the scope and scale of grid computing or cloud computing, no entity has full control and a single mechanism may not be computationally scalable, able to capture the entire scope of a problem, or universally agreed upon. A general consensus on the goal of the mechanism may not exist.

Then how can the model of classic mechanism design be adapted to open environments? I propose *open mechanism design*, an open infrastructure model in which anyone can own resources and deploy mechanisms to support automated decision making and coordination amongst self-interested agents. This model allows for a decentralized control structure, respecting the autonomy of resource owners and supporting innovation and competition. Each mechanism can adopt its own design goals. This vision of an open infrastructure to promote automated and optimal decision making between multiple parties encompasses and expands on much of the thinking that underlies agent-mediated e-commerce and on-demand computing systems.

Multiple entities (e.g., firms, individuals, organizations, network services) can deploy decision mechanisms that can be used, for instance, to coordinate purchasing decisions, allocation tasks or resources, schedule bandwidth, or form coordinated plans of actions. The role of the infrastructure in an open setting – in as much as it applies to resource allocation mechanisms – is to verify via observation and intermediation and/or enforce via constraints and partial control that desirable properties

are maintained by deployed mechanisms.

The key property that the infrastructure ensures and/or verifies throughout this work is whether a given mechanism is *strategyproof* or *truthful*. Strategyproofness is a central property in the design of mechanisms in an open setting, allowing participants to maximize their individual benefit by reporting truthful private information about preferences and capabilities. Truth-revelation is a *dominant strategy* equilibrium in these mechanisms, which can be computed without distributional assumptions, and thus are robust. Strategyproofness is also desirable from a computational standpoint because it reduces the need to model and reason about other agents, and provides a simple and straightforward strategy to an agent. For these reasons, strategyproof mechanisms should also encourage the adoption of agent-mediated decision making.

I present two approaches, with the role of the infrastructure slightly different in each. In the first approach, the infrastructure *verifies* whether a given mechanism is strategyproof using a *light-weight passive* verifier that does not make specific domain assumptions in a setting where static mechanisms are repeatedly used. Both strategyproof and non-strategyproof mechanisms can be deployed, and participants are notified when a violation to strategyproofness is found. Passive verification algorithms leverage general characterizations of strategyproof mechanisms. The infrastructure treats a mechanism as a black box merely observing the inputs, allocation and payments. Verification takes place *after* the allocation and payments are decided. When a violation is detected, the infrastructure intervenes and prevents an allocation from being implemented.

In the second approach, the infrastructure has a more active role: it *ensures* that

the deployed mechanisms are strategyproof by imposing constraints on each component of a mechanism. These constraints are derived from the characterization of strategyproofness developed specifically for the domain of *online* or *dynamic* mechanisms. In online mechanisms, agents arrive and depart at different times and an additional dimension of manipulation, namely time, must be considered to ensure strategyproofness.

In contrast to the first approach where the verifier requires that the rules of a given mechanism do not change, the second approach allows for more flexibility and lets the rules of a mechanism change over time. In the first approach, a mechanism is not published but only the input-output sequence is checked, whereas in the second approach, components of a mechanism must be published in a form specified by the infrastructure that allows for checking. In the second approach, no violations of strategyproofness are possible in that any violation will be detected *before* inputs to a mechanism are received. The second approach makes a stronger assumption about the design space as mechanisms must make use of price tables to publish prices, for example, but still allows for flexibility in defining price schedules.

My dissertation can be divided up into two parts, based on the two approaches described above:

Part I: (Chapter 4) Algorithms for passively verifying whether a mechanism used in repeated, static resource allocation is strategyproof

Part II: (Chapters 5-7) An infrastructure for active enforcement of strategyproofness in decentralized auctions for dynamic resource allocation in open environments

Preliminaries introduce concepts that are used throughout my dissertation, including

formal definitions for a mechanism and strategyproofness, can be found in Chapter 2. Related work in market-based control, dynamic mechanism design, distributed and open mechanism design is discussed in Chapter 3.

Chapter 4 describes a light-weight passive verifier for strategyproofness and extensions that improve the scalability and speed of verification. Chapter 5 describes the decentralized auction framework in detail and defines the constraints imposed on different components in order to ensure strategyproofness. A proof that these constraints are sufficient to ensure strategyproof is also presented.

Chapters 6 and 7 describe how the resource estimation and pricing components of a resource allocation mechanism can be implemented, respectively, while maintaining the constraints that ensure strategyproofness for users in the overall infrastructure. The objective of Chapters 6 and 7 is not to provide the most accurate resource estimation technique or optimal pricing strategy, but to present and compare various techniques that satisfy monotonicity constraints and thus fit within the open, strategyproof infrastructure. These techniques are presented to provide a basis upon which more sophisticated algorithms with better accuracy, or revenue or efficiency properties, can be developed by resources that are willing to compete through these algorithms and other innovations.

1.1.1 EGG project

The main motivating domain for Part II is the *EGG* project, which is an extensible and economics-inspired open grid computing platform. *EGG* is a collaborative effort by physicists, computer scientists, and economists at Harvard University and Boston

University, with the mantra "physicists just want to run their jobs." Users should be able to focus on computational experiments, not gaming to obtain sufficient resources. The work presented in Chapters 5-7 describes the *microeconomic* component of *EGG*.

An initial version of *EGG* is in the development phase, and some components are already in use in managing the T2 ATLAS grid housed at Boston University.

1.1.2 The Crimson Grid

The data set used in my experiments in Chapters 6 and 7 were collected from the Crimson Grid. The Crimson Grid is a campus grid housed at Harvard University's School of Engineering and Applied Sciences. The Crimson Grid was established in 2004 with funding from Harvard University and IBM, for research, data sharing and collaboration among faculty and students across many disciplines. In 2006, the Crimson Grid was supporting the research projects in 21 different scientific disciplines, and grid usage reached 10,000 jobs per month. Crimson Grid researchers have also collaborated with other campuses, including GLOW at the University of Wisconsin. Researchers have used the Crimson Grid to investigate earthquakes and voting patterns, model ocean dynamics and acoustics, and simulate cancerous tumors and subatomic particles [76].

1.2 Contributions

The main focus of my dissertation is to understand how to verify and/or impose (a minimal set of) constraints on mechanisms deployed within an open environment to meet system-wide goals, while allowing for autonomy in decision making. More

specifically, this involves either verifying the strategyproofness of a mechanism using a constraint model with *minimal* overhead, or imposing a small set of constraints ensuring a mechanism is strategyproof. The key components include:

- identifying necessary and/or sufficient conditions for the strategyproofness of resource allocation mechanisms, and
- translating these conditions into constraints on allocation and payment decisions that can be easily verified or imposed.

The contributions of my dissertation are:

1. I design algorithms to passively verify whether a mechanism is strategyproof using a constraint network approach and extensions that accelerate the verification process. I identify conditions that allow a verifier to guarantee that a mechanism is strategyproof for a subset of types and a condition that guarantees that truthful reporting maximizes the worse-case utility of a participant against an adversarial mechanism.

2. I propose an open and extensible framework for active enforcement of strategyproofness in decentralized auctions for dynamic resource allocation. I identify constraints on the resource estimation and pricing components that ensure strategyproofness for users.

3. I design and implement monotonic pricing strategies, and have created a data set based on job histories from the Crimson Grid. I demonstrate the efficacy of the pricing strategies and various resource estimation techniques in conjunction with

constraints guaranteeing strategyproofness via simulations using the Crimson Grid data set.

Chapter 2

Mechanism Design

Mechanism Design is the study of rules for decision making – a mechanism – in multi-agent systems that give agents incentives to do what a designer wants e.g., revealing private information, in order to lead to *good* system-wide decisions. The goal of a mechanism designer is to induce optimal system-wide decision making in domains with self-interested agents with private information. Mechanism Design has received much attention within computer science, with applications in distributed resource allocation and scheduling problems and in electronic commerce [29, 32, 54, 62, 71].

A number of subfields have emerged from classical Mechanism Design, e.g., Online (or Dynamic) Mechanism Design, Distributed Mechanism Design, and Automated Mechanism Design.

I introduce basic definitions in game theory and mechanism design, focusing on *strategyproofness*. I present price-based and monotonicity-based characterizations of strategyproofness, which are used in subsequent chapters to develop algorithms for verification and enforcement.

2.1 Game Theory

2.1.1 Strategies and Utility Functions

Let $\mathcal{O}$ be the set of possible outcomes. Each agent has a valuation function $v \in V$, which determines the agent's preference over outcomes $o \in \mathcal{O}$. An agent's preferences are represented by a *utility function*, which is a mapping from $\mathcal{O} \times V$ to the set of real numbers $\mathcal{R}$. $u_i(o, v_i)$ is the utility of agent i when facing an outcome o, given that her valuation function is v_i.

Definition 1 (strategy)**.** *An agent's* strategy *denoted by* $s_i(v_i) \in \Sigma_i$ *is a complete plan of action defining the agent's behavior in every distinguishable state of the world, where* Σ_i *is the set of all possible strategies, or the* strategy space*.*

A pure strategy defines a deterministic choice of actions at any stage of the game. A mixed strategy is an assignment of a probability to each pure strategy. Let s_i denote the strategy of agent i given her valuation function v_i (conditioning on a valuation function is left implicit). A *strategy profile* $s = (s_1, \ldots, s_n)$ is a set of strategies for each agent fully specifying all actions in a game including exactly one strategy for each agent. Let s_{-i} and v_{-i} denote the set of strategies and valuation functions for all agents except for agent i, respectively.

A *game* defines a strategy space for each agent and a mapping from agent strategies to an outcome. The agent's utility function can now be defined in terms of strategies in a game. Let $u_i(s_1, \ldots, s_n, v_i)$ denote the utility of agent i for the outcome of the game, given valuation function v_i and a strategy profile $s = (s_1, \ldots, s_n)$. The dependence on the valuation functions of all other agents v_{-i} is left implicit. u_i defines preferences

of agent i over her own actions and the actions taken by all other agents, given her valuation function.

Definition 2 (best-response set)**.** *An agent's best response set is a set of one or more strategies that maximize her utility in a game given the agent's valuation function and the strategies of all other agents, i.e.,*

$$br_i(s_{-i}, v_i) = \arg\max_{s_i \in \Sigma_i} u_i(s_i, s_{-i}, v_i)$$

Example 1. *Consider a one-shot closed auction for a single-item with n bidders where the item is sold to the higher bidder, at a price equal to the second highest bid (*second-price auction*).*

An outcome defines an allocation x of the item upon completion of the auction, and the payment p_i of each agent. $x_i = 1$ if and only if bidder i wins the item.

The strategy of each bidder is how much to bid for the item, b_i. Let $v_i(x)$ denote the value that bidder i assigns to an outcome x given a valuation function v_i, e.g., $v_i(x_i = 1)$ is the value that bidder i has for winning the item.

The utility function of each bidder can be written as

$$u_i(b_1, b_2, v_i) = v_i(x) - p_i \tag{2.1}$$

where

$$p_i = \begin{cases} \max_{j \neq i} b_j, & \text{if } x_i = 1; \\ 0, & \text{otherwise.} \end{cases}$$

Utility functions of this form are called quasi-linear *utility functions.*

Assume that $v_i(x_i = 0) = 0$, i.e., the value for not winning the item is 0. A

bidder's best response set given b_{-i}, the bids of all other bidders, is

$$br_i(b_{-i}, v_i) = \begin{cases} \{b_i | v_i(x_i = 1) \geq b_i > hb_{-i}\}, & \text{if } v_i(x_i = 1) > hb_{-i}; \\ \{b_i | b_i \leq v_i(x_i = 1)\}, & \text{otherwise.} \end{cases}$$

where $hb_{-i} = \max_{j \neq i} b_j$ denotes the highest bid from a bidder other than i.

Note that $b_i = v_i(x_i = 1) \in br_i(b_{-i})$ for all b_{-i}.

2.1.2 Solution concepts

Given assumptions about rationality, preferences, and information available to the agents, one can compute the outcome of a game using an equilibrium solution concept. Well-known solution concepts include Nash equilibrium and dominant strategy equilibrium.

Definition 3 (Nash Equilibrium)**.** *A strategy profile $(s_1, \ldots, s_n)$ constitutes a Nash equilibrium given valuation profile $v = (v_1, \ldots, v_n)$ if for each agent, s_i maximizes the utility of agent i given s_{-i}, i.e., s_i is in the best-response set of agent i given s_{-i}*

$$u_i(s_i, s_{-i}, v_i) \geq u_i(s'_i, s_{-i}, v_i) \; \forall s'_i \in \Sigma_i, s'_i \neq s_i$$

The concept of Nash equilibrium is fundamental to game theory, as there exists a Nash equilibrium (potentially in mixed strategies) in every game [52]. However, it makes a perfect information assumption, i.e., the preferences and rationality of every agent are common knowledge. It is also a weak solution concept because there can be multiple Nash equilibria in a game.

The second solution concept I introduce is the dominant strategy equilibrium. It is a stronger solution concept than Nash equilibrium in that the perfect information

assumption is no longer needed and that dominant strategy equilibrium implies Nash equilibrium. A strategy s_i is a *dominant strategy* if s_i maximizes agent i's utility regardless of what strategies are played by other agents.

Definition 4 (Dominant Strategy Equilibrium)**.** *A strategy profile* $(s_1, \ldots, s_n)$ *constitutes a dominant strategy equilibrium if for each agent* i*,*

$$u_i(s_i, s_{-i}, v_i) \geq u_i(s'_i, s_{-i}, v_i) \ \forall s'_i \in \Sigma_i, s'_i \neq s_i, \ \forall s_{-i}$$

for all v_i *and all* v_{-i}*. In other words,* s_i *is in the best-response set of* every s_{-i}*, for all valuation profiles.*

Since agents do not need to know the preferences or rationality (or lack thereof) of other agents to determine which strategy to play, a dominant strategy equilibrium is very robust, and hence, is the solution concept preferred by mechanism designers. For computational agents, the existence of a dominant strategy can help the agent reduce the informational and computational costs of modeling and reasoning.

2.2 Mechanism Design

Formally, a mechanism consists of a social choice function $f : V_1 \times \ldots \times V_n \rightarrow A$, that chooses an alternative $f(v) = a \in A$ from a space of alternatives A, and a payment function $\tilde{p} : V_1 \times \ldots \times V_n \rightarrow \mathbb{R}^n$ that defines a payment $\tilde{p}_i(v)$ by each agent. Here, $v_i \in V_i$ is the valuation of agent i, where $v_i(a) \in \mathbb{R}$ is the value of agent i for alternative a. Also, denote $v = (v_1, \ldots, v_n) \in V$ and $v_{-i} = (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$. The valuation of an agent is private information, although we assume the valuation space V_i

of each agent is common knowledge. Define $E_i(a) = \{d \in A | v_i(a) = v_i(d), \forall v_i \in V_i\}$[1], and let $R_f(v_{-i})$ denote the range of possible alternatives given social choice function f and reports v_{-i} from all but one agent.

The mechanism described above is a *direct-revelation* mechanism, where the only actions available to agents are to make claims about their preferences to the mechanism, i.e., reporting their valuation functions. By the *revelation principle* [26], any mechanism can be transformed into an equivalent incentive-compatible direct-revelation mechanism, which implements the same social choice function.

Let v_i' denote the reported valuation of agent i. The utility function of an agent defined in the previous section can now be written in terms of the *reported* valuations instead of strategy profiles: $u_i(v_i', v_{-i}, v_i)$, or in terms of the outcome implemented by the social choice function f given reported valuations: $u_i(f(v_i', v_{-i}), v_i)$.

A *strategyproof* or *truthful* mechanism is a mechanism for which reporting true valuations constitutes a dominant strategy equilibrium.

Definition 5. *A mechanism M is strategyproof (truthful) if for all agents i with valuation function v_i and for every $v_{-i} \in V_{-i}$:*

$$u_i(f(v_i, v_{-i}), v_i) \geq u_i(f(v_i', v_{-i}), v_i) \ \forall v_i' \in V_i, v_i' \neq v_i, \ \forall v_{-i}$$

For quasi-linear utility functions, this can be written as: $v_i(a) - \tilde{p}_i(v_i, v_{-i}) \geq v_i(b) - \tilde{p}_i(v_i', v_{-i})$ where $a = f(v_i, v_{-i})$ and $b = f(v_i', v_{-i})$, for all types $v_i' \in V_i$, i.e., no agent can do better by misreporting her valuation function, no matter what the other agents report.

[1]For instance, in a resource allocation setting with no allocative-externalities the set of equivalent alternatives $E_i(a)$ would be all allocations that give agent i the same bundle of goods.

Well-known examples of strategyproof mechanisms include the family of Vickrey-Clarke-Groves(VCG) mechanisms [34].

Let $V(n) = \max_{a \in A} \sum_i v_i(a)$ and $V(N_{-i}) = \max_{a \in A} \sum_{j \neq i} v_j(a)$. In a VCG mechanism, each agent i has to pay the amount of loss in total surplus of all other agents caused by agent i.

Definition 6. *The Vickrey-Clarke-Groves (VCG) mechanism defines the social choice function* $f_i(v) = \arg\max_{a \in A} \sum_i v_i(a)$, *and payment rule* $\tilde{p}_i(v) = v_i(f(v_i, v_{-i})) - \big[V(N) - V(N_{-i})\big]$.

The second-price single-item auction is an example of a VCG mechanism. In a second price auction, the price that the winner has to pay is independent of the winner's bid. Other examples of strategyproof mechanisms include: the greedy mechanism for single minded agents due to Lehmann et al. [44], mechanisms for one-parameter agents [5], and truthful and competitive auctions for digital goods [23].

First-price sealed-bid auctions are not strategyproof, as the agent with the highest valuation can improve his utility by bidding ϵ higher than the second highest agent, which is typically lower than her true valuation.

2.2.1 Necessary and Sufficient Conditions for Strategyproofness

In this section I present necessary and sufficient conditions for strategyproofness that are used to derive verification algorithms in Chapter 4.

Necessary: Weak Monotonicity

Extending the work of Roberts [64], Lavi et al. [42] and Bikhchandani et al. [11] define weak monotonicity (*W-MON*) and use it to characterize the set of social choice functions that can be implemented as strategyproof mechanisms.

Definition 7. *A social choice function f satisfies* W-MON *if* $\forall\ v \in V, i,$ *and* $v_i \in V_i$*:* $f(v) = a$ *and* $f(v'_i, v_{-i}) = b \Rightarrow v'_i(b) - v_i(b) \geq v'_i(a) - v_i(a)$

It is easy to show that *W-MON* is a necessary condition for strategyproofness.

Lemma 1. *If* $(f, \tilde{p})$ *is a strategyproof mechanism, then* f *satisfies* W-MON*.*

Proof. Suppose $v_i, v'_i \in V_i$, and let $a = f(v_i, v_{-i})$ and $b = f(v'_i, v_{-i})$. Since $(f, \tilde{p})$ is strategyproof,

$$v_i(a) - \tilde{p}_i(v_i, v_{-i}) \geq v_i(b) - \tilde{p}_i(v'_i, v_{-i})$$

and

$$v'_i(b) - \tilde{p}_i(v'_i, v_{-i}) \geq v'_i(a) - \tilde{p}_i(v_i, v_{-i}).$$

Combining the two inequalities,

$$\begin{aligned} v'_i(b) - v'_i(a) &\geq \tilde{p}_i(v'_i, v_{-i}) - \tilde{p}_i(v_i, v_{-i}) \\ &\geq v_i(b) - v_i(a), \end{aligned}$$

i.e.,

$$v'_i(b) - v_i(b) \geq v'_i(a) - v_i(a).$$

□

Saks and Yu [65] showed that *W-MON* is also sufficient for convex domains.

Necessary and Sufficient: Price-based characterization

The key observation that I use for verifying the truthfulness (or strategyproofness) of mechanisms is that they must be *price-based* (see, for instance, the work of Bartal et al. [9] or Yokoo [74]). Note that the price-based characterization is a condition on both allocation and payments, whereas *W-MON* is a condition on allocation only.

Theorem 1. *A mechanism* $M =< f, \tilde{p} >$ *is strategyproof if and only if for every agent* i *and every* v_{-i}*:*

(A1) the mechanism charges an agent-independent price

$p_i(a, v_{-i})$ *whenever alternative* a *is selected*

(A2) for any report v_i*, any* v_{-i}*, the mechanism chooses an alternative* $a \in R_f(v_{-i})$ *that maximizes* $v_i(a) - p_i(a, v_{-i})$*.*

Here is some intuition for the sufficiency of (A1) and (A2) for strategyproofness: an agent cannot change the price that it faces (A1), and the mechanism maximizes its utility with respect to its reported valuation given these prices (A2). A simple argument can also be constructed for the necessary direction.

If an agent can affect the price she pays given a fixed alternative, she may have incentives to report untruthfully to reduce the price. Similarly, if (A2) is not satisfied, an agent may want to report untruthfully to have the mechanism choose a more desirable alternative.

Hence, if the mechanism is strategyproof, $p_i(a, v_{-i})$ must be constant across different runs of the mechanism for each (a, v_{-i}) pair. Although this characterization is

equivalent to strategyproofness, (A1) and (A2) are not directly verifiable. I can use this price-based characterization to derive constraints on the price space.

Chapter 3

Related Work

In this section, I discuss work related to open mechanism design and economics-inspired methods for resource allocation. Work that specifically relates to verification is presented in Chapter 4 (Section 4.1).

3.1 Market-based Control: Applying Market-based Approaches to Resource Allocation and Scheduling

Market-based control is a paradigm for controlling complex systems using certain features found in markets [17].

A number of existing systems employ market-based approaches to allocate computational resources [2, 7, 15, 16, 54, 71]. These systems assume that resource prices are determined through bidding in a single mechanism or in multiple *identical* mecha-

nisms. For instance, Popcorn [54] and Spawn [71] use auctions for resource allocation. For instance, in Spawn [71], the resource owners each run a sealed-bid second price auction. Popcorn [54] requires buyer agents to split its work into "computelets" and considers the second-price auction and continuous double auction. Gomoluch and Schroeder [29] compare auctions with conventional round-robin approaches.

Another class of market-based control leverages equilibrium theory. The *Market-oriented programming* paradigm [73], proposed by Wellman, leverages the general equilibrium theory from microeconomics [47]. Prices are determined through a tâtonnement [72] process. Kuwabara et al. [39] propose a price-equilibrium approach, where the resource prices are computed by the sellers based on demand.

However, none of these systems consider dynamic manipulations, none provide the bid expressiveness that is supported in the framework of Chapter 5, and none embrace the need for open, flexible and decentralized control in computational grids and cloud computing systems.

Most applications of market-based control have focused on allocating computational resources, but the paradigm has been applied to other complex decision systems, e.g., for factory scheduling [8], distributed climate control [18], and distributing air pollution rights [46].

3.2 Online/Dynamic Mechanism Design

Online or dynamic mechanism design is an extension of mechanism design for settings in which agents arrive and depart over time, and decisions must be made dynamically without full knowledge of future arrivals or departures. This is an at-

tractive model for open environments because it is unlikely that the set of participants in a mechanism will be fixed for a long period.

Several recent papers have focused on dynamic auctions [57]. Some of these papers [12, 37] assume that agents arrive in a predetermined order, and an agent's only private information is her value. In these single-parameter domains, designing truthful auctions is easier. Others [6, 32, 43, 62] present online auctions that are strategyproof against misreports of arrival or departure time using monotonically nondecreasing prices over time, which is also the case for the auction framework presented in Chapter 5.

Porter [62] presents a truthful online mechanism for jobs of different lengths, where an agent must be granted the resource for a total duration equal to its job length. Hajiaghayi et al. [32] consider a setting where users bid for access to a reusable resource such as processor time or wireless network access, and present a characterization for the class of truthful online allocation rules.

The model from Chapter 5 can be seen as an extension of the work of Hajiaghayi et al. [32] that allows for general value schedules and additional parameters (e.g., job description and reliability metric) and supports strategyproofness via a minimal set of rules, i.e., is appropriate for an open environment.

3.3 Distributed Mechanism Design/ Open Mechanism Design

Work in distributed mechanism design and open mechanism design share my motivation for a open and dynamic environment. Distributed Algorithmic Mechanism Design (DAMD) [22] considers distributed resource allocation problems where a mechanism involves distributed computation carried out over a network, focusing on the *network complexity*, i.e., the computational and communication efficiency, of the distributed algorithm.

Shneidman's work on *faithfulness* [66, 67] presents a framework for proving specification faithfulness, whether a rational node implementing a piece of a distributed mechanism would want to deviate from a specification. Faithfulness focuses on proving properties of a single specification of a mechanism programmed into distributed pieces, while my framework guarantees properties when multiple specifications are present.

Mu'alem's work on testing truthfulness [49] shares my motivation for designing algorithms for verifying properties of mechanisms. She discusses a technique for constructing a generic (almost) truthful mechanism. She presents an algorithm for testing truthfulness for a restrictive case of zero-one valuations where an agent has a valuation of 1 for a specific allocation and zero value for all other allocations, drawing upon the algorithm for testing Boolean monotonicity by Goldreich et al. [28].

Chapter 4

Passive Verification of the Strategyproofness of Mechanisms in Open Environments

Abstract

Consider an open infrastructure in which anyone can deploy mechanisms to support automated decision making and coordination amongst self-interested computational agents. Strategyproofness is a central property in the design of such mechanisms, allowing participants to maximize their individual benefit by reporting truthful private information about preferences and capabilities and without modeling or reasoning about the behavior of other agents. But, why should participants trust that a mechanism is strategyproof? I address this problem, proposing and describing a *passive verifier*, able to monitor the inputs and outputs of mechanisms and verify

the strategyproofness, or not, of a mechanism. Useful guarantees are available to participants before the behavior of the mechanism is completely known, and metrics are introduced to provide a measure of partial verification. Experimental results demonstrate the effectiveness of my method.

A new issue arises for mechanism design in open settings. Why should participants *trust* that a mechanism has the strategyproof property that it claims? I address this problem, proposing and describing a *passive verifier*, able to monitor the inputs and outputs of mechanisms and verify the strategyproofness, or not, of a mechanism. Thus, my focus in this chapter is not on the design of new mechanisms but on the verification of strategyproofness. Indeed, without verification careful design is worthless: it is only when a mechanism is both strategyproof *and known to be strategyproof* that participants gain the benefits of simple strategies and mechanisms behave as designed. Thus, verification is in the interest of both designers and participants.

My passive verifier exploits a price-based characterization of truthful mechanisms and is inspired by the work of Gui et al. [31] on graph-theoretic characterizations of truthful mechanisms. I formulate the verification problem as one of checking for feasible solutions to a constraint satisfaction problem defined on a graph representing the rules of a mechanism. Verifiers are situated in the computational infrastructure and act as intermediaries between participants and a mechanism. A verifier has the power to *veto* the outcome of a mechanism when the mechanism is proved to violate strategyproofness.

I identify techniques to both accelerate the process of verification as well as reduce the space complexity of verification. The idea is to place (weak) restrictions on the space of allowable mechanisms in order to simplify the task of verification. I provide three illustrations of this idea. First, I introduce the notion of *summarization*, which requires that a mechanism place restrictions on the complexity of its pricing rule, in identifying some subset $w < n$ of n participants, that define the price faced by each

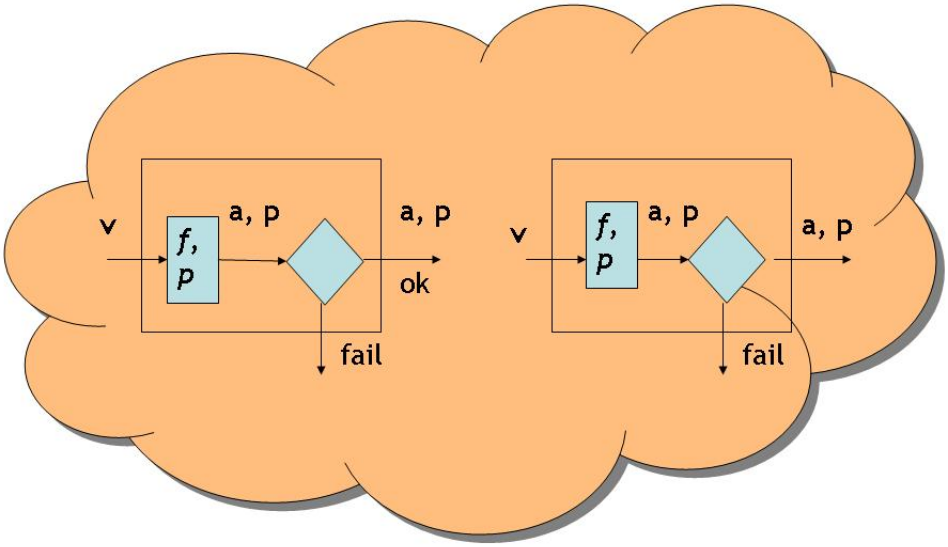


Figure 4.1: A light-weight verifier observes the inputs and outputs of mechanisms, and intervenes if violations to strategyproofness are found (v = valuation profile, (f, p) = (social choice function, payment rule), (a, p) = (allocation, payments)).

agent. Summarization provides an exponential reduction in the space complexity of verification, from $O(md^{n-1})$ to $O(md^w)$, in domains with m alternatives and with d possible agent valuations. Second, when a mechanism is required to satisfy a standard property on payments, that I term *natural payments*, I can leverage constraint propagation to further accelerate verification. Third, when a mechanism is required to satisfy a property of *envy-freeness*, reasonable for simple domains, additional acceleration is achieved.

The biggest challenge is to provide useful feedback to participants before the complete input space for a mechanism has been observed. Here, I identify a property that allows the verifier to guarantee that an agent's best strategy is truthful reporting (conditioned on the mechanism passing the verifier) even though only a subset of possible inputs has been observed. I identify a weaker condition that ensures that truthful reporting maximizes the *worst-case* utility of a participant against an adversarial (but ultimately non-strategyproof) mechanism in early stages of verification. I also provide two metrics, namely *probability-of-strategyproofness* and *price-flexibility*, to measure the degree of strategyproofness that is ensured for a mechanism that is still not completely verified. These metrics facilitate an empirical study in which I demonstrate the speed of verification (or failure of verification when using a weaker, baseline method) on various mechanisms, both strategyproof and non-strategyproof.

4.1 Related Work: Verification

Plenty of prior work has leveraged the methods of cryptography and secure function evaluation [56, 51] to achieve absolute trust (i.e. the rules of a mechanism are

published and correctness is verifiable to any party). When designing an auction that will be used for a multi-million dollar allocation of government bonds the overhead of cryptographically-secured proofs is well justified. My motivation comes from a vision of dynamic, open environments that can support a multitude of frequent, perhaps even mundane, decisions. For instance, I wish to enable infrastructure that can support the minute-by-minute allocation of computational services, the buying and selling of limited-play songs, and services to make dinner reservations, solicit information about flight prices, and find answers to trivia questions.

Similarly, there is a large literature on the use of logic formalisms to verify the correctness of mechanisms and other institutions [25, 59, 60]. Here, a logic specification makes explicit the rules of a mechanism and properties such as strategyproofness can (in principle) be established through methods such as model checking. Of particular relevance is the work of Guerin and Pitt [30], who share the same motivation of allowing for the deployment of trusted mechanisms in open environments. Moreover, these authors discuss the use of "sentinel agents" which can be used to verify the properties of mechanisms at run time. Here, I am *only* interested in the verification of strategyproofness and not in the verification of other properties. Second, I deliberately eschew logic-based formalisms because model checking is intractable for appropriate logics [69], and because logic formalisms are a bad fit for quantitative properties such as strategyproofness and for the methods of combinatorial optimization that are important for internal decision making by mechanisms.

4.2 Problem Definition

The verifiers act as lightweight trusted intermediaries (or "wrappers") that are situated as a default interface between published mechanisms and participants. A verifier receives bids from agents, passes them to the mechanism and then receives the outcome and payments from the mechanism. At this point the verifier checks that the mechanism has not violated strategyproofness, and if this is OK it will pass the outcome on to the agents. Otherwise the verifier can exercise *veto* power on the outcome.

A mechanism is treated as a black box, i.e., the verifier can only observe a sequence of inputs, $v^1, v^2, \ldots v^k, \ldots$, where v^k denotes the reported valuations of the agents in the kth run of the mechanism, and a sequence of outputs, $< a^1, p^1 >$, $< a^2, p^2 >$, $\ldots$, $< a^k, p^k >$, $\ldots$ where $< a^k, p^k >$ denotes the alternative and vector of payments produced by the mechanism in the kth run.

Definition 8 (passive verification problem)**.** *An online decision problem in which a verifier observes a sequence of inputs and outputs to a mechanism (assumed fixed and deterministic) and determines whether the mechanism is strategyproof or not strategyproof.*

A useful verifier will also reject a non-strategyproof mechanism quickly and provide guidance on whether or not a mechanism is likely to be strategyproof even before a final "reject" or "accept" is generated. I also demonstrate that a verifier can prove that a mechanism is strategyproof given the private valuation of a bidder, even before the mechanism has been fully verified.

Here are the main assumptions:

1. *The space of alternatives and the type space are* finite.

2. *The verifier is trusted, is able to observe all inputs and outputs to a mechanism, and has* veto *power on any decision made by the mechanism (e.g. as soon as some decision, perhaps this decision, provides proof that the mechanism is not strategyproof).*

3. *Once the mechanism has decided on an alternative and payments for an instance, it can no longer change that decision in the future.*[1]

4. *The mechanism is* anonymous*, meaning that $f(v_i, v_{-i}) \in E_i(a)$ for all permutations of v_{-i}.*

The first assumption is needed to ensure that a sound and complete verifier can operate with bounded memory requirements. The second assumption is essential to my approach to passive verification. The third assumption ensures that the entire history of instances is always relevant for validating or invalidating strategyproofness. The final assumption allows the valuation profile, v_{-i}, of all agents except i to be treated as an unordered set of $n-1$ elements (allowing for repeated elements). This is helpful computationally, but should be relaxed in settings where anonymity is not provided (e.g. *optimal auctions* [50] where prior information about bidder valuations is used to bias the outcome of a mechanism.) These anonymous semantics for v_{-i} will be assumed for the rest of the paper.

[1]For ties, this requires that ties are broken the same way each time. Notice that this does not rule out some forms of adaptiveness: a mechanism can update its prices via learning on any (a, v_{-i}) that has not been observed.

4.3 Simple Checker

In this section I define a set of rules that can be implemented by a passive verifier, and prove that verification with these rules is sound and complete. A constraint network formulation is given for the rules, which leads to a concrete algorithmic instantiation for a verifier.

4.3.1 Rules for Verification

Recall from Chapter 2 that strategyproofness is equivalent to the price-based characterization.

Theorem 2. *A mechanism $M =< f, \tilde{p} >$ is strategyproof if and only if for every agent i and every v_{-i}:*

(A1) the mechanism charges an agent-independent price

$p_i(a, v_{-i})$ *whenever alternative a is selected*

(A2) for any report v_i, any v_{-i}, the mechanism chooses an alternative $a \in R_f(v_{-i})$ that maximizes $v_i(a) - p_i(a, v_{-i})$.

Fix v_{-i}. By condition (A2) from Theorem 2, a strategyproof mechanism must choose an alternative $a \in A$ such that $v_i(a) - p_i(a, v_{-i})$ is maximized. Hence, if $f(v_i, v_{-i}) = a$, then $v_i(a) - p_i(a, v_{-i}) \geq v_i(b) - p_i(b, v_{-i})$ for all $b \neq a \in A$.

Let $G_a \subseteq \{v_i \in V_i | a \in E_i(f(v_i, v_{-i}))\}$. Rearranging, I have, $p_i(a, v_{-i}) - p_i(b, v_{-i}) \leq v_i(a) - v_i(b)$ $\forall b \neq a$ for all $v_i \in G_a$. Hence, I get the following inequality:

$$p_i(a, v_{-i}) - p_i(b, v_{-i}) \leq \inf_{v_i \in G_a} \{v_i(a) - v_i(b)\} \tag{4.1}$$

Similarly, considering cases where $f(v_i, v_{-i}) = b$, I get:

$$p_i(b, v_{-i}) - p_i(a, v_{-i}) \leq \inf_{v_i \in G_b} \{v_i(b) - v_i(a)\} \tag{4.2}$$

Combining the two, for each pair $< a, b >$, I have:

$$\sup_{v_i \in G_b} \{v_i(a) - v_i(b)\} \leq p_i(a, v_{-i}) - p_i(b, v_{-i}) \tag{4.3}$$

$$\leq \inf_{v_i \in G_a} \{v_i(a) - v_i(b)\} \tag{4.4}$$

This is well known, see for instance Gui et al. [31] and Lavi et al. [42], and suggests the following simple verification procedure.

Let H denote the *history* of instances currently available to the verifier. Define the following: $v \in v^H$ denotes bids $v \in V^N$ received so far; $v_i \in v_i^H(v_{-i})$ denotes the bids from agent i that have been observed for v_{-i} from the other agents; $v_i \in v_i^H(a, v_{-i})$ denote the bids received from agent i with the additional restriction that alternative a (or an equivalent) was selected; $a = f^H(v)$ denotes the alternative selected given input v (it is sufficient to choose any one of a set of equivalent alternatives); $f^H(v_{-i}) \subseteq A$ denotes the set of alternatives selected for v_{-i}, and $p_i^H(a, v_{-i})$ denotes the payment made by agent i given alternative a and v_{-i}. For every new instance I first check whether the exact same value profile has been seen before. If this is the case then the same alternative must be selected. By case (b), if a^k (or equivalent) has been seen before for v_{-i}^k then by (A1) the payment by i must be the same. Case (c) ensures that *previous agents could not have done better* given this new information: if a^k is a new alternative for v_{-i}^k then the price on this alternative must be high enough for (A2), and thus for inequality (4.3) to hold for bids that have already been observed. Case (d) ensures that the *current agent could not have done better given the current*

SIMPLECHECKER.

Initialize history H to an empty history.

1. For a new instance $< v^k, a^k, p^k >$, and for each agent i, consider history $f^H(v^k_{-i})$ and if non-empty:

   (a) if $v^k_i \in v^H_i(v^k_{-i})$ check $a^k \in E_i(f^H(v^k))$.

   (b) if $a^k \in f^H(v^k_{-i})$ check $p^k_i = p^H_i(a^k, v^k_{-i})$.

   (c) if $a^k \notin f^H(v^k_{-i})$ check the price p^k_i is no less than:

   $$\sup_{b \in f^H(v^k_{-i})} \left\{ p^H_i(b, v^k_{-i}) + \sup_{v_i \in v^H_i(b, v^k_{-i})} (v_i(a^k) - v_i(b)) \right\} \tag{4.5}$$

   (d) if $v^k_i \notin v^H_i(v^k_{-i})$ check the price p^k_i is no greater than:

   $$v^k_i(a^k) + \inf_{b \in f^H(v^k_{-i})} \left\{ p^H_i(b, v^k_{-i}) - v^k_i(b) \right\} \tag{4.6}$$

2. If any check fails, then **reject** the mechanism, else update history for v^k_{-i} as necessary (i.e. whenever a new v_i and/or new alternative was observed).

3. Once all $v_i \in V_i$ for all $v_{-i} \in V_{-i}$ have been observed, then **pass** the mechanism.

---

history: if v_i^k is a new bid for v_{-i}^k then the price on this alternative must be low enough for (A2), and thus for inequality (4.4) to hold for other alternatives that have been observed.

Example 2. *Consider an auction mechanism for an allocation problem with multiple identical items. Suppose that the sequence of instances in Table 4.1 are observed (all with two bidders and two items). Instance 1 indicates that bidder 1 has value 4 for 1 unit and 8 for 2 units. Bidder 2 has value 4 for 1 unit and 9 for 2 units. The allocation gives both units to bidder 2, and bidder 2 makes payment 8. The auction implements the VCG mechanism for instances 1–3 but deviates in instance 4 (The VCG mechanism would choose the same allocation but with payments (0, 9)).*

Consider the build-up of history for $v_{-i} = (4, 9)$*, and consider allocations in which bidder i receives 0, 1 or 2 items. Let* $p_i(n)$ *denote the price that agent i faces when it wins n items. After instance 1, I get* $p_i(0) = 0$*. After the second instance, the verifier learns that* $p_i(1) = 5$*, and checks that the constraints (c)* $p_i(1) \geq p_i(0) - (v_i^1(0) - v_i^1(1)) = 4$ *and (d)* $p_i(1) \leq p_i(0) + v_i^2(1) - v_i^2(0) = 5$ *are satisfied. After instance 3, agent i wins one item and pays the price of 5, so the verifier checks that (b)* $p_i^3(1) = p_i(1) = 5$ *holds. After instance 4, the verifier learns* $p_i(2) = 10$*, and checks that (c)* $p_i(2) \geq \max\{p_i(0) - (v_i^1(0) - v_i^1(2)), p_i(1) - (v_i^2(1) - v_i^2(2))\} = 9$*. However, the verifier also checks for the constraint (d)* $p_i(2) \leq p_i(1) + v_i^4(2) - v_i^4(1) = 9$*, which is violated (the price is too high). Hence, this mechanism is rejected by* SimpleChecker *after the fourth instance.*

Alternatively, suppose that the fourth instance is changed to $v^4 = ((7, 9), (4, 9)), a^4 = (2, 0), p^4 = (8, 0)$*, where the mechanism still deviates from the*

VCG outcome ($a^4_{VCG} = (1,1)$ and $p^4_{VCG} = (5,2)$). This time, after instance 4, the verifier learns $p_i(2) = 8$, and checks that (c) $p_i(2) \geq \max\{p_i(0) - (v_i^1(0) - v_i^1(2)), p_i(1) - (v_i^2(1) - v_i^2(2))\} = 9$. Now this constraint is violated (the price too low), and the mechanism is rejected after the fourth instance.

instance	values	allocation	payments
1	$((4,8),(4,9))$	$(0,2)$	$(0,8)$
2	((4, 9), (5, 9))	(1, 1)	(4, 5)
3	((5, 8), (4, 9)	(1, 1)	(5, 4)
4	((4, 9), (5.5, 10))	(0, 2)	(0, 10)

Table 4.1: Sequence of Instances: 2 Agents and 2 Identical Items

4.3.2 Establishing Soundness and Correctness

The first task is to establish soundness and correctness of the verifier. I hold these properties to be necessary for an (exact) verifier, although insufficient to show the utility of passive verification. Soundness simply proves that SimpleChecker will detect the failure of strategyproofness eventually, once all inputs are observed. The main challenge is to provide intermediate feedback, discussion of which is delayed until Section 4.6.

Theorem 3 (Soundness)**.** *The rules as defined by* SimpleChecker *will detect a non-strategyproof mechanism even if all agents are not truthful as long as all inputs are eventually observed.*

Proof. A mechanism is not strategyproof if either:

¬ (A1) there exists some i, v_{-i} and $v_i, v_i' \in V_i$ such that $f(v_i,\ v_{-i}) = f(v_i', v_{-i})$, but $\tilde{p}_i(v_i, v_{-i}) \neq \tilde{p}_i(v_i', v_{-i})$, or

¬ (A2) There exists some i, v_{-i} and $v_i, v_i' \in V_i$, $v_i \neq v_i'$ such that $v_i(f(v_i, v_{-i})) - p_i(f(v_i, v_{-i}), v_{-i}) < v_i(f(v_i', v_{-i})) - p_i(f(v_i', v_{-i}), v_{-i})$.

Suppose toward contradiction that a non-strategyproof mechanism passes the SimpleChecker after all possible reports $v \in V$ are observed. First, suppose ¬ (A1). This is not possible because check (b) of SimpleChecker would catch the price deviation. Second, suppose ¬ (A2), such that there exists some i, v_{-i} and $v_i, v_i' \in V_i$ such that

$$v_i(f(v_i, v_{-i})) - p_i(f(v_i, v_{-i}), v_{-i}) < \\ v_i(f(v_i', v_{-i})) - p_i(f(v_i', v_{-i}), v_{-i}) \tag{4.7}$$

Let $v_i = v_i^k$ and $v_i' = v_i^l$. If $k > l$, then when instance k is observed, $a^l \in f^H(v_{-i})$. By (4.7),

$$p_i^k(a^k, v_{-i}) > p_i^l(a^l, v_{-i}) + v_i^k(a^k) - v_i^k(a^l) \\ \geq \inf_{b \in f^H(v_{-i}^k)} [p_i^H(b, v_{-i}^k) + (v_i^k(a^k) - v_i^k(b))],$$

and step (d) of SimpleChecker would fail. A contradiction. If $l > k$, then when instance l is observed, $a^k \in f^H(v_{-i})$. By (4.7), I have $p_i^l(a^l, v_{-i}) <$

$$p_i^k(a^k, v_{-i}) + v_i^k(a^k) - v_i^k(a^l) \\ \leq p_i^k(a^k, v_{-i}) + \sup_{v_i \in v_i^H(a^k, v_{-i})} (v_i(a^k) - v_i(a^l)) \\ \leq \sup_{b \in f^H(v_{-i})} [p_i^H(b, v_{-i}) + \sup_{v_i \in v_i^H(b, v_{-i})} (v_i(a^l) - v_i(b))],$$

and step (c) would fail. A contradiction. □

Theorem 4 (Correctness)**.** *The rules as defined by* SimpleChecker *will never halt a strategyproof mechanism even for agents that are untruthful.*

Proof. Suppose toward contradiction that a strategyproof mechanism M is rejected by SimpleChecker after k instances. Since M satisfies (A1), it will not fail step (b). So it either fails (c) or fails (d). Suppose it fails (c). Then there exists $a^l \in f^H(v^k_{-i})$, $p^k_i(a^k, v^k_{-i}) < p^l_i(a^l, v^k_{-i}) + (v^l_i(a^k) - v^l_i(a^l))$ for some $l < k$. Then $v^l_i(a^k) - p^k_i(a^k, v^k_{-i}) > v^l_i(a^l) - p^l_i(a^l, v^k_{-i})$, violating (A2) and a contradiction. Now, suppose the mechanism fails (d). Then there exists $a^l \in f^H(v^k_{-i})$, $p^k_i(a^k, v^k_{-i}) > p^l_i(a^l, v^k_{-i}) + (v^k_i(a^k) - v^k_i(a^l))$ for some $l < k$. Then $v^l_i(a^k) - p^k_i(a^k, v^k_{-i}) < v^l_i(a^l) - p^l_i(a^l, v^k_{-i})$, violating (A2) and a contradiction. □

4.4 Network Checker

An algorithm for SimpleChecker can be identified by reformulating the task as that of checking for a feasible solution to a constraint network.

I gain two main advantages from formulating the problem in terms of constraint networks. First, I can leverage standard data structures (e.g. linked-lists) and standard algorithms (e.g. all-pairs shortest path algorithms) to construct a passive verifier. Second, I can gain additional accelerations by introducing constraints *between* networks to capture additional problem structure (see Section 4.5.2).

For each v_{-i}, I construct a constraint network in which each node in the network is associated with an alternative and arcs in the network impose constraints on the difference in prices between pairs of alternatives. Each network contains a single node for each set of equivalent alternatives; e.g. one node for all allocations in which agent

i receives bundle of goods S, irrespective of the allocation to other agents. In addition to binary constraints, each node is also annotated with an exact price, once known.[2]

Recall that every strategyproof mechanism must satisfy inequalities (4.3) and (4.4). In order to capture this requirement for some pair of alternatives $< a, b >$ given current history H, a directed arc $b \rightarrow a$ is labeled with a weight $w(b, a)$ representing the linear inequality $p_i(a, v_{-i}) - p_i(b, v_{-i}) \leq w(b, a)$, with $w(b, a) = \inf_{v_i \in v_i^H(a, v_{-i})} \{v_i(a) - v_i(b)\}$. Similarly, a directed arc $a \rightarrow b$ is created and labeled with weight $w(a, b)$. In addition, each node that represents an observed alternative can be annotated with the price.

As described, my constraint network is of the same form as a Simple Temporal Problem (STPs)(Dechter et al. [21]). Prices take the role of time and the weights on arcs are now bounds on the difference between prices. The existence of prices that form a feasible solution to this network is equivalent to the nonexistence of negative-length cycles in the constraint network.

Theorem 5. *[21] A given STP T is consistent if and only if it has no negative-length cycles.*

Thus, I can check for the existence of prices satisfying all constraints by checking for the existence of negative-length cycles using an all-pairs-shortest-path algorithm. I also refer to this process as *tightening* the network.

Let $\mathcal{N}$ denote the current set of networks (one for each v_{-i} that has been ob-

[2]Compared with the directed graph formalism introduced in Gui et al. [31] for reasoning about strategyproof mechanisms, the constraint network of a passive verifier is typically incomplete (only containing a subset of alternatives) and associates fixed prices with alternatives that have been observed. These prices imply constraints on future prices. In comparison, Gui et al. use the constraint network to reason (in analysis) about whether *any* assignment of prices is possible.

served). New networks are introduced dynamically (for new v_{-i} sets) and a new node is introduced on a network when an additional alternative is observed.

Algorithm NetworkChecker implements the sound and correct rules defined in SimpleChecker. If a set of feasible prices exist, by step (c) and step (e) of NetworkChecker, $p_i^a - p_i^k \leq w(a^k, a) = \inf_{v_i \in v_i^H(a,v_{-i})}(v_i(a) - v_i(a^k))$ or $p_i^k - p_i^a \geq \sup_{v_i \in v_i^H(a,v_{-i})} (v_i(a^k) - v_i(a))$ for all existing nodes a. Rewriting, I get $p_i^k \geq p_i^a + \sup_{v_i \in v_i^H(a,v_{-i})}(v_i(a^k) - v_i(a))\ \forall a \in H(v_{-i}^k)$, which is precisely the condition that is checked in step (c) of the SimpleChecker. Now consider step (d), in combination with step (e), of NetworkChecker: if an arc from node a to node a^k already exists, and the arc has weight $w < v_i^k(a^k) - v_i^k(a)$, then I know that $p_i^{a^k} - p_i^a \leq w$ even before observing instance k, and no constraints are updated. Otherwise, I have a new (or tighter) constraint on the set of feasible prices, namely, $p_i^k - p_i^a \leq w(a, a^k) = v_i^k(a^k) - v_i^k(a)$ for all a such that $w(a, a^k)$ is updated after observing instance k. $p_i^k \leq p_i^a + (v_i^k(a^k) - v_i^k(a))$ for all a such that $w(a, a^k)$ is updated. Combining the two cases, I get $p_i^k \leq v_(^k a^k) + \inf_{a \in f^H(v_{-i}^k)}[p_i^a - v_i^k(a)]$, which is the condition checked in step (d) of the SimpleChecker.

Example 3. *I can revisit the example in Table 4.1. Figure 4.2 illustrates the constraint network for* $v_{-i} = (4, 9)$. *After instance 4, the arc from node* $< 1, 5 >$ *to node* $< 2, 10 >$ *has weight 4, the arc from node* $< 2, 10 >$ *to node* $< 1, 5 >$ *has weight -4, the arc from node* $< 0, 0 >$ *to node* $< 2, 10 >$ *has weight 10, and the arc from node* $< 2, 10 >$ *to node* $< 0, 0 >$ *has weight -8. Given these weights, I see that the prices* $p(0) = 0$, $p(1) = 5$, *and* $p(2) = 10$ *are inconsistent, since* $p(1) - p(2) = -5 < -4$.

NETWORKCHECKER.

Initialize $\mathcal{N} = \emptyset$.

1. For a new instance $< v^k, a^k, p^k >$, and for each agent i, if there is a constraint graph $G \in \mathcal{N}$ for v^k_{-i} then work with this graph. Otherwise, create an empty graph G and add to set $\mathcal{N}$. Then:

   (a) if $v^k_i \in v^H_i(v^k_{-i})$ then check $a^k \in E_i(f^H(v^k))$.

   (b) if node a^k (or equivalent[3]) exists in graph G then check p^k_i equals price on node.

   (c) Otherwise, add node a^k and assign price p^k_i to the node, and for every other node a in the graph:

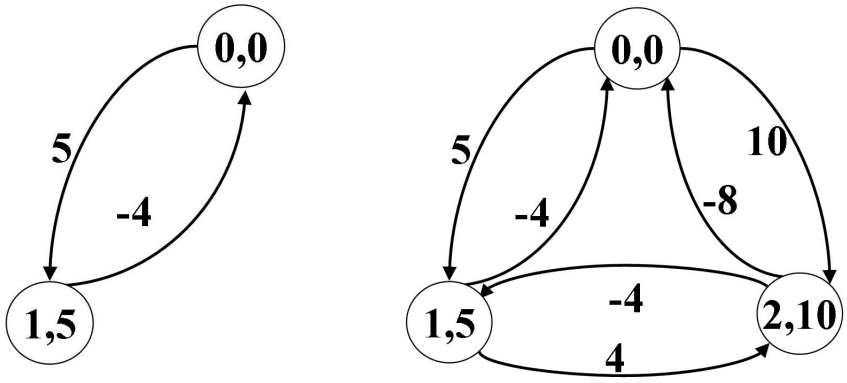
   add a new arc from the new node a^k to a with weight $w = \inf_{v_i \in v^H_i(a, v_{-i})}(v_i(a) - v_i(a^k))$

   (d) add a new arc from every other node a to the new node a^k with weight $v^k_i(a^k) - v^k_i(a)$ if no arc exists. If one already exists, and $v^k_i(a^k) - v^k_i(a)$ is less than the current weight then update the weight to be $v^k_i(a^k) - v^k_i(a)$.

   (e) tighten the network and check for feasibility.

2. If any check fails, then **reject** the mechanism, else update the history as necessary.

3. Once all $v_i \in V_i$ for all $v_{-i} \in V_{-i}$ have been observed then **pass** the mechanism.



(a) After instance 2 (b) After instance 4

Figure 4.2: Constraint network for Multiple Identical Items with Two Bidders and Two items.

4.5 Accelerated Verification via Structural Requirements

The biggest potential shortcoming of this approach to verification is that the space complexity quickly becomes untenable: there are d^{n-1} subnetworks, for a type space of size d and n agents, and thus the space complexity is exponential in the number of agents.

In order to address this problem I propose that the passive verifier impose *structural requirements*. I consider three kinds of structural requirements: *(a) summarization, (b) natural payments, (c) envy-freeness.* These are illustrative of a more general approach which is to restrict the space of implementable mechanisms in order to allow for efficient verification. More than just reducing the space requirements for verification, these structural requirements also *accelerate* verification by allowing for

stronger inference and more rapid proofs of non-strategyproofness.

A strategyproof mechanism may wish to volunteer structural requirements in order to facilitate faster verification. But, given that the concern in verification is to identify the "bad apples" then it is, in general, more appropriate for the verifier to require certain additional properties, especially properties that appear to hold for many plausible mechanisms. I will comment on the restrictiveness of each property as it is introduced.

4.5.1 Summarization

Summarization provides an exponential reduction in memory and computational requirements on the verifier and also significantly accelerates the process of verification.

Definition 9 (valid summarization function)**.** *Given reports v_{-i}, a summarization function s selects a subset $s(v_{-i})$ of reports, and is valid when $p_i(a, v_{-i}) = p'_i(a, s(v_{-i}))$ for all a, all v_{-i}, for some price function p'_i.*

Example 4. *In a single-item Vickrey auction the price that an agent faces is determined by the highest bid among bids from other agents and $s(v_{-i}) = \{\max_{j \neq i}\{v_j\}\}$ is a valid summarization function. In this case, the induced price rule $p'_i(i \text{ wins item}, s(v_{-i})) = x$ given $s(v_{-i}) = x$ and $p'_i(i \text{ does not win item}, s(v_{-i})) = 0$.*

Example 5. *Consider a combinatorial auction with single-minded bidders. A bidder is single-minded if there exists a set g of goods and value b such that $v(g') = b$ if $g \subseteq g'$, and $v(g') = 0$ otherwise. The* LOS mechanism*[44] consists of the greedy allocation*

rule and the greedy payment rule. The greedy allocation rule works as follows: bids are sorted in decreasing order according to the average amount[4] *of a bid* $v_j =< g_j, b_j >$*, defined as* $\frac{b_j}{|g_j|}$*, where* $|g_j|$ *is the number of goods in the desired set* g_j *of agent* j*, and* b_j *is her value for* g_j*. Then given this sorted list* L*, each bid is examined in order and is granted if and only if it does not conflict with any of the bids previously granted. For each* $v_j \in L$*, define the set* $D_j = \{i | i > j, g_i \cap g_j \neq \emptyset, \forall l < i, l \neq j, l'\text{th bid granted} \Rightarrow g_l \cap g_i = \emptyset\}$*. Note that* $i > j \Leftrightarrow \frac{b_i}{|g_i|} \leq \frac{b_j}{|g_j|}$*.* D_j *is the set of indices of the bids that are denied, but would have been granted if it were not for the presence of* v_j*. Under the greedy payment rule, if agent* j*'s bid is granted and* $D_j \neq \emptyset$*,* j *pays* $|g_j|\frac{b_i}{|g_i|}$ *where* $i = \min D_j$*. In other words,* j *pays the average amount of the first bidder whose bid was denied due to* j*, per good won. Otherwise,* j *pays 0. In the LOS mechanism, the summarization function* $s(v_{-i}) = \{v_j | \ \forall k < j, k \neq i, g_k \cap g_j = \emptyset\}$*. If VCG payments are used instead, the set* $s(v_{-i})$ *is given by first sorting the set* v_{-i} *according to the norm used in the LOS allocation rule and then removing each bid whose set of desired items is a superset of the set of desired items of any bid that appears before it in the ordered list.*

Summarization information can be defined by a mechanism incrementally, by identifying a subset of values v_{-i} that define the prices to agent i for each instance. In this case, this is done by extending the interface between the verifier and the mechanism. Summarization functions can also be defined statically by instantiating an explicit function. As long as the summarization function is *constant*, then passive verification remains sound and correct.

[4]The average amount can be replaced with any norm that satisfied bid-monotonicity, i.e., is nondecreasing in b_i and $norm(< g, b >) \geq norm(< g', b >)\ \forall g \subseteq g'$.

Theorem 6. *A passive verifier that implements rules* SimpleChecker *will continue to detect a non-strategyproof mechanism and continue to pass any strategyproof mechanism when used in combination with a fixed summarization function.*

Proof. Correctness is immediate. To show soundness, suppose a mechanism is demonstrated to be non-strategyproof without summarization after k instances. I argue that the mechanism will still be demonstrated to be non-strategyproof with summarization. First, note that the use of summarization function does not affect step (a) of SimpleChecker. I now have the following 3 cases:

1. Check fails in (b) of SimpleChecker: $a^k \in f^H(v^k_{-i})$ and $p^k_i(a^k, v^k_{-i}) \neq p^l_i(a^k, v^k{-}i) = p^H_i(a^k, v^k_{-i})$ for some $l < k$. By definition of the summarization function, $p^k_i(a^k, v^k_{-i}) = p^k_i(a^k, s(v^k_{-i})$ and $p^l_i(a^k, v^k{-}i) = p^l_i(a^k,\ s(v^k{-}i))$. It follows that $p^k_i(a^k, s(v^k_{-i}) \neq p^l_i(a^k, s(v^k_{-i})$.

2. Check fails in (c) of SimpleChecker: there exists $a^l \in f^H(v^k_{-i})$, $p^k_i(a^k, v^k_{-i}) < p^l_i(a^l, v^k_{-i}) + (v^l_i(a^k) - v^l_i(a^l))$ for some $l < k$. Since $p^k_i(a^l, v^k_{-i}) = p^k_i(a^l, s(v^k_{-i}))$ and $p^l_i(a^l, v^k_{-i}) = p^k_i(a^l, s(v^k_{-i}))$, $p^k_i(a^k, s(v^k_{-i})) < p^l_i(a^l, s(v^k_{-i})) + (v^l_i(a^k) - v^l_i(a^l))$.

3. Check fails in (d) of SimpleChecker: there exists $a^l \in f^H(v^k_{-i})$, $p^k_i(a^k, v^k_{-i}) > p^l_i(a^l, v^k_{-i}) + (v^k_i(a^k) - v^k_i(a^l))$ for some $l < k$. Then $p^k_i(a^k, s(v^k_{-i})) > p^l_i(a^l, s(v^k_{-i}))$ $+ (v^k_i(a^k) - v^k_i(a^l))$.

Hence, if the mechanism fails the check in one of the 3 steps above without summarization, then it will fail the check in the same step with summarization. □

The following example shows that summarization can facilitate faster identification of a non-strategyproof mechanism.

Example 6. *Consider an auction for multiple identical items, and suppose three instances are observed for the case of three bidders and two items. The auction*

instance	values	allocation	payments
1	((4, 8), (4, 9), (3, 6))	(0, 2, 0)	(0, 8, 0)
2	((0, 8), (4, 9), (5, 9))	(0, 1, 1)	(0, 3, 5)
3	((4, 9), (2, 8), (6, 10))	(0, 0, 2)	(0, 0, 10)

Table 4.2: Sequence of Instances: 3 Agents and 2 Identical Items

implements a VCG mechanism for the first two instances but deviates in instance 3. Now, consider the summarization function $s(v_{-i}) = \{v_j | \forall j \neq i, j$ *wins at least 1 item in the allocation without agent* $i\}$. *Note that* $s(v^1_{-1}) = s(v^3_{-1}) = s(v^2_{-3}) = s(v^3_{-3}) = \{(4, 9)\}$. *Consider the constraints when* $s(v_{-i}) = \{(4, 9)\}$. *Following the notation from Example 1, after instance 1, I get* $p_i(0) = 0$. *After the second instance, the verifier learns that* $p_i(1) = 5$, *and checks that the constraints (c)* $p_i(1) \geq p_i(0) - (v^1_i(0) - v^1_i(1)) = 4$ *and (d)* $p_i(1) \leq p_i(0) + v^2_i(1) - v^2_i(0) = 5$ *are satisfied. After instance 3, agent i wins one item and pays the price of 5, so the verifier checks that (b)* $p^3_i(1) = p_i(1) = 5$ *holds. After instance 3, the verifier also learns* $p_i(2) = 10$, *and checks that (c)* $p_i(2) \geq \max\{p_i(0) - (v^1_i(0) - v^1_i(2)), p_i(1) - (v^2_i(1) - v^2_i(2))\} = 9$. *However, the verifier also checks for the constraint (d)* $p_i(2) \leq p_i(1) + v^3_i(2) - v^3_i(1) = 9$, *which is violated (the price is too high). Hence, this mechanism is rejected.*

Without summarization none of the v_{-i} subnetworks have more than one node and the mechanism would not have been rejected until more instances were seen.

Summarization also provides an exponential improvement in space complexity. Let $d = max_i|V_i|$, $m = |A|$ and let n denote the maximal number of agents. With-

out summarization, I need d^{n-1} subnetworks, each with at most m nodes and thus $O(md^{n-1})$ nodes.

Theorem 7. *With summarization there are $O(md^w)$ nodes across all subnetworks, where w is the maximal number of agents required in a summarization, i.e. $w = \max_{i,v_i} |s(v_{-i})|$.*

Proof. Proof trivial and omitted. □

Thus, the space complexity is exponential in the worst-case number of agents required for summarization instead of the worst-case number of agents.

Example 7. *Consider the single item Vickrey auction with 4 agents, where $|V_i| = 10$. In this case, only the highest bid among bids of other agents matters. Without summarization, I would need to keep 1000 subnetworks, while with summarization, I only need to keep 10 subnetworks.*

4.5.2 Natural Payment Functions

So far I have focused on the constraints within a single v_{-i} subnetwork. The additional structure provided by *natural payments*, allows internetwork constraints which improve the speed with which a mechanism can be validated as strategyproof, or proved not to be strategyproof.

Consider again inequalities (4.3) and (4.4). Following Lavi et al. [42], define $\delta_{ab}(v_{-i}) = \inf\{v_i(a) - v_i(b) | v_i \in V_i, a \in E_i(f(v_i, v_{-i}))\}$. Then, I have:

$$-\delta_{ba}(v_{-i}) \leq p_i(a, v_{-i}) - p_i(b, v_{-i}) \leq \delta_{ab}(v_{-i}) \tag{4.8}$$

Definition 10 (Natural payment functions)**.** *A mechanism has a natural payment function if the agent-independent price function satisfies*

$$p_i(a, v_{-i}) - p_i(b, v_{-i}) = \delta_{ab}(v_{-i}) = -\delta_{ba}(v_{-i}), \tag{4.9}$$

for all pairs $< a, b >$.

The natural payment function is often satisfied by known mechanisms, for instance, it is typically satisfied by the VCG mechanism (this depends on the value domain) and the LOS mechanism. Natural payments permit the introduction of internetwork constraints, as demonstrated by the following lemma.

Lemma 2. *Consider a strategyproof mechanism with natural payment functions. Suppose for an alternative* a, v_{-i} *and* v'_{-i} *are such that* $\forall j \neq i$, *either* $v'_j = v_j$ *or* $v'_j(a) - v'_j(b) > v_j(a) - v_j(b)$ *for all* $b \neq a$. *Then* $p_i(a, v'_{-i}) - p_i(b, v'_{-i}) \leq p_i(a, v_{-i}) - p_i(b, v_{-i})$, *for all* $b \neq a$.

I introduce the following lemma to aid the proof of Lemma 2.

Lemma 3. *Consider a truthful social choice function* f *and some valuations* v *for which* $f(v) = a$. *Suppose* v' *is such that for each* j, *either* $v'_j = v_j$ *or* $v'_j(a) - v'_j(b) > v_j(a) - v_j(b)$ $\forall b \neq a$. *Then* $f(v') = a$.

Proof. [of Lemma 3] The proof follows from W-MON [42], which is a necessary condition for truthfulness.

Definition 11. *A social choice function* f *satisfies* W-MON *if* $\forall\, v \in V, i,$ *and* $v_i \in V_i$*:* $f(v) = a$ *and* $f(v'_i, v_{-i}) = b \Rightarrow v'_i(b) - v_i(b) \geq v'_i(a) - v_i(a)$.

Without loss of generality, suppose the first k agents are such that $v'_j = v_j$ and the rest of the agents are such that $v'_j(a) - v'_j(b) > v_j(a) - v_j(b)\ \forall b \neq a$. By W-MON, $f(v'_1, \ldots, v'_k, v'_{k+1}, v_{k+2}, \ldots, v_N) = f(v_1, \ldots, v_k, v'_{k+1}, v_{k+2}, \ldots, v_n) = a$. Now, applying W-MON again with agent $k+2$, $f(v'_1, \ldots, v'_k, v'_{k+1}, v'_{k+2}, v_{k+3} \ldots, v_n) = a$. I can repeatedly apply W-MON up to agent n to get $f(v_1, v'_2, \ldots, v'_n) = a$. □

Now, I prove Lemma 2.

of Lemma 2. First, note that for any v_i such that $f(v_i, v_{-i}) = a$, $f(v_i, v'_{-i}) = a$ by Lemma 3. Consider $\bar{v}_i$ such that for a fixed $b \in A$, $b \neq a$,

$$\bar{v}_i = \arg_{v_i} \inf\{v_i(a) - v_i(b) | f(v_i, v_{-i}) = a\}.$$

Then $\bar{v}_i(a) - \bar{v}_i(b) = \delta_{ab}(v_{-i})$. Now, since $f(\bar{v}_i, v'_{-i}) = a$, $\delta_{ab}(v_{-i}) \in \{v_i(a) - v_i(b) | f(v_i, v'_{-i}) = a\}$. So, $\delta_{ab}(v_{-i}) \geq \inf\{v_i(a) - v_i(b) | f(v_i, v'_{-i}) = a\} = \delta_{ab}(v'_{-i})$. Hence, $\delta_{ab}(v'_{-i}) \leq \delta_{ab}(v_{-i})$. Since $\delta_{ab}(v_{-i}) = p_i(a, v_{-i}) - p_i(b, v_{-i})$ (by natural payments), it follows that $p_i(a, v'_{-i}) - p_i(b, v'_{-i}) \leq p_i(a, v_{-i}) - p_i(b, v_{-i})$. □

Thus, given a constraint in the subnetwork for v_{-i} and some alternate reports v'_{-i} by all agents except one that satisfies the condition of Lemma 2, then I can add a corresponding constraint to the subnetwork for v'_{-i}, or vice versa. The additional constraint provides the following kind of inference: given $p_i(a, v_{-i}) \in [c, d]$ for some constants c and d, then $p_i(a, v'_{-i}) \leq d$. To illustrate the condition of Lemma 2, consider the case of single-minded bidders. Here, the condition is satisfied if each agent $j \neq i$ bids for the same bundle in v'_{-i} as in v_{-i}, and all agents $j \neq i$ who win in v_{-i} submit the same or larger value in v' while losing agents do not increase their bid in v'.

Example 8. *Consider a strategyproof mechanism with 2 agents and two items s and t, and suppose* $a = \{$*agent 1 wins s, agent 2 wins t* $\}$*. If* v_2' *is such that agent 2 values alternative a higher relative to all other alternatives, then if the alternative is a for both* v *and* v'*, agent 1 must not pay a higher price under* v'*.*

I can implement the internetwork constraints as follows. First, define the indicator function:

$$\mathbb{I}(a, v_{-i}, v'_{-i}) = \begin{cases} 1, & \text{if } v_j = v'_j \text{ or} \\ & v'_j(a) - v'_j(b) > v_j(a) - v_j(b)\ \forall b \neq a; \\ 0, & \text{otherwise.} \end{cases}$$

Then, for each alternative a and valuations v_i, v_{-i} such that $\mathbb{I}(a, v_{-i}, v'_{-i}) = 1$, I add the following constraints: $p_i(a, v'_{-i}) - p_i(b, v'_{-i}) \leq \max_{p_i \in \mathbb{F}_i(v_{-i})} [p_i(a, v_{-i}) - p_i(b, v_{-i})]$ to subnetwork v'_{-i} and $p_i(a, v_{-i}) - p_i(b, v_{-i}) \geq \min_{p_i \in \mathbb{F}_i(v'_{-i})} [p_i(a, v'_{-i}) - p_i(b, v'_{-i})]$ to subnetwork v_{-i}, where $\mathbb{F}_i(v_{-i})$ denotes the space of feasible prices defined by the constraint network. Note that $\max_{p_i \in \mathbb{F}_i(v_{-i})} [p_i(a, v_{-i}) - p_i(b, v_{-i})] = w^{v_{-i}}(b, a)$ and $\min_{p_i \in \mathbb{F}_i(v'_{-i})} [p_i(a, v'_{-i}) - p_i(b, v'_{-i})] = -w^{v'_{-i}}(a, b)$, where $w^{(v^{-i})}(a, b)$ denotes the weight on the arc from a to b in subnetwork v^{-i}.

4.5.3 Envy-freeness

The third example to illustrate the power of structural requirements is the property of *envy-freeness* [27].

Let $f_i(v)$ denote the allocation of goods to agent i defined by social choice function f.

Modified NetworkChecker.

Add a new step (d1) between steps (d) and (e):

(d1) For each v_{-i} such that $G(v_{-i}) \in N$ and $a^k \in f^H(v'_{-i})$:

- if $\mathbb{I}(a^k, v^k_{-i}, v_{-i}) = 1$, then
  - In $G(v^k_{-i})$: update the weight on the arc from a^k to each $a \in f^H(v^k_{-i}) \cap f^H(v_{-i})$ to be
    $\min\{w^{v^k_{-i}}(a^k, a), w^{v_{-i}}(a^k, a)\}$.
  - In $G(v_{-i})$: update the weight on the arc from $a \in f^H(v^k_{-i}) \cap f^H(v_{-i})$ to a^k to be
    $\min\{w^{v^k_{-i}}(a, a^k), w^{v_{-i}}(a, a^k)\}$.
- If $\mathbb{I}(a^k, v_{-i}, v^k_{-i}) = 1$, then reverse the role of v^k_{-i} and v_{-i}, and repeat previous step.

Definition 12. *Mechanism* $M =< f, \tilde{p} >$ *is envy-free if* $v_i(f_i(v)) - \tilde{p}_i(v) \geq v_i(f_j(v)) - \tilde{p}_j(v), \forall i, \forall j, \forall v \in V$.

This property is easily verified by checking that there is no agent that prefers an outcome-price pair that was received by another agent. Thus, this can be checked by adding an additional set of internetwork constraints.

Many strategyproof mechanisms are also envy-free. For example, VCG mechanisms in domains with superadditive valuations satisfy the property of envy-freeness [55]. As another example, it is easy to show that envy-freeness holds for any strategyproof mechanism in the domain of single-minded combinatorial auctions that is *fair*, which requires that for any set w and any two agents i and j such that the bids are $b_i = (w, v_i)$ and $b_j = (w, v_j)$ where $v_i > v_j$, then agent j should not win. The LOS family of mechanisms satisfy fairness and therefore all LOS mechanisms are envy-free.

4.6 Providing Intermediate Feedback to Participants

I describe in this section some intermediate feedback that can be provided to participants to guide their behavior even before a mechanism's strategyproofness (or lack thereof) is completely established. This is the main technical achievement of my work.

4.6.1 Partial Strategyproofness

First, recall that the verifier is able to check the outcome of a mechanism *before* the outcome is implemented and before payments are collected from agents and *veto* the result if the input-output instance provides proof that the mechanism is not strategyproof. This *veto* power is important in establishing the following useful result.

Theorem 8. *Given history H, consider an instance in which the reports of agents except i are $\hat{v}_{-i}$ and instance $(v_i, \hat{v}_{-i})$ has been observed by the mechanism for agent i's true valuation v_i. In this case, and conditioned on the mechanism passing the verifier in this instance, then agent i's best-response is to report her true valuation.*

Proof. Fix $\hat{v}_{-i}$. Condition on the case that all checks in the verifier pass. First, if the agent reports v_i then the mechanism must make the same decision as previously, by checks (a) and (b) in SimpleChecker. Now suppose the agent reports some $v_i' \neq v_i$. I argue that this cannot improve the agent's utility. Case 1. The mechanism selects an alternative a' already observed. Now I argue that the agent must actually prefer the outcome, a, that would have been selected given truthful report v_i. Either a' was observed after a in which case check (c) would have ensured that a was preferred to a' by an agent with type v_i, or a was observed after a' and check (d) would have ensured that a was preferred to a' by an agent with type v'. Case 2. The mechanism selects a new alternative, not previously observed for $\hat{v}_{-i}$. In this case, because v_i was already observed (with outcome a) then to pass check (c) it must be the case that outcome a would preferred by an agent with true type v_i. This concludes the proof. □

Note that the above theorem continues to hold even if the mechanism is ultimately

not strategyproof: it is still rational for an agent to bid her true type if v_i and $\hat{v}_{-i}$ (reports by the other agents) have already been observed. Moreover, the observation leads to the following two corollaries. Here, when truthful reporting is an *ex post* Nash equilibrium given knowledge $\tilde{V} \subset V$ about joint types, then every agent must maximize its utility by reporting its true type in equilibrium as long as every other agent is truthful and whatever the actual joint valuation $v \in \tilde{V}$.

Corollary 1. *Given history H, let $\tilde{V} \subset V$ denote some subspace of joint type space V for which all joint inputs $v \in V$ have been observed. Given knowledge that $v \in \tilde{V}$, and conditioned on the mechanism passing the verifier in this instance, then truthful reporting is an ex post Nash equilibrium.*

Corollary 2. *Given history H, and considering agent i with true type v_i, then if (v_i, v_{-i}) has been observed by the verifier for all v_{-i}, then conditioned on the mechanism passing the verifier in this instance truthful bidding is a dominant strategy for bidder i.*

I see that there are reasonable conditions under which a partially informed agent (e.g. with information about the space of possible types of other agents) should report its type truthfully even before the strategyproofness of a mechanism is fully verified. Moreover, these observations suggest that it will be useful for a verifier to *publish* the set of types for which the mechanism is provably strategyproof. Note also that these results can all be rephrased in terms of *summarizations* of reports from other agents when summarization is used.

A somewhat weaker guarantee is also available to a bidder in the case that report $\hat{v}_{-i}$ has been observed but not in combination with the agent's type v_i. This guarantee

is provided in the context of an *adversarial* mechanism, which *seeks to minimize the utility to agent i while passing the verifier in this instance.* The earlier guarantees are provided for any mechanism.

Theorem 9. *Given history H, then agent i maximizes her worst-case utility against an adversarial mechanism by reporting truthfully whenever report $\hat{v}_{-i}$ has been observed, contingent on the mechanism passing the verifier in this instance.*

Proof. Worst-case utility refers to the minimum utility achieved over all possible choices that can be made by the mechanism in responding to the input, while still passing the verifier. Fix v_i and $\hat{v}_{-i} \in H$. First suppose the agent is truthful. In this case the best the adversary can do, in terms of minimizing the utility to the agent, and still pass the checks (e.g. in SimpleChecker) is to select the preferred alternative for the agent in $a \in f^H(\hat{v}_{-i})$. Choosing some less-preferred alternative in $f^H(\hat{v}_{-i})$, or some less-preferred alternative outside of $f^H(\hat{v}_{-i})$ would violate check (d) in SimpleChecker. Now suppose the agent is untruthful and reports $v_i' \neq v_i$. In this case, the mechanism can always choose one of the current alternatives and pass since check (c) would not be triggered and check (d) is satisfied by choosing $a' \in f^H(\hat{v}_{-i})$, the most-preferred alternative with respect to v_i'. Note that the alternative selected when reporting v_i is at least this good. Also, if the mechanism chooses to select some new alternative then this must necessarily be even worse for the agent (since the mechanism is adversarial). This completes the proof. □

4.6.2 Metrics for Partial Verification

I introduce two quantitative metrics for the degree to which a mechanism's strategyproofness has been partially verified. First, Corollary 1 can be used to define a lower-bound on the probability that truthful reporting is an *ex post* Nash equilibrium, given history H and given a distribution on types of bidders.

Corollary 3. *Given history H and probability distribution g on agent types, then truthful bidding is an ex post Nash equilibrium conditioned on the mechanism passing the verifier in this instance, with probability:*

$$Pr(SP|H) = \sum_{v \in V^N} g(v)\mathbb{I}^H(v) \tag{4.10}$$

where $g(v)$ is the probability that agents have types v and $\mathbb{I}^H(v)$ is an indicator function, and equal to 1 if and only if $v \in v^H$.

I refer to this metric as the *probability-of-strategyproofness.* In combination with summarization, this becomes:

$$Pr(SP|H) = \sum_{v \in V^N} g(v)\mathbb{I}^H(s(v_{-1}), \ldots, s(v_{-N})) \tag{4.11}$$

where $\mathbb{I}^H(s(v_{-1}), \ldots, s(v_{-N}))$ is an indicator function, and equal to 1 if and only if summarizations $s(v_{-i})$ for all i have been observed by the mechanism given history H. A probability-of-strategyproofness metric can also be defined for a single agent i, where the average is computed with respect to the marginal distribution on the reports of agents $\neq i$, given agent i's type v_i.

A complementary measure is provided by the *price-flexibility* metric. This metric takes into account the range of prices still available to a mechanism in setting prices

on alternatives not yet observed. Let $\mathit{Feas}^H(v) \subseteq A$ denote the set of alternatives that can be selected without failing the verifier given input v and history H. If v has been observed this is the singleton containing the alternative $f^H(v)$ that was previously selected. But, even when v has not been observed I have restrictions on alternatives that can be selected. There can be alternatives for which, if selected, there is no price that can be assigned to satisfy checks (c) and (d) in the verifier for all agents.

For $a \in \mathit{Feas}^H(v)$, I define the price flexibility on a, denoted $\mathit{flex}^H(v, a)$ as:

$$v_i(a) + \inf_{b \in f^H(v_{-i})} \{p_i^H(b, v_{-i}) - v_i(b)\} \\ - \left[\sup_{b \in f^H(v_{-i})} \left\{ p_i^H(b, v_{-i}) + \sup_{v_i' \in v_i^H(b, v_{-i})} (v_i'(a) - v_i'(b)) \right\} \right] \tag{4.12}$$

This represents the range of prices available to the mechanism without violating the checks in the verifier, and is non-empty since a is in the feasibility set. Now, for valuation profile $v = (v_1, \ldots, v_N)$, define the *price-flexibility*, as:

$$PF^H(v) = \sum_{v \in V^N} g(v) \left[\frac{1}{N^H(v)} \sum_{a \in \mathit{Feas}^H(v)} \mathit{flex}^H(v, a) \right], \tag{4.13}$$

where $N^H(v) = |\mathit{Feas}^H(v)|$. Notice that if all inputs v_{-i} have been observed for v_i then $PF^H(v_i) = 0$. This is as I would expect: a fully-verified mechanism has no flexibility and must continually make the same decisions and assign the same prices as in the past.

4.7 Experiments: Passive Verification

I present experimental results to demonstrate that the verifier can impose useful restrictions on the mechanism design space (via summarization, natural payments,

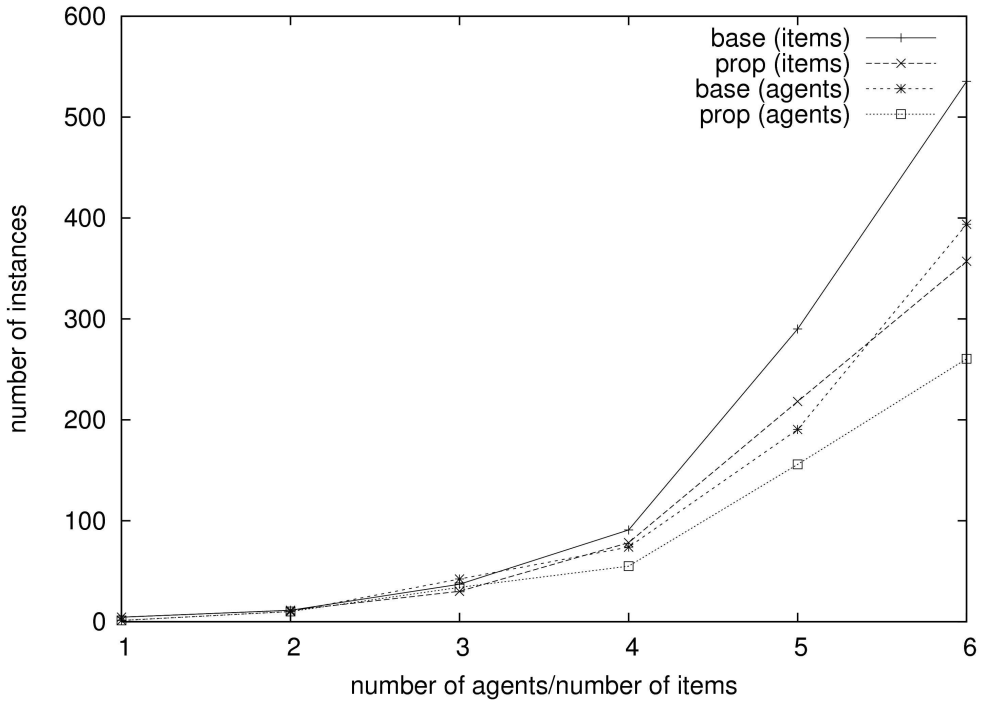
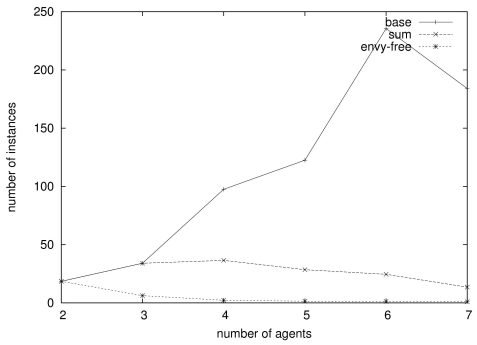


Figure 4.3: Number of instances before failure in a first-price multi-item auction.

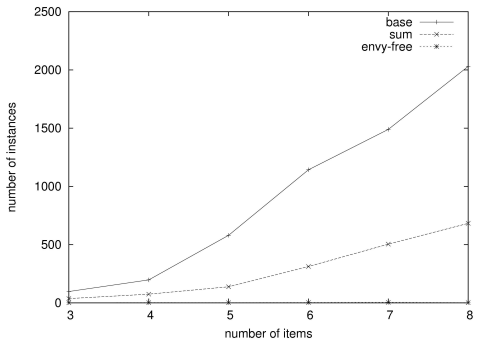
and envy-freeness) and increase the confidence in truthful mechanisms and accelerate the detection of failure in manipulable mechanisms.

For illustrative purposes, the experimental results are presented for a mixture of known and invented mechanisms. I consider both Vickrey and first-price auctions for the allocation of multiple identical items. In addition, I consider single-minded bidders in a combinatorial auction (with multiple non-identical items) and the LOS mechanism [44] as well as the LOS greedy allocation rule in combination with VCG payments (this mechanism is called *greedyVCG*, and is manipulable; see example 5 for more details on the LOS mechanism and the LOS greedy allocation rule). I also find

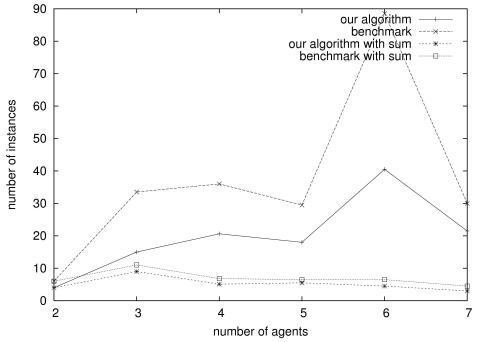
it useful to construct an artificial, manipulable mechanism for single-minded bidders using local search techniques combined with VCG payments. I call this mechanism *localVCG*.



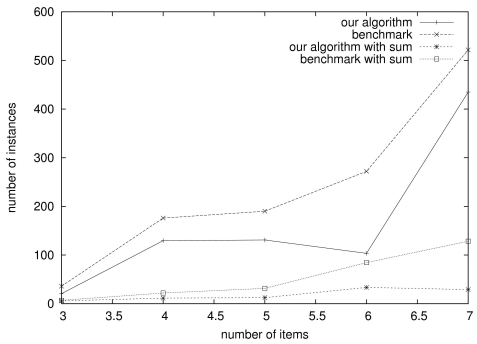
(a) Number of instances before failure in greedyVCG: Agents.



(b) Number of instances before failure in greedyVCG: Items.



(c) Number of instances before failure in localVCG: Agents.



(d) Number of instances before failure in localVCG: Items.

Figure 4.4: Exploring Verification with Single-Minded Bidders.

To provide a baseline I compare with an algorithm that simply checks for violations

of (A1). This algorithm is sound but not complete. Actually, it completely fails to reject the greedyVCG mechanism, which only makes errors of type (A2). On the other hand, it is sound against first-price auctions, which only violate (A1). The localVCG mechanism is useful because it violates both (A1) and (A2), permitting a comparison between the baseline and my verification methods.

4.7.1 Domain 1: Multi-item Auctions

I consider a first-price multi-item auction with g identical goods and construct agent valuations by defining the marginal value of the kth item for agent i as independently drawn from a uniform distribution over $\{0, 1, \ldots, z-1\}$. Here, z denotes the max number of values in the value domain for any one of these items. With g items the size of the type space for each agent is $m = z^g$. I consider the verifier both with and without the internetwork constraints that arise from the requirement of natural payment functions (which hold for the VCG mechanism in this auction environment), but do not consider the use of summarization.

Figure 4.3 illustrates the number of instances observed before failure. Fixing $z = 5$, I first vary the number of items g from 1 to 10 for $n = 3$ agents (i.e. with $m = z^g = 5^g$ types per agent, m^3 different instances, and at most m^2 networks). Second, I vary the number of agents n from 1 to 10 for $g = 3$ items (i.e. with $m = 5^3 = 125$ types per agent, 125^n instances and at most 125^{n-1} networks). For each experiment each point represents an average over 10 runs. The NetworkChecker is tested with and without internetwork constraints due to natural payments, and labeled *base* and *prop* accordingly.

Observe that internetwork constraints considerably speed up the verification process. In addition, since at most n new nodes are created for each instance, the number of nodes in the graph is roughly proportional to a multiple of n times the number of instances observed before failure and the running time and space complexity of the verifier is roughly linear in the number of instances observed. Thus, imposing natural payments and enabling internetwork constraints improves space and time complexity.

4.7.2 Domain 2: Single-Minded Bidders

For single-minded bidders I generated a distribution of instances by using the *L4 Legacy distribution* of the Combinatorial Auctions Test Suite (CATS) [45]. I discretized the type space by rounding the values to the nearest 250 (the values fell into a range of [0, 3000]). There are g distinct items, and each agent can choose any subset of the g items, and have a value $\in \{0, 250, \ldots, 2750, 3000\}$ for the desired bundle. The results are averaged over 10 trials.

In this experiment I impose both summarization and envy-freeness, which is a reasonable structural requirement in this environment (as noted in Section 4.5). As a summarization function I adopt the summarization function defined in Example 5 for the VCG mechanism, which is also valid for the LOS mechanism and therefore reasonable to impose. I also compare with the benchmark algorithm, which fails to catch the non-strategyproofness of greedyVCG but is effective for localVCG.

Figures 4.4 (a) and (b) illustrate the results for Network Checker ("base"), and also with summarization ("sum") and with the additional structure provided by

envy-freeness ("envy-free"), which is used here without summarization [5] In Figure 4.4 (a) I vary the number of agents n while fixing the number of items at $g = 3$ ($m = 13^7$ types, since there are $2^3 - 1$ bundles of 3 items). In Figure 4.4 (b) I vary the number of items g while fixing the number of agents at $n = 4$ ($m = 13^{2^g-1}$). The use of summarization provides a significant speed-up, and the imposition of envy-freeness makes the verification of a non-strategyproof mechanism almost immediate. As expected, summarization becomes more important as the number of agents increases, and its benefit increases, because without summarization it gets less and less likely that the history contains observations relevant to a new instance.

Figures 4.4 (c) and (d) illustrate the results for NETWORK CHECKER with and without summarization and the benchmark algorithm with and without summarization, this time with the localVCG mechanism. Local search is modeled after the stochastic local search algorithm by Hoos and Boutilier [33]. The only difference is that my search is non-stochastic: I do greedy hill climbing with their neighborhood definition and improvement criteria. Local search is used first to approximate a solution with all agents. During this phase I also track the best solution found for each marginal economy. For payments, I then run local search with a smaller number of search steps on each of the marginal economies and adopt for the economy without i the best allocation discovered in the main search and in the explicit search in the marginal economy. In addition to failing (A2), localVCG can also fail (A1) because the price to a buyer can now depend on the search path adopted in the main economy, which can on the buyer's own bid.The same summarization function is adopted as

[5]For single minded bidders, the internetwork constraints from natural payments used in NETWORKCHECKER are trivially satisfied, and do not improve the checking.

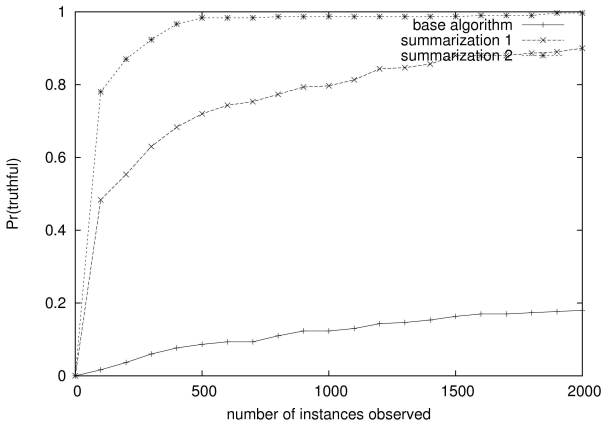
above. I again vary the number of agents, and then the number of items. In comparison with the benchmark algorithm, the verifier detects non-strategyproofness more quickly both with and without summarization, with the benefit over the benchmark with summarization most noticeable as the number of items increases.

4.7.3 Computing Metrics

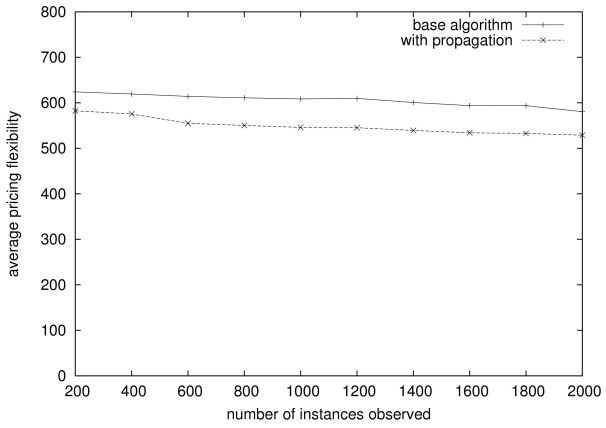
I have also experimented with the *probability of strategyproofness* (PrSP) and *price flexibility* (PF) metrics defined in Section 4.6. These metrics allow the verifier to provide feedback to agents before finally verifying that a mechanism is strategyproof (or not). I present results for single-minded combinatorial auctions and the (strategyproof) LOS mechanism. Monte Carlo analysis is used to estimate PrSP and PF over all possible valuations $v \in V$, where the valuations are drawn according to the CATS Legacy distribution.

Figure 4.5 (a) displays the estimated PrSP for LOS in a domain with 4 agents and 3 items. The results are averaged over 100 trials. I consider NetworkChecker without summarization and then with the two summarization functions described in Example 5 (the first is the more general VCG rule, the second is the rule designed for LOS.) Summarization greatly improves the probability that truthful reporting is a dominant strategy at any given point in the verification process, especially the second summarization function which is designed for LOS. Thus, in deploying the LOS mechanism it would be useful to instruct the verifier to adopt this strict summarization function in order to accelerate verification.

Figure 4.5 (b) displays the estimated PF for LOS in a domain with 3 agents and 3



(a) Estimated probability of strategyproofness: Effect of summarization



(b) Estimated pricing flexibility: Effect of internetwork constraints (from natural payments)

Figure 4.5: Evaluating metrics for partial strategyproofness in verification of the LOS mechanism in single-minded combinatorial auctions.

items. The results are averaged over 300 trials. I compare NetworkChecker with and without the imposition of natural payments and thus with and without the availability of internetwork constraints. I find that internetwork constraints are somewhat effective in reducing the price flexibility, although the effect is not as pronounced as that of summarization on the probability of strategyproofness.

Chapter 5

A Decentralized Auction Framework to Promote Efficient Resource Allocation in Open Computational Grids

Abstract

Computational grids enable the sharing, aggregation, and selection of geographically distributed computational resources that can be used for solving large-scale and data intensive computing applications. Computational grids are an appealing target application for market-based resource allocation especially given the attention in recent years to "virtual organizations" and policy requirements. In this chapter, I present a framework to promote truthful, decentralized, dynamic auctions in computational

grids. Rather than a fully-specified auction, I propose an open, extensible framework that is sufficient to promote simple, truthful bidding by end-users while supporting distributed and autonomous control by resource owners. The auction framework facilitates the use of resource prediction in enabling an expressive language for end-users and highlights the role of *infrastructure* in enforcing rules that balance the goal of simplicity for end users with autonomy for resource owners. The technical analysis leverages simplifying assumptions of "uniform failure" and "threshold-reliability" beliefs.

Computational grids enable the sharing, aggregation, and selection of geographically distributed computational resources that can be used for solving large-scale and data intensive computing applications. The distributed ownership and use of computational resources make market mechanisms, where prices coordinate decision making both within and between organizations, a good fit for solving resource allocation problems in grids. Market mechanisms promote efficiency: in the short run, resources are allocated to the best use, and in the long run, prices provide signals for long-term investments that enable the best policy decisions. Market mechanisms also enable policy and, together with appropriate macroeconomic controls, may offer *the* compelling solution to the problem of "virtual organizations" that has taxed the grid community [24].

Substantial advances have been made in grid software and its conceptual framework in recent years. However, real-world open science grids are still limited in size to hundreds of sites and thousands of computers. The complexity in expressing a user's needs and the lack of flexibility and autonomous control in resource management in existing systems have been some of the main obstacles. Users should be able to focus on computational experiments, not gaming, to obtain sufficient resources, and organizations should be able to share resources in a flexible and locally managed manner.

The vision for market-based computational grids and the broad scope of the *EGG* project are described in [14]. EGG is a collaborative project between Harvard and Boston University, involving high-energy physicists, computer scientists, and economists. The EGG architecture brings together environment computing and

economic principles to create a vision of grid computing that realizes the promise of autonomy and openness. Here I provide additional details about the *microeconomic* aspect of EGG and ignore macroeconomic issues (e.g. currency supply, inflation, questions of policy, etc.). These are of equal importance and part of the fabric of EGG, components of which are currently being implemented and will be deployed as a practical platform.

The auction framework is designed around the following design principles:

- **Efficiency:** in the long-run, as the system adapts, jobs should be allocated to resource owners best able to run the job at low cost, subject to other policy constraints.

- **Simplicity:** A simple bidding language is provided for end-users and the framework is strategyproof for users and thus non-manipulable.[1] Strategyproof mechanisms mean that users need not engage in wasteful counterspeculation about how best to game the system and can focus on *using* grids rather than gaming grids.

- **Decentralized control**: Resource owners retain control and flexibility over their own resources. A complete auction (e.g. with pricing policy) is not specified, but rather the framework provides a minimal set of rules that ensure truthfulness.

- **Extensibility:** Resource owners can replace and improve components (e.g. pricing strategies, resource estimation algorithms) over time. This supports

[1]Given the simplifying assumptions of uniform-failure and threshold-reliability beliefs, to be discussed in Section 5.1.2.

innovation.

Details about *how* resource owners can implement algorithms for resource estimation or pricing is provided in the subsequent chapters, Chapters 6 and 7. In this chapter, I focus on the design of the auction framework.

I consider a dynamic model in which jobs arrive over time and allocation decisions need to be made dynamically. The dynamics complicate the question of strategyproofness (introducing new opportunities for manipulation). They also make the problem a poor fit for existing technologies, such as combinatorial auctions [20] and combinatorial exchanges [58] that assume all bids are present at the same time.

A user can bid for resources at any time by describing her job and annotating the job with a *value schedule* that defines her willingness to pay as a function of the job completion time. This bid spawns a *one-time reverse auction* in which qualified resource providers compete for the right to execute the job. The auction framework completes this auction by consulting *price tables* maintained by each resource provider (representing the bids of the resource providers) and the *resource estimates* of each resource provider for the job. The auctioneers (and not the sellers) look up a price to quote to the user in the context of the reverse auction after performing payoff-maximization on behalf of the user. The auction framework enforces rules on price tables and resource estimation that ensure strategyproofness for end users. Technical assumptions about uniform-failure and threshold-reliability beliefs are made in order to support claims about non-manipulability.

5.1 Model and Key Components

I consider the problem of allocating computational resources on compute servers when jobs can have different size/lengths and arrive dynamically. A similar problem of allocating storage space and time can be defined for file servers, but is not analyzed in this thesis. I do not study combinatorial issues related to how a user structures her work across multiple jobs (see Section 5.4); rather I assume that each user has a predetermined set of jobs.

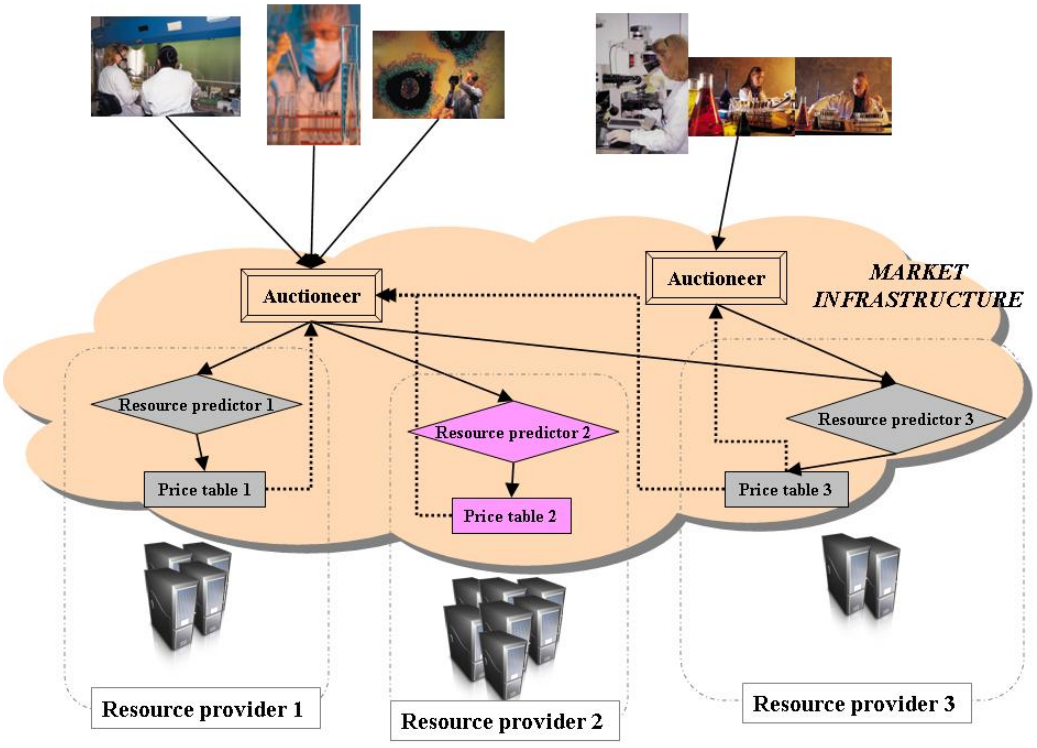


Figure 5.1: Control Flow: resource providers own and control their price tables, resource predictors, and schedulers, but the market infrastructure imposes constraints on the price tables and resource predictors.

5.1.1 Actors

The model consists of four types of entities: the market infrastructure, auctioneers, users, and resource providers:

Market infrastructure: Trusted by both users and resource providers. One important role is to impose constraints on the resource providers (e.g. on the price tables, resource estimates) in order to ensure strategyproofness. The infrastructure also maintains and publishes statistics that the auctioneers can use to compute the reliability metrics for resource providers.

Auctioneers: Part of the infrastructure, a one-time auctioneer is created on behalf of a user whenever she runs a job. The auctioneer is personalized in terms of the collection of resource providers from which the auctioneer should solicit offers (i.e., the user's own view of the grid.)

Users: Individuals or organizations who wish to employ the resources available on the grid to run jobs, e.g., a physicist who wants to run Monte Carlo simulations.

Resource Providers: Resource providers maintain a collection of computational resources and schedule and perform jobs that are won through auctions. Each resource provider maintains a resource prediction module and price tables within the microeconomic framework and also runs its own local scheduler.

5.1.2 Modeling Users: Jobs and Beliefs

Each job has a description and an arrival time and a user has a value schedule for the completion time of the job and a minimum reliability threshold for resource providers from which the user wishes to solicit offers. (Recall that one role of the

market infrastructure is to maintain a reliability metric for each resource provider. This is a measure of the frequency with which a resource provider has successfully completed a job by the scheduled completion time. Note that while reliability is a similar concept to reputation, I emphasize that reliability here is objectively measured and non-manipulable.) The arrival time of a job is the earliest time the value of the job is known to the user and the job can be described. A user can be associated with multiple jobs. Taken together, type, $\theta_i = (J_i, w_i, a_i, \gamma_i)$, completely characterizes the information about a job:

- J_i is a string describing the job to be submitted. J_i can be thought of as a list of fields for executable files, input files, number of loops, etc., and specifies how to perform the job (e.g., file location). The default syntax and semantics are defined by the market infrastructure. I assume that attributes can be extracted from J_i, to be used for resource prediction.

- $w_i : [a_i, a_i + \Delta] \rightarrow \mathcal{R}_{\geq 0}$ is a function of the *completion time* where $w_i(\tau)$ is a nonnegative real number representing the willingness to pay for job completed by time τ. I call w_i the value schedule for job i. I assume every job has *bounded patience*, i.e., there exists a minimal constant $\Delta \in \mathcal{R}_{\geq 0}$ such that $w_i(\tau) = 0$ for all $\tau > a_i + \Delta$, for all jobs i. I assume Δ is known to the market infrastructure and resource providers. I call Δ *user patience*.

- $a_i \in \mathcal{R}_{\geq 0}$ is the arrival time

- $\gamma_i \in [0, 1]$ is the minimum tolerable reliability of the resource providers to which the job should be submitted.

Throughout this paper, let d_i denote the latest time τ for which $w_i(\tau) > 0$, i.e., the maximal deadline or departure time of the job.

A bid for a job consists of the job description, value schedule and an optional parameter specifying a *minimal reliability*. The arrival time is defined by when a user can bid, and is not explicitly included in a bid. For example, a bid with a linear value schedule could define ("download Atlas 5.x[2]",(10, 2,Apr-01-06 00:00:01), 99%), where the second component describes a monotonically decreasing willingness-to-pay of $(2 + (10 - 2)(t_d - \tau))/(t_d - t_0)$, where τ is the time of completion, t_d is the maximal deadline (Apr-01-06 00:00:01 in the example), and t_0 is the time at which the bid is submitted to the auctioneer. The auctioneer should then only accept offers from resource providers with reliability $\geq 99\%$. A simple special case is a constant willingness-to-pay with a hard deadline. I assume that a user has no value for receiving a completed job outside of a job's arrival-departure interval or for an incomplete job.

I assume that there is a partial order $\succ$ on job descriptions such that $J' \succ J$ whenever a job with description J' has as much value as a job with description J, to a user who has a job with description J. In other words, I assume that if $J' \succ J$, then any user with a job with description J values another job with description J' as much, which can potentially be restrictive. For example, a job with description J' may require a more recent software installation version number with backwards compatibility, have more loops, or run on a larger input file. For instance, a job with 2000 Monte Carlo iterations is better than a job with 1000 of the same Monte Carlo iterations. A job description may include fields for software versions, number

[2]http://atlas.web.cern.ch/Atlas/index.html

of Monte Carlo iterations, and the size of the input file so that this partial order can be defined. Given this I formalize what it means for a user to be "single-minded" in my model:

Definition 13. *Let J_i be the description for job i. Define $\nu_i(J_i', \tau)$ to be the value to the user (who submits job i) if she is given some completed job with description J_i' by time τ. A user is single-minded if:*

$$\nu_i(J_i', \tau) = \begin{cases} w_i(\tau), & \text{if } J_i' \succ J_i; \\ 0, & \text{otherwise.} \end{cases}$$

The technical results about strategyproofness rely critically on two simplifying assumptions about user beliefs. Threshold-reliability beliefs are beliefs that a user holds about the comparative reliability across resource providers (and for a particular job) while uniform-failure beliefs are beliefs that a user holds about the comparative reliability across different jobs (and for a particular provider). Denote the set of resource providers with reliability metric at least γ_i by Γ_i.

Definition 14 (threshold-reliability beliefs)**.** *A user has threshold-reliability beliefs for job i with description J_i if she holds belief that all resource providers $\in \Gamma_i$ are equally likely to successfully complete a specific job with description J_i' such that $J_i' \succeq J_i$, and that success is independent of the completion time.*

By this assumption, a user does not reason about the probability that the winning resource provider will successfully complete a specific job. For example, if $\gamma_i = 90\%$, the user is indifferent between a resource provider with reliability 93% and another resource provider with reliability 95% (given that both yield the same payoffs at their respective scheduled completion times).

Definition 15 (uniform-failure beliefs)**.** *A user has uniform failure beliefs if she holds beliefs that Pr(A job with description J_i' successfully completes if scheduled with resource provider k) $\leq$ Pr(A job with description J_i successfully completes if scheduled with resource provider k), for all jobs with description $J_i' \succ J_i$, for all k with reliability at least γ_i.*

With uniform-failure beliefs, a user believes that she cannot improve the probability that a job completes by replacing it with another job with description $J_i' \succ J_i$. Note that the uniform failure assumption is about completion probabilities *if scheduled by the same resource provider.*

5.1.3 Resource Prediction: Auction Rules

A resource predictor takes the description J_i and the arrival time a_i of job i as inputs and outputs a vector Q_i of estimated resource requirements. A resource provider can use any learning algorithm it chooses, but the market infrastructure imposes a monotonicity requirement that constrains the estimates for different jobs and also how the model can be refined across time. I assume that the learned model has a concise representation so it can easily be verified by the infrastructure (e.g., Naïve Bayes, continuous Gaussian linear regression, decision tree, etc.). A resource provider can also select periods in which it is *inactive*, where it does not generate any resource estimates (but simply queues jobs arriving in this period such that an estimate will be given once the inactive period is over, if these jobs are still not scheduled by any other resource provider).

Let $Q_{ik}^t = \hat{R}_k^t(J_i) \in (\mathcal{R}_{\geq 0})^L$ denote the L-dimensional vector of resource require-

ments given J_i produced by the learned model used by resource provider k to estimate resource requirements at time t.

Definition 16 (monotonicity w.r.t. job description)**.** *Resource provider k's estimator $\hat{R}_k^t(J)$ is monotonic if $\hat{R}_k^t(J') \geq \hat{R}_k^t(J)$ for all $J' \succ J$, for all t.*

Monotonicity with respect to job description is a reasonable property. For example, using a larger input file or a running a larger number of Monte Carlo iterations of a loop should not require less disk space or shorter runtime. Moreover, I can assume without loss that if $J' \succ J$ requires less resources than J, then a resource provider can perform J' for the user instead, and set $R_k^t(J) = R_k^t(J')$. It should also hold when estimating network resources for software installation with backwards compatibility. For instance, installing Atlas 5.x should never have a lower resource estimate than installing Atlas 4.x because if Atlas 5.x actually requires less resources (and is backwards compatible), then whenever Atlas 4.x needs to be installed, Atlas 5.x can be installed instead.

Monotonicity with respect to time is imposed to ensure that users cannot benefit from misreporting the arrival time, i.e., delaying job submission, hoping to get a lower price from a lower future resource estimate. Let Δ be the user patience (see Section 5.1.2):

Definition 17 (time monotonicity)**.** *Resource provider k's estimator $\hat{R}_k^t(J)$ is monotonic with respect to time if $\hat{R}_k^{t'}(J) \geq \hat{R}^t(J)\ \forall\ J$, for all $t < t' < t + \Delta$ such that resource provider k is not inactive in either t and t'.*

If a resource provider wishes to introduce a new model with lower estimates for

some jobs then it can become *inactive* for a period of at least Δ. No inactive period is required if the new model does not decrease resource estimates.

In this vision, I expect there to be competition among resource providers. In the long run, as competition grows, it will be in the interest of a resource provider to accurately estimate its required resources. If it underestimates the resource requirements, the resource provider will not be able to complete an assigned job, damage its reliability, or receive a lower price. On the other hand, if it overestimates, it will forego a chance to earn revenue by losing the job to a resource provider that offers a lower price.

5.1.4 Price Tables: Auction Rules

A resource provider maintains a *price table* visible to the market infrastructure, and a function of a vector of resource requirements Q and the completion time τ. It can be thought of as a table of dimension $1 + dim(Q)$ where $dim(Q)$ is the dimension of the vector Q and the last dimension corresponds to period in which the job is to be completed. The price table is used by the market infrastructure to generate a resource provider's bid, given a reverse auction for a particular job in some time period.

The ability of resource providers to fill the price table based on their own objectives is a key feature that provides autonomous control. A resource provider can update its price table entries to incorporate scheduling constraints, meet a target load level, or to improve its revenue. However, in order to maintain strategyproofness for end-users, the market infrastructure imposes that the price table entries are *admissible*.

Let $\phi_k^t(Q,\tau)$ denote the price table entries quoted by resource provider k in period t for resource requirement Q for the completion time τ. $\phi_k^t(Q,\tau)$ can be interpreted as: the price (quoted in time t) that resource provider k wishes to receive for completing a job with resource requirements Q, if it were scheduled to complete by time τ. Given a job, a *scheduled completion time* τ_k^* for each resource provider k is selected by the auctioneer, such that the payoff of the user is maximized.

Let $Q[l]$ denote the lth component of the vector Q, and let Δ be the user patience.

Definition 18 (admissibility)**.** *Price table entries*

$\phi_k^t(Q,\tau)$ *are admissible if both:*

$$\phi_k^t(Q'[l],\tau) \geq \phi_k^t(Q[l],\tau) \quad \forall Q'[l] > Q[l], \forall l, \forall \tau, \forall t \tag{5.1}$$

$$,\phi_k^{t'}(Q,\tau) \geq \phi_k^t(Q,\tau) \quad \forall t' > t, \forall t' \leq \tau \leq t+\Delta, \forall Q. \tag{5.2}$$

Prices are admissible if (1) the price table entries are nondecreasing in each component of the resource requirements, e.g., if Q consists of runtime (r) and disk space (s)

$$\phi_k^t((r',s),\tau) \geq \phi_k^t((r,s),\tau)\ \forall r' > r$$

$$\phi_k^t((r,s'),\tau) \geq \phi_k^t((r,s),\tau)\ \forall s' > s$$

and (2) the price table entry for a given completion time within the user-patience window $(t+\Delta)$ does not decrease over time. Note that this still allows a resource provider to decrease prices outside of the user-patience window.

Example 9 (Admissible prices)**.** *Consider the following 2-dimensional price table where each row corresponds to estimated memory usage, and each column to the*

scheduled completion time. Let t_0 denote the current time. Suppose the intervals for memory are $[0, 2)$, $[2, 4)$, $[4, 8)$ Megabytes, and the intervals for a job's scheduled completion time is $[t_0, t_0 + 1)$, $[t_0 + 1, t_0 + 2)$, $[t_0 + 2, t_0 + 3)$,$[t_0 + 3, t_0 + 4)$ hours.

Suppose the table entries are currently:

memory	$\tau \in [0, 1)$	$\tau \in [1, 2)$	$\tau \in [2, 3)$	$\tau \in [3, 4)$
[0,2)	*5*	*4*	*6*	*10*
[2,4)	*17*	*13*	*15*	*12*
[4,8)	*20*	*15*	*21*	*25*

Suppose $\Delta = 3$ hours. Then, by admissibility the resource provider can increase or decrease all the entries beyond the user patience window. For example, the entries 10, 12, and 25 from the fourth column of the table can be updated to 4, 16, and 18, respectively. However, the other entries cannot be decreased. An updated price table with admissible entries may look like:

memory	$\tau \in [0, 1)$	$\tau \in [1, 2)$	$\tau \in [2, 3)$	$\tau \in [3, 4)$
[0,2)	*8*	*4*	*7*	*4*
[2,4)	*18*	*17*	*15*	*16*
[4,8)	*24*	*18*	*100*	*18*

Note that earlier completion times may have higher prices. While I might expect later completion times to have lower prices than earlier ones, the system does not require this.

5.2 Auction Framework

I now define the rules of the reverse auction that is created on-the-fly each time a user submits a job. As well as defining the auction, which determines which resource provider (if any) gets to run the job, the auction framework imposes the following additional assumptions:

- When a user reports $\hat{J}_i$, $\hat{J}_i$ is the job that will be performed (a user cannot report $\hat{J}_i$ and then have the resource perform J_i instead).
- The user makes no payment and does not receive an incomplete job if the job is not completed by the scheduled completion time.

Consider the following auction protocol:

1. On receiving bid $(\hat{J}_i, \hat{a}_i, \hat{w}_i, \hat{\gamma}_i)$:
   (a) Let $\hat{d}_i$ be the latest time τ such that $\hat{w}_i(\tau) > 0$.
   (b) Consider the resource providers with reliability at least $\hat{\gamma}_i$. Ensure the resource provider does not change its estimator function based on information about the job.
      i. For each resource provider k, compute the resource estimates $\hat{R}_k^{\hat{a}_i}(\hat{J}_i)$. Let $\hat{r}_k$ denote the estimated runtime.
      ii. The relevant price table entries are revealed to the auctioneer by the market infrastructure. Given the price table entries at time $t = \hat{a}_i$, the auctioneer computes τ_k^* where τ_k^* is the earliest time in $[\hat{a}_i + \hat{r}_k, \hat{d}_i]$ such that:

$$\tau_k^* = \arg \max_{\tau \in [\hat{a}_i + \hat{r}_k, \hat{d}_i]} \{\hat{w}_i(\tau) - \phi_k^{\hat{a}_i}(\hat{R}_k^{\hat{a}_i}(\hat{J}_i), \tau)\} \tag{5.3}$$

   (c) Select a resource provider k^* with maximal utility. Denote the set of resource providers with reliability at least $\hat{\gamma}_i$ by $\hat{\Gamma}_i$. Let k^* be

$$k^* = \arg \max_{k \in \hat{\Gamma}_i} \{\hat{w}_i(\tau_k^*) - \phi_k^{\hat{a}_i}(\hat{R}_k^{\hat{a}_i}(\hat{J}_i), \tau_k^*)\} \tag{5.4}$$

The price that a user submitting job i whose type is $\hat{\theta}_i = (\hat{J}_i, \hat{a}_i, \hat{w}_i, \hat{\gamma}_i)$ faces is:

$$p_i(\hat{J}_i, \hat{a}_i, \hat{w}_i, \hat{\gamma}_i) = \phi_{k^*}^{\hat{a}_i}(\hat{R}_{k^*}^{\hat{a}_i}(\hat{J}_i), \tau_{k^*}^*) \tag{5.5}$$

The job is not scheduled if all resource providers have prices higher than value. Break ties in favor of earlier times.

(d) Collect payment from job i and place in escrow.

(e) The market infrastructure requires that resource provider k^* encrypts the outcome of job to prevent job i from accessing outcome until $\tau_{k^*}^*$.

2. On the scheduled completion time $\tau_{k^*}^*$:

(a) Check whether job is completed. Update resource provider reliability.

(b) If completed, allow job to transfer outcome of computation.

(c) If completed, transfer payment from escrow to resource provider. Else return payment to job.

A resource provider can employ a scheduling algorithm of choice, and retains autonomy to decide when to actually schedule a job. Notice though, that to get paid for a job it wins, it must schedule the job so the job can be completed on or before the *scheduled completion time* τ_k^*, specified by the auctioneer.

Example 10 (Looking up prices in a price table)**.** *Given a job request and a price table then a row is chosen based on the estimated resource requirement of the job. The ultimate price is determined based on the optimal scheduled completion time, which is chosen by the reverse auction to be where* value - price *is maximal among the entries before the deadline, breaking ties earlier.*

memory	$\tau \in [0,1)$	$\tau \in [1,2)$	$\tau \in [2,3)$	$\tau \in [3,4)$
[0,2)	*9*	*5*	*4*	*6*
[2,4)	*19*	*17*	*13*	*15*
[4,8)	*24*	*20*	*15*	*21*

Suppose the estimated memory usage is 2.5 megabytes and the value schedule given by the user is $w_i([t_0, t_0+2)) = 19$, $w_i([t_0+2, t_0+3)) = 17$, *and* $w_i([t_0+3, t_0+\Delta]) = 0$. *Then the auctioneer examines the row 19, 17, 13, and chooses the scheduled completion time of 2 hours from now, at a price of 13 (i.e., the entry corresponding to* $[t_0+2, t_0+3)$*), since 17-13 = 4 is the maximum value - price among the corresponding entries.*

5.3 Theoretical Analysis

5.3.1 Simplicity for Users

Given that users are modeled with *threshold-reliability* beliefs I work with the following relaxed notion of strategyproofness:

Definition 19 (t-strategyproofness)**.** *An online mechanism with limited misreports (no early arrivals) is t-strategyproof if all users hold threshold-reliability beliefs, and no user has incentive to misreport her job description, value schedule, arrival time, or minimum tolerable reliability, regardless of the reports of other users.*

A t-strategyproof mechanism chooses the most favorable resource provider for a user with threshold-reliability beliefs when given the user's true parameters J_i, w_i, a_i, and γ_i, for any reports of the other users.

A formal definition of t-strategyproofness can be found in Definition 20.

Definition 20 (t-strategyproofness)**.** *Let $\theta = (\theta_i, \ldots, \theta_n)$, and let θ_{-i} denote $(\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n)$. Let $\theta_i = (J_i, w_i, a_i, \gamma_i)$ for all i. An online mechanism with limited misreports (no early arrivals) is t-strategyproof if all users hold threshold-reliability beliefs, and $\forall i$, $\forall \theta_i' \in \Theta$ and $\forall \theta_{-i} \in \Theta^{n-1}$ such that $f_i(\theta_i', \theta_{-i})$ is not null, and $J_i' \succ J_i$:*

$$w_i(\tau) - \phi_k^{a_i}(\hat{R}_k^{a_i}(J_i), \tau) \geq w_i(\tau') - \phi_{k'}^{a_i'}(\hat{R}_{k'}^{a_i'}(J_i'), \tau')$$

if resource provider k' has reliability at least γ, where $f_i(\theta) = C^k(a_i, J_i, \tau)$ and $f_i((J_i', w_i', a_i', \gamma_i'), \theta_{-i}) =$
$C^{k'}(a_i', J_i', \tau')$.

Given $\theta_i = (J_i, w_i, a_i, \gamma_i)$ for all users i, suppose $f_i(\theta) = C^k(a_i, J_i, \tau)$, i.e., the auctioneer selects the contract offered by resource provider k. Now, since a user has no value and makes no payments for an incomplete job, there are two cases:

$$v_i(\theta_i, C^k(a_i, J_i, \tau)) = \begin{cases} w_i(\tau), & \text{if job completed;} \\ 0, & \text{otherwise.} \end{cases}$$

$$p_i(\theta, C^k(a_i, J_i, \tau)) = \begin{cases} \phi_k^{a_i}(\hat{R}_k^{a_i}(J_i), \tau), & \text{if job completed;} \\ 0, & \text{otherwise.} \end{cases}$$

However, since the users hold *threshold-reliability beliefs*, they do not explicitly reason about the probability that their jobs will not complete. This leads to the concept of t-strategyproofness. Define $\tilde{p}^t(J, \gamma, \tau) = \min_{r(k) \geq \gamma} p_k^t(R_k^t(J), \tau)$ to be the price schedule given scheduled completion time τ that a user faces if she reports job description J, and reliability γ at time t.

Lemma 4. *$\tilde{p}^t(J, \gamma, \tau)$ is nondecreasing with respect to t, all $J' \succ J$, and all $\gamma' > \gamma$.*

Proof. Fix t. For all $J' \succ J$, for all τ, and for all resource providers k,

$$R_k^t(J') \geq R_k^t(J) \ \forall t$$

$$p_k^t(R_k^t(J'), \tau) \geq p_k^t(R_k^t(J), \tau)$$

Hence, $\tilde{p}^t(J', \gamma, \tau) \geq \tilde{p}^t(J, \gamma, \tau) \ \forall \tau, \ \forall J' \succ J$. For $\gamma' > \gamma$ the set of which the auctioneer takes the minimization is smaller, hence, $\tilde{p}^t(J, \gamma', \tau) \geq \tilde{p}^t(J, \gamma, \tau) \ \forall \tau, J, \ \forall \gamma' > \gamma$.

Finally, if $t' > t$, then for all τ, k, by admissibility,

$$p_k^{t'}(R_k^{t'}(J'), \tau) \geq p_k^t(R_k^t(J), \tau),$$

$$\text{and } \tilde{p}^{t'}(J, \gamma, \tau) \geq \tilde{p}^t(J, \gamma, \tau).$$

□

Theorem 10. *The auction protocol is t-strategyproof for jobs with bounded patience and users with limited misreports, uniform failure and threshold beliefs, if price table entries are admissible, resource predictors satisfy monotonicity for each resource provider.*

Proof. I show that reporting true $\theta = (J_i, v_i, a_i, \gamma_i)$ is a t-dominant strategy given that the contract chosen by Auction 1 succeeds.

First, by threshold beliefs, a user holds beliefs that the probability that a contract holds is the same for all contracts offered by a resource provider in Γ_i. Hence, she is indifferent across all contracts $C'(a_i, J_i', \tau)$ offered at time a_i by any resource provider with reliability at least γ_i, where C' is any contract for $J_i' \succ J_i$, completing by τ.

If a user reports $\hat{J}_i$ such that $\hat{J}_i \not\succeq J_i$, she has no value for the completed job. If the user reports $\hat{\gamma} < \gamma_i$ then the only way the user's utility is affected is if a resource provider $\hat{k}$ with reliability γ_i is selected, in which case, the user has no utility. By Lemma 4, for any given τ, the user cannot improve her utility by reporting $\hat{J}_i \succ J_i$, $\hat{\gamma}_i > \gamma_i$, or by delaying her arrival time. □

By monotonicity of the resource estimator and admissible prices, a user cannot receive a lower price by reporting a later arrival time or $J'_i \succ J_i$. The uniform failure belief assumption implies that for a specific job J_i, a user has no incentive to report $J'_i \succ J_i$ to achieve a higher probability of success on a specific resource provider. Threshold-reliability belief spares us from reasoning about the probability that a job will complete on the winning resource provider.

5.4 Discussion: Strategic Properties

First, I discuss properties that are essential for Theorem 1. The market infrastructure requires that the resource provider holds the result of a job until the scheduled completion time. This prevents a job from benefiting by overstating its patience and getting a lower price, while still getting the result of the computation early enough. Another important role of the market infrastructure is to ensure that the price quotes do not change based on the bid of a job. It is a role of the auctioneer, not of the resource providers, to select the best scheduled completion time for a user and perform the payoff-maximization decision. Resource providers define admissible prices but can update prices only in between receiving bids. Prices are set based on *scheduled*

completion times rather than based on the actual times in which jobs are performed. This is to prevent a resource provider from deliberately rescheduling the job to extract more revenue. Completion risk is carried by resource providers in that if they fail to complete a job by the scheduled completion time, they receive no payment and the user makes no payment (in this case, the user also receives no benefit). This is necessary to maintain strategyproofness in the value schedule model. Note that the estimate does not need to be accurate for the auction to remain truthful.

One possible form of useful manipulation that remains, e.g. when prices are super-additive in the size of resource allocations, is for a user to split a job into multiple smaller jobs. This kind of manipulation has been observed when very strategic players are present [53]. Having recognized this, super-additive prices can provide opportunities for significant improvement in efficiency, and can be incorporated by allowing price discrimination.

One additional concern that is relevant to the performance of the system is what happens when resource estimates are poor quality, as would be expected when a new user enters the system or when a new class of jobs are run. Here, I can augment the proposal to allow a user to *optionally* state explicit (computational) resource requirements. The auction would now need to be changed so that the resource estimate is adopted as a hard limit on the amount of resources provided to a job. This prevents a new manipulation in which a user underreports resource requirements of a job but is able to complete her work anyway for a lower price. This change was not needed for the current model because of the coupling of resource estimation with job descriptions. A user should exercise this "override option" and report explicit information

about resource requirements of her job when: (1) the resource provider overestimates resource requirements in which case a user can get a lower price by correcting the resource provider and still complete her job successfully, or (2) the resource provider underestimates resource requirements to such a degree that the job will not be completed successfully in which case a user will pay more but prefer to report correct resource requirements so that her job completes. Note that if a user believes that all resource providers have accurate estimators, then she has no incentive to override the estimates.

Chapter 6

Resource Estimation with Monotonicity Constraints in Open Systems

Abstract

The ability to accurately estimate resource requirements is crucial for resource providers to efficiently schedule jobs in open systems. I study the resource prediction problem that each resource provider faces in the decentralized auction framework presented in Chapter 5 through experiments using historical data from the Crimson Grid.

I explore learning techniques including linear regression, Naïve Bayes classification, Bayesian network classification, and conditional linear gaussian to predict the runtime of a job given attribute values available at submission time. I report the

relative accuracy of each technique given historical data from the Crimson Grid. Recall that the market infrastructure discussed in Chapter 5 requires that the learned models used by the resource predictors satisfy the *monotonicity* constraints to ensure strategyproofness for users. I discuss how the monotonicity requirement can be imposed or verified in combination with each learning technique.

The ability to accurately estimate resource requirements is crucial for resource providers to efficiently schedule jobs in open systems. Furthermore, accurate resource prediction enables a better assignment between jobs and resources, and improves the overall utility of an open system. I study the resource prediction problem that each resource provider faces in the decentralized auction framework presented in Chapter 5 through experiments using historical data from the Crimson Grid.

I explore learning techniques including linear regression, Naïve Bayes classification, Bayesian network classification, and conditional linear gaussian networks to predict the runtime of a job given attribute values available at submission time. I report the relative accuracy of each technique given historical data from the Crimson Grid collected between October 2006 and October 2007.

Recall that the market infrastructure discussed in Chapter 5 requires that the learned models used by the resource predictors satisfy the *monotonicity* constraints to ensure strategyproofness for users when price quotes are generated based on the predicted resource usage. I discuss how the monotonicity requirement can be imposed or verified in combination with each learning technique.

The goal of this chapter is not to propose an optimal resource estimation method, but rather to explore well-known learning techniques in conjunction with monotonicity requirements, demonstrate that these techniques achieve reasonable accuracy, and provide a stepping stone for resource providers to compete through building and deploying more advanced resource estimation techniques.

6.1 Problem definition

The resource prediction problem can be formulated as follows: given a job description that assigns value to a set of attributes, estimate the resource usage of a job using a function L satisfying monotonicity constraints. Throughout this chapter, the objective is to predict the *run time* of a given job, but the same approaches may be applied to predict the use of other computational resources (e.g., disk space, memory usage).

A job J is characterized by a vector of attribute and value pairs $< (a^1, v^1)$, ..., $(a^n, v^n) >$. There are two phases to the resource prediction problem:

Phase 1: Given a training set (subset of past observations), learn the parameters of the function L.

Phase 2: Given L, compute $L(J)$. The output of the function L can be a continuous value (regression), a class label (classification), or a distribution.

As more jobs are observed, *Phase 1* is repeated to improve the accuracy of L.

6.1.1 Error Metric

Let $r(J)$ be the observed run time and $L(J)$ be the predicted run time of job J. The *root mean squared error (RMSE)* metric is used when $L(J)$ predicts a continuous value. Let n be the total number of jobs in the prediction set.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(L(J) - r(J))^2} \tag{6.1}$$

For classification models, $L(J)$ returns a class label or a range of run times. Let $\overline{L(J)}$ be the mean value of the class $L(J)$, and $\overline{r(J)}$ be the mean value of the class that $r(J)$ belongs to. Two different metrics analogous to the RMSE metric can be defined.

$$RMSE^1_{dis} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\overline{L(J)} - \overline{r(J)})^2} \tag{6.2}$$

Defining the root squared mean error via equation 6.2 has the advantage that if the correct class is predicted, the error is 0.

$$RMSE^2_{dis} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\overline{L(J)} - r(J))^2} \tag{6.3}$$

On the other hand, using the definition from (6.3) incorporates the tradeoff between the size of the classes and the accuracy of prediction. I use equation 6.3 for the classification techniques.

If the output of L is a distribution for each job, then the distribution of root squared mean error can be computed via sampling:

Repeat M times:

1. For each of the n distributions, sample the distribution to obtain $\overline{L(J)}_k$.
2. Compute $RTSE_k = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\overline{L(J)}_k - r(J))^2}$.

Create a distribution using $RTSE_1, \ldots, RTSE_M$.

6.1.2 Monotonicity

Recall from Chapter 5 that there exists a partial order $\succ$ over the set of jobs that represents the users' preference ordering over attribute values. For example, if

attribute j is the input file size and $v^j = 10KByte$ and $v^{j'} = 20KByte$, and $v^k = v^{k'}$ for all $k \neq j$, then $J' \succ J$.

For many learning techniques, the probability distribution of $L(J)$ or the expectation and variance over $L(J)$ can be computed. First, when the probability distribution of $L(J)$ is available, monotonicity can be defined as follows:

Definition 21. *[Monotonicity: Distribution] A learning model L satisfies monotonicity if for all $J' \succ J$, if $F' \geq F$ everywhere, where F' is the cumulative distribution function on $L(J')$ and F is the cumulative distribution function on $L(J)$ (stochastic dominance).*

However, in the case where the entire probability distribution is not available, a relaxed version of monotonicity can be defined in terms of the expectation and variance of the prediction.

Definition 22 (Monotonicity: Mean-variance)**.** *A learning model L satisfies monotonicity if for all $J' \succ J$, $E[L(J')] \geq E[L(J)]$ and $Var[L(J')] \geq Var[L(J)]$.*

To ensure that the auction framework is strategyproof, higher expectation and higher variance in the prediction must translate to an increase in price. For example, if $J_1 \succ J_2$ and $L(J_1)$ and $L(J_2)$ have the same expectation, but $L(J_1)$ has a higher variance. Then the price quote given $(E[L(J_1)], Var[L(J_1)])$ must be no less than the price quote given $(E[L(J_2)], Var[L(J_2)])$. This is to prevent *preferred* jobs from receiving a lower price quote.

Example Attributes

The following are examples of attributes, values of which can be determined at submission time. These attributes can become input to the resource predictor.

1. User/ Organizational information
    - Domain: set of research institutions, research groups, or individual users
    - Hierarchical information can be revealed by e-mail or IP addresses.
    - In my data set, the username is available for every job. Research group name may also be available (e.g., seas/group/groupname/).
2. Geographical location
3. Virtual location (IP addresses)
4. Time of submission
5. Type of executable
    - Domain: { Unknown, Fortran, Rand, Matlab, R, etc. }
    - The executable type may be determined by the suffix of the command.
6. Memory size (Image Size in Condor), CPU speed
7. Internet connection speed
8. Operating system (All jobs in the Crimson Grid data set run on Linux, but may be applicable for other data sets).
9. Current load of the machine for which resource usage is predicted

10. Currency type used for payment: virtual organizations may create their own currency to implement sharing policies

11. Size of executable and input files

12. Program-specific parameters (flags)

13. Number of Arguments

6.2 Description of the Crimson Grid data set

I created a data set by collecting Condor job histories of jobs deployed on the Crimson Grid, a campus grid housed at Harvard University (see Chapter 1, Section 1.1.2), from October 2006 to October 2007. The data set includes approximately 100,000 jobs. Condor is a specialized workload management system for computationally intensive jobs developed at University of Wisconsin, Madison [68]. The Crimson Grid leverages the job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management provided by Condor. Users submit their jobs to Condor. A sample user input file for Condor is presented in Figure 6.1. Users can specify the name of the executable, the input files, machine requirements (e.g., memory, operating system, architecture), and where to store the output and error information. Then Condor places the jobs into a queue, chooses when and where to run the jobs, monitors their progress, and informs the user upon completion.

Figures 6.2 and 6.3 show the information available for each job in the data set. The bold fields are used to compute the value of the input attributes as shown in Figure 6.4. The underlined fields contain information needed to compute the output,

```
Executable = matlab
Requirements = Memory >= 32 && OpSys
  == "IRIX65" && Arch == "SGI"
Image Size = 28 MB
Input = test.data
Output = loop.out
Error = loop.error
Log = loop.log
```

Figure 6.1: Sample Condor user input file.

run time. Jobs with 1 or more restarts or more than one host are left out to ensure that *CompletionTime* - *JobStartTime* actually reflects the run time of a given job. Jobs that did not complete, i.e., ones with *ExitCode* not equal to 0, are not included in the data set. Figure 6.5 shows the decision process.

Figure 6.6 shows the run time distribution of the jobs in the data set. The X-axis is labeled by the midpoint of each bucket (the size of each bucket is 5,000 seconds). The histogram includes jobs with run time up to 50,000 seconds. Approximately 4% of jobs have length beyond 50,000 seconds ($\approx$ 14 hours), with the longest job having a run time of 4 days and 16 hours. More than 80% of the jobs are relatively short, taking less than 10,000 seconds.

Mean	12996.38 seconds
Standard Deviation	46645.15 seconds
Maximum	404204.00 seconds

Figure 6.7 shows the relationship between the size of the executable and the run time of a job. The fitted line shows that there is a positive correlation between the two

MyType = "Job"
TargetType = "Machine"
ClusterId = 1
QDate = 1144523333
Owner = "xxxxxx"
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
ExitStatus = 0
NumCkpts = 0
NumRestarts = 0
NumSystemHolds = 0
CommittedTime = 0
TotalSuspensions = 0
CumulativeSuspensionTime = 0
CondorVersion = "$CondorVersion: 6.7.18 Mar 1 2006 PRE-RELEASE-UWCS $"
CondorPlatform = "$CondorPlatform: I386-LINUX_RH9 $"
RootDir = "/"
Iwd = "/home/xxxxxx/stilt.revisemanuscript/Rsc1"
JobUniverse = 5
Cmd = "/home/xxxxxx/stilt.revisemanuscript/stilt.bat"
MinHosts = 1
MaxHosts = 1
WantRemoteSyscalls = FALSE
WantCheckpoint = FALSE
RemoteSpoolDir = "/local/condor/spool/cluster1.proc0.subproc0"
JobPrio = 0
User = "xxxxx@yyyy.harvard.edu"
NiceUser = FALSE
Environment = ""
JobNotification = 2
WantRemoteIO = TRUE
NotifyUser = "xxxxxx@yyy.harvard.edu"
UserLog = "/home/xxxxxx/stilt.revisemanuscript/Rsc1/condorstilt.log"
CoreSize = 0
KillSig = "SIGTERM"
Rank = 0.000000
In = "/dev/null"
TransferIn = FALSE
Out = "condorstilt.out"
StreamOut = FALSE
Err = "condorstilt.err"
StreamErr = FALSE
BufferSize = 524288
BufferBlockSize = 32768
ShouldTransferFiles = "IF_NEEDED"

Figure 6.2: Snapshot of data: Bold fields are used as inputs in the experiments.

```
WhenToTransferOutput = "ON_EXIT"
TransferFiles = "ONEXIT"
ExecutableSize = 1
DiskUsage = 1
Requirements = (Arch == "INTEL") && (OpSys == "LINUX") && (Disk >=
    DiskUsage) && ((Memory * 1024) >= ImageSize) && ((HasFileTransfer) ||
    (TARGET.FileSystemDomain == MY.FileSystemDomain))
FileSystemDomain = "yyyy.harvard.edu"
PeriodicHold = FALSE
PeriodicRelease = FALSE
PeriodicRemove = FALSE
OnExitHold = FALSE
OnExitRemove = TRUE
LeaveJobInQueue = FALSE
Arguments = ""
GlobalJobId = "node-10.deas.harvard.edu#1144523933#1.0"
ProcId = 0
WantMatchDiagnostics = TRUE
LastMatchTime = 1144523401
NumJobMatches = 1
OrigMaxHosts = 1
JobStartDate = 1144523460
JobCurrentStartDate = 1144523460
JobRunCount = 1
LastJobLeaseRenewal = 1144523489
RemoteSysCpu = 1.000000
RemoteUserCpu = 4.000000
ImageSize = 31124
ExitBySignal = FALSE
ExitCode = 0
BytesSent = 0.000000
BytesRecvd = 0.000000
JobStatus = 4
EnteredCurrentStatus = 1144523490
LastSuspensionTime = 0
RemoteWallClockTime = 30.000000
LastRemoteHost = "vm1@node-30.deas.harvard.edu"
LastClaimId = "<10.101.1.30:32794>#1144443296#27"
CurrentHosts = 0
CompletionDate = 1144523490
AutoClusterId = 1
AutoClusterAttrs =
    "JobUniverse,LastCheckpointPlatform,NumCkpts,Scheduler,DiskUsage,ImageSize,
    FileSystemDomain"
JobFinishedHookDone = 1144523490
```

Figure 6.3: Snapshot of data: Bold fields are used as inputs in the experiments.

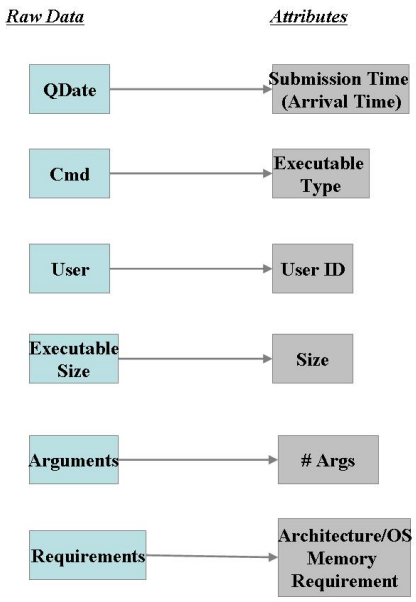


Figure 6.4: Input attributes

variables. The slope of the line is 1.646 with a standard error = 0.02, and the positive relationship is statistically significant.

Attributes available at submission time include user and research group information(*Owner* or *User*, time of submission (*QDate*), the command name (*Cmd*)(from which one can extract the type of executable), size of executable (*ExecutableSize*), and arguments (*Arguments*). A user can also provide an estimate of memory usage at submission time (*ImageSize*). If a user does not provide an estimate, *ImageSize* is set to *ExecutableSize*. However, after a job runs to completion, the *ImageSize* field is updated to the actual memory usage. The run time of a job can be computed using the *JobStartDate* and *CompletionDate* fields.

Time variables (e.g., *QDate*, *JobStartDate*, and *CompletionDate*) are measured in

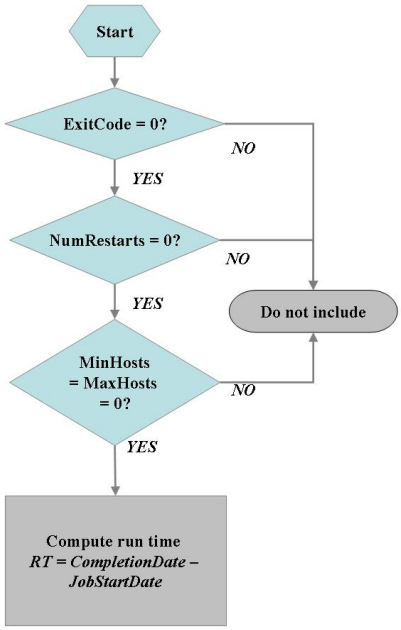


Figure 6.5: Computing the run time

seconds elapsed since the epoch (00:00:00 UTC, January 1, 1970). *ExecutableSize* and *ImageSize* are measured in KBytes.

6.3 Learning Techniques

I employ several machine learning techniques to learn the run time prediction function. I start with a *regression* method, where all variables are assumed to be continuous, then *classification* methods where all variables are assumed to be discrete, and finally, explore a combination method where both continuous and discrete variables can be used.

Linear regression produces a continuous model of run time, where Naïve Bayes

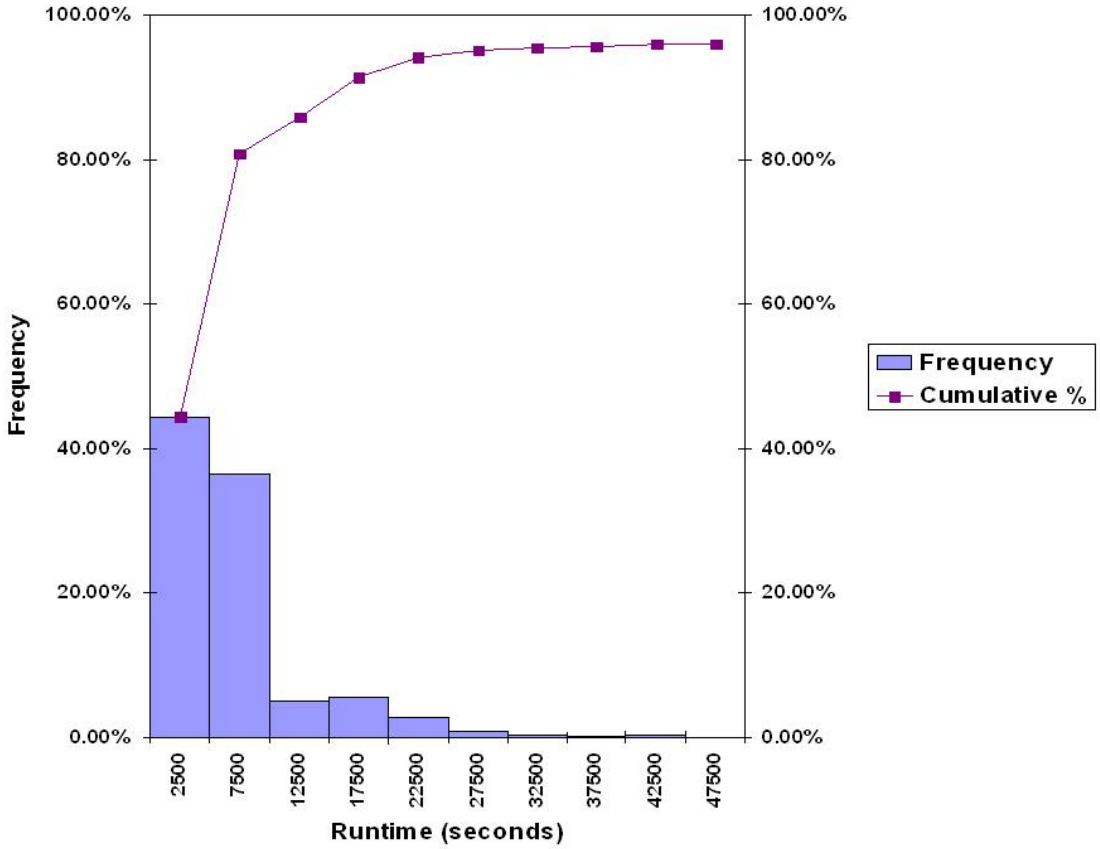


Figure 6.6: Distribution of the run time of Crimson Grid jobs

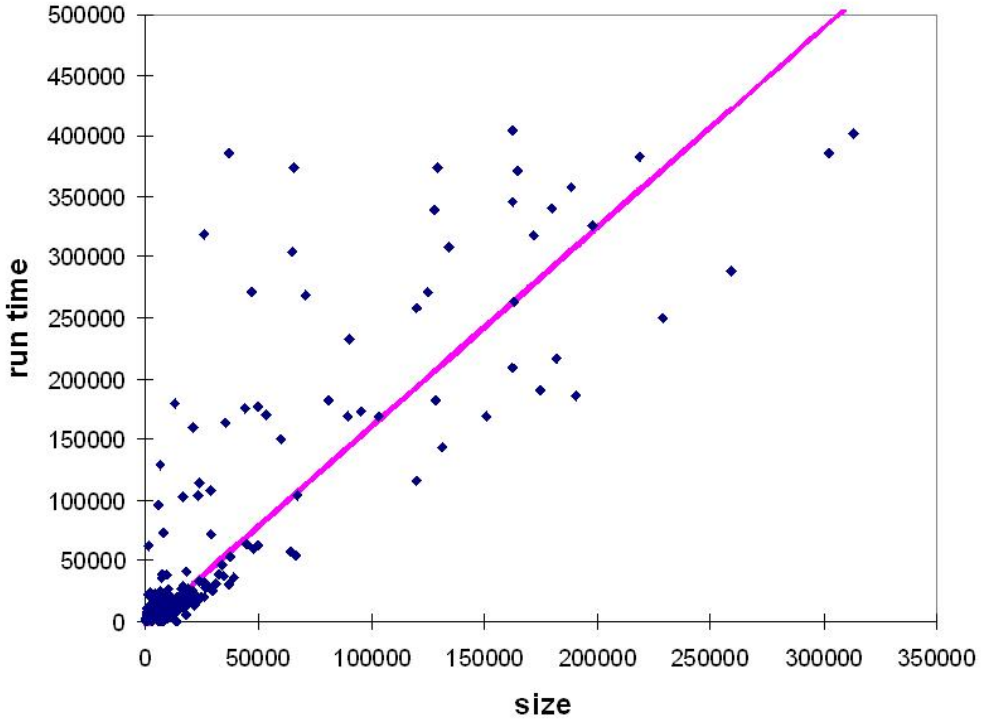


Figure 6.7: Executable size vs. Run time

classification and Bayesian network classification predict ranges (class labels) of run time using discretized attributes using *MAP* (Maximum A-Posteriori Estimator). Conditional linear gaussian takes both discrete and continuous attributes, and produces a Gaussian distribution over run time. Details on these techniques and other machine learning approaches can be found in [41, 48, 70, 75]. Attributes are written in uppercase, while specific attribute values are denoted by lowercase letters.

6.3.1 Linear Regression

Linear regression uses a *least-squares* function to model the relationship between one or more independent variables and the dependent variable. This function is written as a linear combination of one or more model parameters (regression coefficients). The learned model can be represented concisely as a vector of coefficients. The independent variables and the dependent variable are real-valued. For discrete attributes, indicator variables (dummy variables) with values $\in \{0, 1\}$ can be used.

The objective function minimizes the sum of squared residuals, r_i, the difference between the observed value and the value predicted by the model:

$$r_i = y_i - \sum_{j=1}^{m} X_{ij}\hat{\beta}_j,$$

where y is the (observed) vector of the dependent variable, X is the matrix of independent variables (input attributes), and $\hat{\beta}$ is the vector of estimated coefficients.

In matrix form, the vector of residuals can be written as $y - X\hat{\beta}$, thus the objective function becomes:

$$\min_{\hat{\beta}}(y - X\hat{\beta})'(y - X\hat{\beta}). \tag{6.4}$$

The solution to 6.4 is $\hat{\beta} = (X'X)^{-1}X'y$ [35].

6.3.2 Naïve Bayes Classification

Naïve Bayes is the simplest form of Bayesian networks, where the class node (the predicted variable) has no parents and is the only parent of all other nodes. Naïve Bayes classification is based on the independent attribute assumption, i.e., that each attribute is independent given the predicted attribute.

Although the independent attribute assumption is often violated in real world applications, many empirical comparisons [61, 40] have showed that Naïve Bayes performs as well as modern decision tree algorithms, e.g., C4.5 [63].

The main advantage of Naïve Bayes models is that only a small amount of training data is necessary to estimate the parameters (means and variances of the variables) needed for classification.

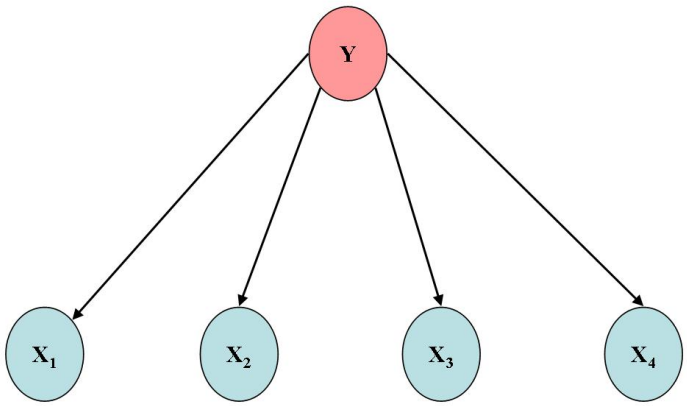


Figure 6.8: A Naïve Bayes Network

The algorithm for Naïve Bayes classification given the network in Figure 10 is as follows:

1. Estimate $Pr(Y = y)$ as fraction of records with $Y = y$

2. Estimate $Pr(X^i = x^i|Y = y)$ as fraction of $Y = y$ records that also have $X^i = x^i$.

3. To predict the Y value given observations of all the X^i values, compute

$$\hat{Y} = \arg\max_y Pr(Y = y|X^1 = x^1, \ldots, X^m = x^m)$$

$$\text{MAP(Maximum A} - \text{Posteriori Estimator)}$$

$$= \arg\max_y Pr(Y = y) \prod_{i=1}^{m} Pr(X^i = x^i|Y = y)$$

6.3.3 Bayesian Network Classification

Formally, a Bayesian network is a *directed acyclic graph (DAG)*, where each node represents an attribute, and each arc between nodes represents a probabilistic dependency, which is quantified via a conditional probability distribution. The joint distribution of node values can be written as a product of the distributions of each node and its parents:

$$Pr(X_1, X_2, \ldots, X_m) = \prod_{j=1}^{m} Pr(X_j|parents(X_j))$$

A Bayesian network can be used to compute the conditional probability of a node given values assigned to the other nodes. Thus, a Bayesian network can be used as a classifier that outputs the posterior probability distribution of the class node given the values assigned to other attributes. A major advantage of Bayesian network classifiers is that the network structure can represent the inter-relationships among the attributes. Learning the parameters for a given network structure that are optimal for

a set of complete data simply involves computing the empirical conditional frequencies from the data [19].

Suppose one wants to predict an output Y with values in $y \in \{y_1, \ldots, y_{n_Y}\}$ when there are m input attributes, $X^1, \ldots, X^m$.

The model is built as follows:

1. Divide data set into n_Y smaller data sets, $DS_1, \ldots, DS_{n_Y}$, where DS_j is the set of records for which $Y = y_j$.

2. For each DS_j:

   Learn Density Estimator M_j to model the input distribution among the records for which $Y = y_j$.

   M_j estimates $Pr(X^1, \ldots, X^m | Y = y_j)$.

3. Estimate $Pr(Y = y_j)$ as the fraction of records for which $Y = y_j$.

4. For a new input with attribute values $X^1 = x^1, \ldots, X^m = x^m$

$$\begin{aligned} \hat{Y} &= \arg\max_y Pr(Y = y | X^1 = x^1, \ldots, X^m = x^m) \\ &= \arg\max_y Pr(X^1 = x^1, \ldots, X^m = x^m | Y = y) Pr(Y = y) \end{aligned}$$

When a new input arrives with attributes $X^1 = x^1, \ldots, X^m = x^m$, the model predicts the value of Y that makes $Pr(Y = y | X^1 = x^1, \ldots, X^m = x^m)$ most likely (Maximum A-Posteriori Estimator).

6.3.4 Conditional Linear Gaussian

A conditional linear gaussian (CLG) network [41] is a variation of a Bayesian network where every discrete node has only discrete parents and every continuous node has Condition Linear Gaussian conditional probability tables. Conditional linear gaussian networks have the advantage that they allow for both discrete and continuous variables, that the learned model has a concise description that can easily be published, and that distributional information (mean and variance) for the output variable is available. Figure 6.9 is an example of a conditional linear gaussian network. The output variable Y is continuous, and has two continuous parents X_1, X_2, and one discrete parent D_1. The distribution of Y follows a normal distribution where its mean is a linear combination of the values of continuous parents and the variance depends on only its discrete parent.

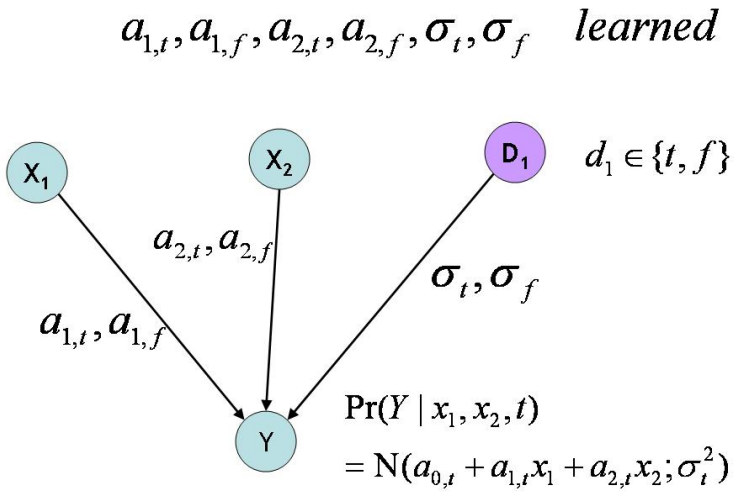


Figure 6.9: Conditional Linear Gaussian Network

6.4 Experimental Results

The following attributes available at submission time were used in the experiments.

1. User ID: user IDs are assigned to each distinct user based on the *Owner* field in the Crimson Grid data.

2. Size: *ExecutableSize* in KBytes

3. Type of Executable: the name of the executable file is available through the *Cmd* field. This variable is a *boolean* variable of whether the job is a mathematical and

4. Number of Arguments: number of arguments provided in the *Arguments* field.

The dependent variable or the class node is the *run time* of a job. Initially, the time of submission and requirements (architecture, operating system, and memory requirement) were also included as inputs, but were removed because of the lack of correlation between these variables and the run time.

I randomly partitioned the data set into two equal-sized sets and used one as the training set and the other as the prediction set.

The Naïve Bayes classification gave results that were slightly less accurate but comparable (2.94% higher root mean squared error) to the Bayesian network classification using a learned network structure.

Table 6.1 summarizes the root mean squared errors of each learning method.

Learning Method	RTSE
Linear Regression	18871
Naïve Bayes classification	12090
Bayesian network classification	11745
Conditional linear gaussian[1]	11258

Table 6.1: Root Mean Squared Errors

6.4.1 Linear Regression

A separate indicator variable was created for each distinct user ID. Another indicator variable was used for the *type of executable*. The *R-Squared* value was 0.8229, indicating that 82.29% of the variability in run time can be explained by the independent variables, *user ID*, *size*, *type of executable*, and *number of arguments*. The estimated coefficients and their standard errors are reported below (only the coefficients for the first three unique user IDs are presented).

Variable	Coefficient	Standard Error
Number of Arguments	-46.32	58.79
Type of Executable	-534.92	10145.40
Size	1.120	0.02
userId=0	-8245.33	10227.42
userId=1	22213.16	10612.18
userId=2	-4690.67	10252.53

The results suggest that *user ID* has a strong impact on *run time*, and that *size* has a strong positive correlation with *run time*. *Number of arguments* and *type of executable* both have a slightly negative effect on *run time*. The root mean squared error of the linear regression model given the prediction set with the above coefficients was 18871.

6.4.2 Naïve Bayes Classification

The Naïve Bayes network used for classification is presented in Figure 6.4.2.

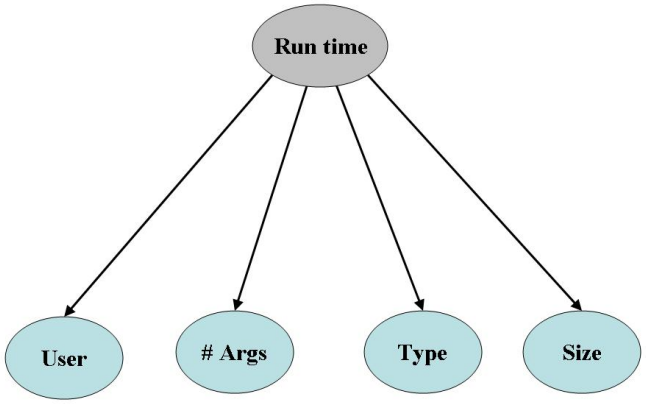


Figure 6.10: Naïve Bayes Network

The Naïve Bayes Classifier requires that all nodes are discrete. The *run time* node is discretized such that the range of each class is 5000 (same as the bucket size of the histogram presented in Figure 6.6). *Number of arguments* is divided up into 5 categories, *0-9*, *10-19*, *20-29*, *30-39*, and *at least 40*. *Size* is also discretized into classes of range 5000.

The conditional probability tables for when the run time is between 0 and 5000 ($Y = y_1$) is presented in Tables 6.2 through 6.5.

The root mean squared error defined in equation 6.3 of the Naïve Bayes model is 12090.

Number of Arguments(X^1)	$Pr(X^1 \mid Y = y_1)$
0-9	0.7811
10-19	0.0927
20-29	0.0635
30-39	0.0459
40+	0.0168

Table 6.2: Conditional Probability Table for *Number of arguments*

Executable Type (X^2)	$Pr(X^2 \mid Y = y_1)$
false	0.9223
true	0.0777

Table 6.3: Conditional Probability Table for *Executable Type*

6.4.3 Bayesian Network Classification

The same set of discretized data used in section 6.4.2 is used. The proposed network structure is presented in Figure 11. This structure is learned using the K2 algorithm [19], where all orderings of the 5 nodes were considered. Then the maximum likelihood parameters for the fully observed model are computed. The learned conditional probability table of *Number of Arguments* (up to *20-29*) given *User ID* for the first five user IDs is presented in Table 6.6 The root mean squared error defined in equation 6.3 of the classification model given the Bayesian network in Figure 11 is 11745.

6.4.4 Conditional Linear Gaussian

Run time and *size* are treated as continuous variables with gaussian conditional probability tables and all the other attributes as discrete variables.

Let u be the vector containing values of discrete parents, *User ID*, *Type*, and

Size(X^3)	$Pr(X^3 \mid Y = y_1)$
$[0, 5000)$	0.2657
$[5000, 10000)$	0.7282
$[10000, 15000)$	0.0062
$[15000, 20000)$	0
$[20000, 25000)$	0
$[25000, 30000)$	0
$[30000, 35000)$	0
$[35000, 40000)$	0
$[40000, 45000)$	0
45000+	0

Table 6.4: Conditional Probability Table for *Size*

User ID(X^4)	$Pr(X^4 \mid Y = y_1)$
1	0
2	0.0009
3	0.0018
4	0.0026
5	0.0035
6	0.0044
7	0.0053
8	0.0062
9	0.0071
10	0.0079

Table 6.5: Conditional Probability Table for *User ID* (only the first ten user IDs are presented)

User ID	$Pr(0-9 \mid \text{User ID})$	$Pr(10-19 \mid \text{User ID})$	$Pr(20-29 \mid \text{User ID})$
1	0.0189	0.0314	0.761
2	1	0	0
3	0.0272	0.0612	0.1905
4	0.996	0	0
5	0.9231	0.0769	0

Table 6.6: Conditional Probability Table for *Number of Arguments* given *user ID*

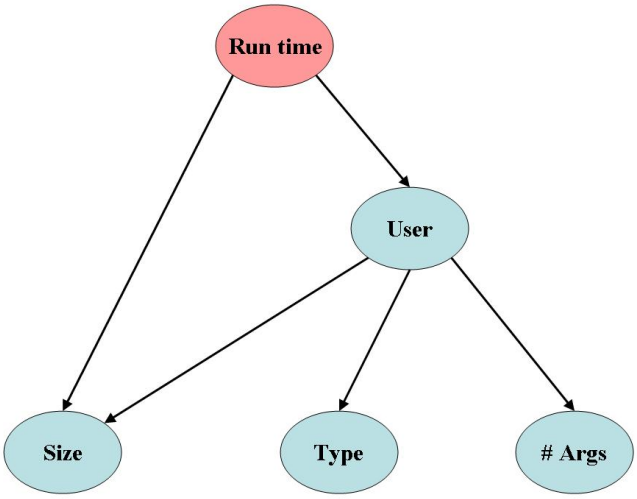


Figure 6.11: Bayesian Network Structure

Number of Arguments, and x be the value of the continuous parent, *Size*.

$$Pr(Y|u,x) = \mathcal{N}(a_{u,0} + a_{u,1}x, \sigma_u^2)$$

In contrast with the other models, the prediction is not a point value or class label, but rather a distribution. The distribution of the root mean squared error is computed as described in Section 6.1.1, where the number of trials, M, is 1000. The mean of the distribution of the root mean squared error is 11258, and the standard deviation is 1205.

6.5 Verifying and Imposing Monotonicity

For linear regression and conditional linear gaussian, monotonicity can be imposed by restricting that the coefficients of the attributes for which the partial order is defined are *nonnegative*. Similarly, for verification, one can check whether the published

coefficients are nonnegative. For conditional linear gaussian, the variance must also be checked if the partial order is defined on discrete variables.

For Naïve Bayes and Bayesian network classification, the parameters are estimated such that likelihood of the data is maximized (the negative of the log likelihood minimized). In other words, the parameter estimation problem is an optimization problem with the objective of maximizing likelihood of data. Hence, one can add monotonicity constraints to the parameter estimation problem and solve the optimization problem with respect to the added constraints. For verification, the conditional probability distributions of the output node can be compared for stochastic dominance.

Experimental results for linear regression and conditional linear gaussian show that the learned models satisfy monotonicity constraints. For instance, the coefficient of *size* estimated by linear regression is 1.120 with a standard error of 0.02, hence, one can conclude that the coefficient is nonnegative with more than 99.7% confidence. Similarly, for conditional linear gaussian, I check whether each learned coefficient $a_{u,1} \geq 0$ for each vector u of values of discrete parents, *User ID*, *Type*, and *Number of Arguments*. In this case, the variance does not depend on *size*.

Chapter 7

Methods for Optimal Monotonic Pricing in Open Systems

Abstract

Decentralized and autonomous control of resources and extensibility are desirable properties for efficient and sustainable resource allocation schemes in open computational grids. In a dynamic, distributed, and asynchronous setting, it is unrealistic to propose a particular selling mechanism that all resource owners should use. The auction framework supports the design principle of decentralized and autonomous control by allowing resource owners the flexibility of designing their own pricing algorithms and requiring only that the prices are *admissible*. This is achieved through resource owners defining and publishing their own price schedules called *price tables*. Resource owners can further replace and improve their pricing algorithms over time, leading to innovation.

In this chapter, I discuss several strategies with which a resource owner can define an admissible price schedule while maximizing total expected revenue in the context of the auction framework from Chapter 5. I present results from a simulation of the auction framework combined with price tables populated using these strategies. I compare the revenue properties of each strategy and formulate the offline optimal revenue problem and the offline optimal value problem as linear programs to use the solutions as benchmarks to evaluate the efficacy of the auction framework, given when these strategies are used, and estimate the cost of imposing the monotonic prices requirement.

In Chapter 5, I argued that decentralized and autonomous control and extensibility of resources is a desirable property of any efficient and sustainable resource allocation scheme in open computational grids. In a dynamic, distributed, and asynchronous setting, it is unrealistic to propose a particular selling mechanism that all resource owners should use. The auction framework supports the design principle of decentralized and autonomous control by allowing resource owners the flexibility of designing their own pricing algorithms and requiring only that the prices are *admissible*. This is achieved through resource owners defining and publishing their own price schedules, which I call *price tables*. Resource owners can further replace and improve their pricing algorithms over time to extract more revenue, leading to innovation.

In this chapter, I suggest several strategies with which a resource owner whose goal is to maximize total expected revenue can define an admissible price schedule.

Load: The load-based heuristic adjusts prices based on the current load, i.e., the number of time slots currently scheduled.

Elasticity: The elasticity-based heuristic adjusts prices based on the estimated price elasticity of demand.

Consensus: The consensus strategy is based on the Consensus algorithm [10], an Online Stochastic Optimization technique. Prices are adjusted based on votes given from job arrival scenarios by solving the offline optimization problem given each scenario.

In all of these strategies, the price for each time slot is only allowed to fall if the time slot is outside of the scheduling horizon to respect monotonicity.

I design a simulation of the auction framework with multiple resource providers where price tables are populated using the strategies described above. I separate the resource estimation module from the pricing module and assume that the resource estimates are given. I compare the revenue properties of each strategy and formulate the offline optimal revenue problem and the offline optimal value problem as linear programs to use the solutions as benchmarks when evaluating the efficacy of the auction framework given when these strategies are used. I estimate the cost of imposing the monotonic prices requirement by comparing the revenue and value achieved with admissible prices to the case where price cuts are allowed.

The analysis is based on the simple case where the only resource requirement a resource provider cares about is the run time of a job (job length), however, it can be generalized to the case with multi-dimensional resource requirements. I assume that each job requires a fixed number of *contiguous* slots, and that users have constant willingness to pay (denoted by v_i) for a completed job within their arrival-departure window.

7.1 Price tables

Recall that a resource provider maintains a *price table* visible to the market infrastructure, which is a function of a vector of resource requirements Q and the completion time τ. It can be thought of as a table of dimension $1 + dim(Q)$ where $dim(Q)$ is the dimension of the vector Q and the last dimension corresponds to period in which the job is to be completed. The price table is used by the market infrastructure to generate a resource provider's bid, given a reverse auction for a particular job in some

time period.

The ability of resource providers to fill the price table based on their own objectives is a key feature that provides autonomous control. A resource provider can update its price table entries to incorporate scheduling constraints, meet a target load level, or to improve its revenue. However, in order to maintain strategyproofness for end-users, the market infrastructure imposes that the price table entries are *admissible*.

Let $\phi_k^t(Q, \tau)$ denote the price table entries quoted by resource provider k in period t for resource requirement Q for the completion time τ. $\phi_k^t(Q, \tau)$ can be interpreted as: the price (quoted in time t) that resource provider k wishes to receive for completing a job with resource requirements Q, if it were scheduled to complete by time τ. Given a job, a *scheduled completion time* τ_k^* for each resource provider k is selected by the auctioneer, such that the payoff of the user is maximized.

Let $Q[m]$ denote the mth component of the vector Q, and let Δ be the user patience.

Definition 23 (admissible prices)**.** *Price table entries* $\phi_k^t(Q, \tau)$ *are admissible if both:*

$$\phi_k^t(Q'[m], \tau) \geq \phi_k^t(Q[m], \tau) \quad \forall Q'[m] > Q[m], \forall m, \forall \tau, \forall t \tag{7.1}$$

$$,\phi_k^{t'}(Q, \tau) \geq \phi_k^t(Q, \tau) \quad \forall t' > t, \forall t' \leq \tau \leq t + \Delta, \forall Q. \tag{7.2}$$

Prices are admissible if (1) the price table entries are nondecreasing in each component of the resource requirements, e.g., if Q consists of runtime (r) and disk space

(s)

$$\phi_k^t((r', s), \tau) \geq \phi_k^t((r, s), \tau) \ \forall r' > r$$

$$\phi_k^t((r, s'), \tau) \geq \phi_k^t((r, s), \tau) \ \forall s' > s$$

and (2) the price table entry for a given completion time within the user-patience window ($t + \Delta$) does not decrease over time. Note that this still allows a resource provider to decrease prices outside of the user-patience window.

7.1.1 Single-dimensional Price tables

For illustrative purposes, I consider here a single resource, the run time or *length* of a job. I further assume that the run time of a job is given (recall that the run time is computed by the resource estimation module of each resource provider) and is an integer at least 1. This can be represented via a two-dimensional price table indexed by job length and scheduled completion time, or a single dimensional price table indexed by scheduled completion time, where the price quote for a job of length l is computed by summing up the slot prices of l contiguous slots.

Alternative 1: Multi-dimensional price table

A price table is a two-dimensional table of *number of slots* (or job length) $\times$ *time period*. Each entry $\phi(l, \tau)$ defines a price for l slots with scheduled completion time τ. Note that $\phi(l', \tau) \geq \phi(l, \tau)$ for all $l' > l$, for all τ by admissibility, and that $\phi(l, \tau)$ is undefined (or ∞) for $\tau < t_0 + l$ where t_0 is the current time. Moreover, I have the restriction that the scheduling must be feasible given the set of jobs already won,

e.g., if the number of slots already scheduled prior to τ is $n^s(\tau)$, then the price table entries are undefined for all (l, τ) such that $l + n_s(\tau) < \tau - t_0$.

Alternative 2: Additive prices (single-dimensional price table)

The resource can maintain a single-dimensional table, as in the case of unit-length jobs. Each entry $\phi(\tau)$ represents the price of a given slot τ. Let the length of job i be l_i, user patience be Δ_i, the current time be t_0. Then the price quote from resource j is

$$\rho_j = min_{t \in [t_0+l_i, t_0+\Delta_i]} \sum_{\tau=t-l_i}^{\tau=t} \phi_j(\tau). \tag{7.3}$$

An additional restriction must be placed: if the number of slots already scheduled prior to τ is $n^s(\tau)$, $l + n_s(\tau) < \tau - t_0$. However, this is not sufficient to ensure scheduling feasibility in this model.

Incorporating capacity constraints

To ensure scheduling feasibility, the resource providers can publish capacity constraints in addition to the price schedule. The auctioneer does not obtain price quotes from resource providers whose capacity constraints would be violated if job is accepted.

The scheduled completion time for a job j becomes:

$$\tau^* = \arg\max_{\tau} w_j(\tau) - \rho(\tau)$$

subject to Capacity Constraints

instead of

$$\tau^* = \arg\max_{\tau} w_j(\tau) - \rho(\tau)$$

where $w_j(\tau)$ is the maximum willingness to pay for job j if the completion time is τ, and $\rho(\tau)$ is the price quote based on completion time τ.

7.2 Offline Omniscient Optimal Revenue Problem

The offline omniscient optimal revenue problem solves for the optimal price schedules of a resource given that it knows the exact sequence of future job arrivals within a fixed scheduling horizon. The objective is to maximize total revenue subject to capacity constraints and *admissible prices* constraints.

I study the problem in the case of a single resource with a single-dimensional price table, where jobs have constant value(v_i) within the arrival-departure window. Let $J = \{(v_1, l_1, a_1, d_1), \ldots, (v_n, l_n, a_n, d_n)\}$ be the set of jobs, and let Δ be the scheduling horizon. Then the total revenue received is:

$$\sum_i \min_{t \in [a_i + l_i, d_i]} \sum_{\tau=t-l_i}^{\tau=t} \phi^{a_i}(\tau) \tag{7.4}$$

where $\phi^t(\tau)$ denotes the price table entry for slot τ at time t.

First, define $\rho(a, d, l) = \min_{t \in [a+l, d]} \sum_{\tau=t-l}^{\tau=t} \phi^a(\tau)$. $\rho(a, d, l)$ is the price quote for a job of length l with arrival-departure window $[a, d]$. Let $\rho_i = \rho(a_i, d_i, l_i)$ for each job i.

Also define

$$x_i(s,t) = \begin{cases} 1, & \text{job i is scheduled in the interval } [s,t] \ (t = s + l_i); \\ 0, & \text{otherwise.} \end{cases} \tag{7.5}$$

and let $y_i = \sum_{(s,t): a_i \le s < t \le d_i} x_i(s,t))$ be the indicator variable for whether job i is scheduled within its arrival-departure window. Let t_0 be the current time period. Then the offline optimal revenue problem becomes:

Maximize $\sum_i \rho_i y_i$ subject to:

$$\rho_i y_i \leq v_i \ \forall i \text{ (schedule if value is at least price)}$$

$$\sum_i x_i(s,t) \leq 1 \ \forall [s,t] \subseteq [t_0, t_0+\Delta] \text{ (capacity constraint)}$$

$$\phi^{t'}(\tau) \geq \phi^t \ \forall t' > t \in [t_0, t_0+\Delta], \tau \in [t, t_0+\Delta]$$

This can be written as the following linear program:

$$\text{Maximize } \sum_i z_i \text{ subject to}$$

Constraints:

$$\phi^t(\tau) \leq \phi^{t+1}(\tau) \ \forall t \in [t0, t0+\Delta], \ \forall \tau \in [t, t0+\Delta] \tag{7.6}$$

$$\sum_i x_i(s,t) \leq 1 \forall [s,t] \subseteq [t0, t0+\Delta] \qquad \forall i: \tag{7.7}$$

$$z_i \leq \rho_i \tag{7.8}$$

$$z_i \leq M_i y_i \tag{7.9}$$

$$y_i \leq \sum_{s=a_i}^{s=d_i-l_i} x_i(s, s+l_i) \tag{7.10}$$

$$v_i - \rho_i \geq -N_i(1-y_i) \tag{7.11}$$

$$\sum_{[s,t]\subseteq[a_i,d_i]} x_i(s,t) \leq 1 \tag{7.12}$$

$$\rho_i \leq \sum_{\tau=t}^{\tau=t+l_i} \phi^{a_i}(\tau) \forall t \in [a_i, d_i - l_i] \tag{7.13}$$

with constants M_i and N_i (maximum possible value of ρ_i and v_i)

$$M_i \geq \max \rho_i$$

$$N_i \geq \max v_i$$

Bounds:

$$\phi^t(\tau) \geq 0 \ \forall t \in [t0, t0 + \Delta], \ \forall \tau \in [t, t0 + \Delta] \tag{7.14}$$

$$\forall i : \tag{7.15}$$

$$y_i \in \{0, 1\} \tag{7.16}$$

$$x_i(s, t) \in \{0, 1\} \ \forall [s, t] \subseteq [a_i, d_i] \tag{7.17}$$

$$z_i \geq 0 \tag{7.18}$$

$$\phi^t(\tau) \geq 0 \tag{7.19}$$

$$\rho_i \geq 0 \tag{7.20}$$

I prove a lemma showing that there is a single optimal price schedule that yields the same revenue as the optimal general formulation with one price schedule per time period. This allows us to solve a simpler linear program where there is only one price variable per time slot.

Lemma 5. *There exists a single price schedule that yields the same revenue as the optimal general formulation with admissible prices.*

Proof. Note that a set of price schedules determines an allocation. Let J be the set of jobs assigned to a resource provider given the optimal general formulation, ordered in increasing order of scheduled time. Note that scheduling all jobs in S must be feasible, i.e., there exists (t_j^S, t_j^E) for each j such that

$$a_j \leq t_j^S < t_j^S + l_j = t_j^E \leq d_j$$

$$\text{and } t_j^E < t_{j+1}^S \ \forall j$$

Consider a price schedule where the prices for slots within the interval $[t_j^S, t_j^E]$ are exactly the quoted prices of the slots for job j. Hence, there exists an equivalent formulation where the scheduling rule is "schedule-where-price-quoted" that yields the same revenue given the resource wins the same set of jobs.

Now fix a job $j \in J$, and suppose there exists a job j' that the resource provider could have scheduled with earlier arrival time and higher valuation than j such that $J \backslash \{j\} \cup \{j'\}$ is still feasible. Then the resource provider could have increased revenue by accepting j' in lieu of j. This contradicts the optimality of the J.

Given the fixed price schedule, suppose there exists a job $j \in J$ such that $\rho(a_j, d_j, l_j) < \rho(t_j^S, t_j^E, l_j)$, i.e., job j faces a lower price given the single price schedule. Then there exists $i \in J$ such that for some alternate start time for job j s_j, $[s_j, s_j + l_j] \subseteq [t_i^S, t_i^E]$, and $\rho(s_j, s_j + l_j, l_j) < \rho(t_j^S, t_j^E, l_j)$. Then i must have arrived earlier than j, since otherwise, the prices faced by i cannot be lower than that faced by j by monotonicity. Since i arrived earlier and is scheduled in $[t_i^S, t_i^E]$, those slots are not available when j arrives, i.e., capacity constraints prevent j from receiving a lower price quote.

Hence, the single price schedule generates the same revenue as the optimal general formulation. □

The offline omniscient value problem can be formulated similarly.

7.3 Description of Pricing Strategies

In this section, I describe each pricing strategy in detail. Consider the single-dimensional price table model. For all three strategies, I divide the slots within the scheduling horizon Δ into three regions: urgent(U), medium(M), not urgent(NU), and start with a single price per slot for each region, by setting the initial price vector p^0 such that $p^0_U \geq p^0_M \geq p^0_{NU}$, $p^0_r \in [0, V]$, $r \in [U, M, NU]$.

7.3.1 Heuristic based on load

This simple heuristic helps resource owners maintain a target load level. If the resource is facing high demand, i.e., load is heavy, then the base prices for each region is increased, and vice versa.

1. Let the time between price table updates be T, and fix $\delta p > 0$ for each region.
2. At the end of each period, for each region, record the number of slots that were scheduled in that period. This is the load in each period. Suppose load $\in$ {heavy, medium, light}.
3. For each region $r \in \{U, M, NU\}$, at the end of period t
   - if (load == heavy) then $p^{t+1}_r = p^t_r + \delta p_r$
   - if (load == medium) then $p^{t+1}_r = p^t_r$
   - if (load == light) then $p^{t+1}_\tau = p^t_\tau - \delta p$ for slots τ beyond the scheduling horizon, and $p^{t+1}_\tau = p^t_\tau$ for slots within the scheduling horizon.

Example 11. *Suppose* $\Delta = 6$. *At every time step* t, $U = \{t+1, t+2\}$, $M = \{t+3, t+4\}$, *and* $NU = \{t+5, t+6\}$. *Let* $\delta p = 1$.

We start at time $t = 0$, *with* $p_U^0 = 5$, $p_M^0 = 4$, *and* $p_{NU}^0 = 3$. *For each job scheduled at* t, *with scheduled completion time* τ, *record if* τ *falls into U, M, or NU when the job is submitted. Suppose after the observation period* $T = 2$, *we find that the load in U is heavy, and the load in NU is light. Then we update* $p_U = 6$, *and* p_{NU} *for the slots outside the scheduling interval to be* 2.

current time	1	2	3	4	5	6	7	8
t = 0	5	5	4	4	3	3		
t = 1		5	5	4	4	3	3	
t = 2			6	6	4	4	3	2

Table 7.1: Example 11: updating the price table using the load heuristic.

7.3.2 Heuristic based on elasticity

The elasticity-based heuristic adjusts prices based on the estimated price elasticity of demand. If demand is inelastic, base prices for each region is increased to extract more revenue. The basic idea behind the elasticity estimation is to treat demand for a job of length l as demand for l separate slots.

1. Let the time between updates in prices be T. Let the price vector in the interval $[kT, (k+1)T)$ be p^k.

2. Fix $\Delta p > 0$ (price increments), and $\delta p > 0$ (perturbation).

3. While time $\in [k\tau, (k+1)\tau)$:

- For each job that arrives, first determine which slots the default price quote is based on (compute the scheduled completion time). Let S be the set of slots selected.
- For each slot $\tau \in S$, let r be the region corresponding to slot τ. Flip a coin, and let $\pi(\tau) = p^k(\tau)$ if H, and $\pi(\tau) = p^k(\tau) + \delta p$ if T. Offer $\sum_\tau \pi(\tau)$, and increment the number of times that $p^k(\tau)$ or $p^k(\tau) + \delta p$ was offered for each region.
- If $\sum_\tau \pi(\tau)$ is accepted by the job, i.e., if the resource wins the job, increment the number of times that $p^k(\tau)$ or $p^k(\tau) + \delta p$ was accepted for each region.
- For each region r: let p_r^k denote the price vector for region r and W be the event that the resource wins the job.
  - Estimate $Pr(\text{W } | p_r^k)$ and $Pr(\text{W } | p_r^k + \delta p)$.
  - Compute
$$\eta_r = \frac{p_r^k}{Pr(\text{W } | p_r^k)} \times \frac{Pr(\text{W } | p_r^k + \delta p) - Pr(\text{W } | p_r^k)}{\delta p}$$
  - If $|\eta_r| < 1$, $p_r^{k+1} = p_r^k + \Delta p$.
  - If $|\eta_r| > 1$, $p_r^{k+1} = p_r^k - \Delta p$ for slots outside the scheduling horizon.

Note that δp must be small enough so that $p_U^k \geq \delta p + p_M^k \geq 2\delta p + p_{NU}^k$.

Example 12. *Let $\Delta p = 1$. Suppose 5 jobs of length 2 arrive in $[0, T]$, and all jobs can be scheduled in the urgent region. 5 of them are offered a price of $p_U^0 = 4$, which all 5 accept, and the other 5 are offered 5, among which 4 accept. Then*

$$\eta_U = \frac{4}{1} \times \frac{1 - 0.8}{1} = 0.8 < 1.$$

For the next observation period, the price for the urgent region p_U^1 is increased to 5.

7.3.3 Online stochastic optimization: Consensus algorithm

The idea behind the consensus algorithm is to sample a set of scenarios, solve the offline optimization problem given each scenario, then keep a vote as to which strategy is optimal. Prices are updated every T periods via the consensus algorithm.

- Scenario s: Sequence of job arrivals (facing this machine) within scheduling horizon ($[t_0, t_0 + \Delta]$), where each job associated with a value and length.
- Strategy space: for each region r, $\Delta p \in \{-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2.0\}$
- States consist of jobs already won, current price table entries

Consensus Algorithm:

1. Generate n scenarios.
2. For each scenario:
   (a) Solve the offline optimization problem given the set of jobs already won, and current price table entries.
   (b) Compare the slot prices computed for time step $t_1 > t_0$ to the current price table entries. Let $d(\tau) = p^{t_1}(\tau) - p^{t_0}(\tau)$ (rounded to the nearest multiple of 0.5).
   (c) For each region, if $\tau \in r$, increment the votes for bucket corresponding to $d(\tau)$.

3. For each region, select Δp to be the value of the bucket that received the most votes. Let $p_r = p_r + \Delta p$, and update price table entries.

7.4 Empirical Study

In this section, I compare the relative performances of the load-based, elasticity-based, and consensus pricing strategies, first by simulating job arrival based on the Poisson arrival distribution, then by using a job arrival distribution modeled after the Crimson Grid data described in Chapter 5 Section 3.

7.4.1 Design of the simulation

The load-based, elasticity-based, and consensus pricing strategies are implemented and tested in conjunction with the auction framework. Each resource provider maintains its own price table, which is populated with a base price vector. Then after every 5 time steps, the price table is updated via the pricing strategy elected by the resource provider. For simplicity, I assume that a resource provider does not switch pricing strategies throughout each run of the experiment. However, all 28 distinct combinations of 6 resource providers electing one of three strategies are explored.

Each job has the following parameters:

- arrival time
- patience
- willingness-to-pay or value of a job given that it is completed by arrival time + patience

- reliability threshold
- length of the job estimated by the resource estimation module

Job arrival distributions are obtained by simulating Poisson arrival and by using a distribution modeled after the Crimson Grid data described in Chapter 5 Section 3.

Poisson Arrival Distribution

Given a constant λ, the number of arrivals in each period is sampled as follows:

Initialize $L \leftarrow e^{-\lambda}$, $n \leftarrow 0$, and $P \leftarrow 1$.

While $P \geq L$

$n \leftarrow n + 1$

Generate a uniform random number $r \in [0, 1]$

$P \leftarrow rP$

Return n

λ is the expected number of arrivals within each period. In each period, n new jobs are generated and added to the job queue of the Auctioneer. The length of each job is a uniform random number between 1 and 5.

Crimson Grid Arrival Distribution

Each (arrival time, length) pair is sampled from the Crimson Grid data distribution. Given monthly job submission data, beginning of each month corresponds to

time 0 and the end of each month corresponds to time 300. Each day is divided up into roughly 10 intervals. Jobs that run longer than a day are not included in the experiment. The arrival time of each job is mapped into [0, 300] according to which interval the *job submission time* falls into, and the length of a job is mapped into [0, 10], where the length is the number of intervals between *job start time* and *job completion time* (inclusive).

The default values for the remaining parameters are set as follows for both arrival distributions:

- patience: length $\times(1 + r)$ where r is a uniform random number in $[0, 1]$
- willingness-to-pay: length$^2 \times r$ where r is a uniform random number in $[0, 1]$
- reliability threshold: $0.8 + 0.2r$ where r is a uniform random number in $[0, 1]$

7.4.2 Performance of the strategies

I present the revenue and value properties of each combination of the three strategies. First, I analyze a two-player game, in the case of only two resource providers, where the payoff is (1) revenue earned and (2) value served (total value of the jobs scheduled on each resource provider) under both the Poisson arrival distribution and the distribution modeled after the Crimson Grid data. Second, I present a more extensive heuristic game analysis in the case where 6 resource providers are competing for jobs for both revenue and value. In each game, the payoffs are normalized by dividing the revenue and value by the revenue of an omniscient offline revenue-optimal resource provider and the value of an omniscient offline value-optimal resource provider, re-

spectively. Finally, I estimate the cost of imposing monotonic prices constraints by comparing the revenue earned and value served when the constraints are present to those when the constraints are removed. Unless otherwise noted, the results are generated using the Crimson Grid distribution.

Two-player game

Poisson distribution

The following tables are generated using a Poisson arrival distribution with $\lambda = 2$. Payoffs are normalized with respect to the revenue of an omniscient offline revenue-optimal resource provider or the value of an omniscient offline value-optimal resource provider, where the optimal payoff is the average revenue earned by 2 omniscient offline revenue-optimal resource providers or the average value served by 2 omniscient offline value-optimal resource providers, respectively.

	L	E	C
L	(0.377, 0.377)	(0.382, 0.515)	(0.361, **0.543**)
E	(0.515, 0.382)	(0.510, 0.510)	(0.506, **0.543**)
C	(**0.543**, 0.361)	(**0.543**, 0.506)	(**0.526**, **0.526**)

Table 7.2: Two-player game (Poisson): Payoffs are normalized revenue earned over the duration of the experiment. Best responses are italicized.

	L	E	C
L	(0.513, 0.513)	(0.524, 0.625)	(0.498, **0.682**)
E	(0.625, 0.524)	(0.637, 0.637)	(0.632, **0.675**)
C	(**0.682**, 0.498)	(**0.675**, 0.632)	(**0.661**, **0.661**)

Table 7.3: Two-player game (Poisson): Payoffs are normalized value served over the duration of the experiment. Best responses are italicized.

Crimson Grid Distribution

	L	E	C
L	(0.401, 0.401)	(0.418, 0.429)	(0.383, **0.513**)
E	(0.429, 0.418)	(0.444, 0.444)	(0.435, **0.513**)
C	(**0.515**, 0.383)	(**0.513**, 0.435)	(**0.496**, **0.496**)

Table 7.4: Two-player game (Crimson Grid): Payoffs are normalized revenue earned over the duration of the experiment. Best responses are italicized.

	L	E	C
L	(0.778, 0.778)	(0.834, 0.843)	(0.791, **0.917**)
E	(0.843, 0.834)	(0.854, 0.854)	(0.829, **0.932**))
C	(**0.917**, 0.791)	(**0.932**, 0.829)	(**0.891**, **0.891**)

Table 7.5: Two-player game (Crimson Grid): Payoffs are normalized value served over the duration of the experiment. Best responses are italicized.

The payoffs and best-responses in a 2-player symmetric game with strategy space {load-based(L), elasticity-based (E), consensus (C)} are presented in tables 7.2 through 7.5. In all games, the unique Nash equilibrium is (C, C). Moreover, (C, C) constitutes a dominant strategy equilibrium of the game.

The difference in normalized revenue earned across strategy pairs is smaller under the Crimson Grid distribution. A resource provider playing strategy C earns a relatively higher payoff when facing the Poisson arrival distribution. However, the relative value served is higher for all strategy pairs when facing the Crimson Grid distribution.

Heuristic games analysis [36]

I present a game theoretic analysis of a six-player symmetric game. There are $\binom{6+3-1}{6} = 28$ distinct combinations of six players choosing from three pricing strategies. Table 2 summarizes the payoff (average total revenue) of each player. Each entry denotes the mean or the standard error of the payoff of a player who played a strategy

given the strategy profile for all 6 players. Figure 1 shows the revenue gain by a player if the player switches to another strategy. As noted in Section 4.2.1, choosing the Consensus strategy yields the highest payoff regardless of the other players' strategy profiles.

	L	E	C	Total
(0 , 0 , 6)	0.000 (0.000)	0.000 (0.000)	0.492 (0.006)	2.952
(0 , 1 , 5)	0.000 (0.000)	0.031 (0.005)	0.579 (0.007)	2.926
(0 , 2 , 4)	0.000 (0.000)	0.050 (0.005)	0.705 (0.009)	2.919
(0 , 3 , 3)	0.000 (0.000)	0.106 (0.005)	0.859 (0.010)	2.895
(0 , 4 , 2)	0.000 (0.000)	0.183 (0.006)	1.058 (0.010)	2.849
(0 , 5 , 1)	0.000 (0.000)	0.302 (0.006)	1.251 (0.014)	2.761
(0 , 6 , 0)	0.000 (0.000)	0.440 (0.006)	0.000 (0.000)	2.639
(1 , 0 , 5)	0.019 (0.005)	0.000 (0.000)	0.624 (0.007)	3.139
(1 , 1 , 4)	0.040 (0.009)	0.063 (0.019)	0.611 (0.019)	2.548
(1 , 2 , 3)	0.116 (0.010)	0.064 (0.004)	0.859 (0.010)	2.821
(1 , 3 , 2)	0.229 (0.013)	0.119 (0.004)	1.058 (0.010)	2.702
(1 , 4 , 1)	0.422 (0.015)	0.205 (0.005)	1.251 (0.014)	2.492
(1 , 5 , 0)	0.648 (0.014)	0.315 (0.005)	0.000 (0.000)	2.224
(2 , 0 , 4)	0.031 (0.005)	0.000 (0.000)	0.705 (0.009)	2.880
(2 , 1 , 3)	0.067 (0.006)	0.097 (0.005)	0.859 (0.010)	2.809
(2 , 2 , 2)	0.132 (0.009)	0.151 (0.004)	1.058 (0.010)	2.681
(2 , 3 , 1)	0.231 (0.009)	0.251 (0.005)	1.251 (0.014)	2.466
(2 , 4 , 0)	0.445 (0.013)	0.295 (0.005)	0.000 (0.000)	2.071
(3 , 0 , 3)	0.065 (0.005)	0.000 (0.000)	0.859 (0.010)	2.773
(3 , 1 , 2)	0.109 (0.007)	0.201 (0.005)	1.058 (0.010)	2.643
(3 , 2 , 1)	0.192 (0.007)	0.286 (0.006)	1.251 (0.014)	2.398
(3 , 3 , 0)	0.325 (0.010)	0.348 (0.006)	0.000 (0.000)	2.019
(4 , 0 , 2)	0.113 (0.006)	0.000 (0.000)	1.058 (0.010)	2.567
(4 , 1 , 1)	0.176 (0.006)	0.364 (0.006)	1.251 (0.014)	2.320
(4 , 2 , 0)	0.272 (0.007)	0.431 (0.007)	0.000 (0.000)	1.950
(5 , 0 , 1)	0.186 (0.006)	0.000 (0.000)	1.251 (0.014)	2.183
(5 , 1 , 0)	0.260 (0.006)	0.523 (0.007)	0.000 (0.000)	1.824
(6 , 0 , 0)	0.271 (0.006)	0.000 (0.000)	0.000 (0.000)	1.627

Table 7.6: Two-player game: Payoffs are normalized revenue served over the duration of the experiment.

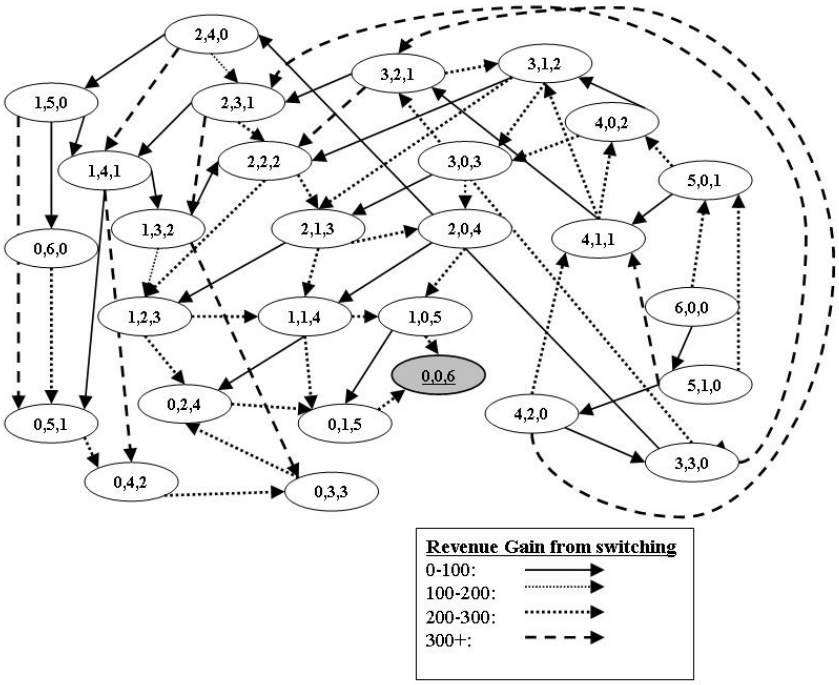


Figure 7.1: Revenue gain by a player if the player switches to another strategy. Arrows show direction of improvement.

	L	E	C	Total
(0 , 0 , 6)	0.000 (0.000)	0.000 (0.000)	0.922 (0.002)	5.531
(0 , 1 , 5)	0.000 (0.000)	0.084 (0.007)	1.091 (0.002)	5.541
(0 , 2 , 4)	0.000 (0.000)	0.125 (0.007)	1.328 (0.003)	5.562
(0 , 3 , 3)	0.000 (0.000)	0.247 (0.009)	1.627 (0.003)	5.623
(0 , 4 , 2)	0.000 (0.000)	0.440 (0.008)	1.966 (0.007)	5.691
(0 , 5 , 1)	0.000 (0.000)	0.735 (0.007)	2.117 (0.013)	5.790
(0 , 6 , 0)	0.000 (0.000)	0.905 (0.004)	0.000 (0.000)	5.432
(1 , 0 , 5)	0.042 (0.007)	0.000 (0.000)	1.091 (0.002)	5.499
(1 , 1 , 4)	0.089 (0.015)	0.072 (0.004)	1.328 (0.003)	5.473
(1 , 2 , 3)	0.262 (0.037)	0.109 (0.003)	1.627 (0.003)	5.361
(1 , 3 , 2)	0.534 (0.066)	0.227 (0.005)	1.999 (0.007)	5.215
(1 , 4 , 1)	0.983 (0.074)	0.427 (0.004)	2.317 (0.013)	5.007
(1 , 5 , 0)	1.502 (0.071)	0.663 (0.003)	0.000 (0.000)	4.818
(2 , 0 , 4)	0.062 (0.007)	0.000 (0.000)	1.328 (0.003)	5.437
(2 , 1 , 3)	0.145 (0.014)	0.159 (0.003)	1.627 (0.003)	5.332
(2 , 2 , 2)	0.306 (0.025)	0.263 (0.001)	1.999 (0.007)	5.137
(2 , 3 , 1)	0.563 (0.027)	0.473 (0.003)	2.317 (0.013)	4.863
(2 , 4 , 0)	1.020 (0.026)	0.560 (0.002)	0.000 (0.000)	4.280
(3 , 0 , 3)	0.124 (0.009)	0.000 (0.000)	1.627 (0.003)	5.253
(3 , 1 , 2)	0.236 (0.013)	0.335 (0.002)	1.999 (0.007)	5.042
(3 , 2 , 1)	0.440 (0.017)	0.517 (0.003)	2.317 (0.013)	4.670
(3 , 3 , 0)	0.728 (0.013)	0.651 (0.002)	0.000 (0.000)	4.137
(4 , 0 , 2)	0.219 (0.008)	0.000 (0.000)	1.999 (0.007)	4.874
(4 , 1 , 1)	0.386 (0.008)	0.589 (0.006)	2.317 (0.013)	4.448
(4 , 2 , 0)	0.605 (0.009)	0.741 (0.006)	0.000 (0.000)	3.901
(5 , 0 , 1)	0.367 (0.007)	0.000 (0.000)	2.317 (0.013)	4.153
(5 , 1 , 0)	0.550 (0.005)	0.823 (0.006)	0.000 (0.000)	3.572
(6 , 0 , 0)	0.527 (0.004)	0.000 (0.000)	0.000 (0.000)	3.160

Table 7.7: Two-player game: Payoffs are normalized value served over the duration of the experiment. Best responses are italicized.

7.4.3 Varying Supply and Demand

I vary the value of λ in the Poisson distribution and the number of machines to test the effect of a change in demand and supply, respectively. I assume all machines elect the *Consensus* strategy, since it is a dominant strategy equilibrium. As

λ increases, the relative efficiency of the system with respect to the value-optimal allocation increases.

As the number of machines increases, the proportion of value served increases. However, the revenue earned by each additional machine decreases as seen in Table 7.8.

Number of Machines	Additional Revenue
1	230.1
2	218.0
3	214.5
4	205.88
5	189.86

Table 7.8: Additional Revenue Earned by Adding Machines

7.4.4 Relaxing the monotonic prices constraint

I study the cost of imposing monotonic prices constraints by comparing the revenue earned when monotonic prices constraints are present to that when the constraints are relaxed and resource providers are allowed to freely decrease prices, for each of the strategies.

In each run of the experiment, all machines play the same strategy, and the average revenue per machine is computed with and without the monotonicity constraint. We observe that monotonic prices constraints are the most costly given a resource provider using the load-based strategy, and have smaller effect for the elasticity-based heuristic and the consensus strategy. When the Consensus strategy is employed, imposing monotonic prices constraints sacrifices less than 20% of the total revenue.

Strategy	% Revenue loss	Standard error
Load	41.84%	2.15%
Elasticity	78.36%	1.94%
Consensus	81.41%	4.73%

Table 7.9: Relaxing monotonicity

Table 7.9 shows the proportion of total revenue earned when monotonic prices constraints are imposed relative to when the constraints are relaxed.

Chapter 8

Conclusions

I introduced the problem of verifying whether a mechanism is strategyproof and provided a constraint-network based algorithm for verification. The verifier is able to reject mechanisms that are not strategyproof based on the violation of constraints imposed by strategyproofness on the price space. Experimental results demonstrate the potential for accelerated checking when there is additional structure to exploit and also suggests metrics that can be informative in guiding bidders before a mechanism is fully verified. Useful intermediate guarantees can also be provided to participants.

In practice, I believe that the most benefit from passive verification (as defined here) will be realized in combination with *approximations* that allow the verifier to forget some of its history and further reduce the space complexity of verifiers. In future work, limited memory checking, where one lets the verifier forget part of the history, and allow false positives, can be explored; e.g. using data structures such as Bloom filters [13] to allow for fast checking. Another important avenue for future work is to introduce methodologies that can allow for a continuous type space.

I presented a open framework with which to support truthful, dynamic and decentralized auctions in computational grids. The framework is designed to be extensible, aligned with incentives, simple for users, and allow for distributed and autonomous control of resources. Some innovations described in this work include: the use of resource prediction in quoting prices, embracing openness and extensibility, the infrastructure's role of enforcing rules to maintain strategyproofness, and the concepts of uniform-failure and threshold-reliability beliefs that allow for a technical analysis of strategyproofness.

I explored the use of machine learning methods for resource prediction when monotonicity constraints are imposed, and presented the relative accuracy of each method from a simulation using Crimson Grid data. I designed and implemented various monotonic pricing strategies and presented their revenue and value efficiency when facing different arrival distributions. These components can be replaced over time with ones employing more advanced techniques, driven by innovation and competition. The ultimate goal of this work is to deploy a market-based system to support efficient resource allocation across multiple administrative domains on an open computational infrastructure.

Key challenges that remain include how to ensure privacy, security, and reliability in open systems. A verifier that checks the integrity of results may be beneficial. The issues of control and ownership of data that arise, for example, if a user wants to move her computational job to another resource provider, possibly a competitor, are not yet resolved. The initial distribution of wealth in virtual marketplaces may have a significant effect on the global behavior of systems. An interesting future project may

quantify the effects that various initial distributions of wealth have on the efficiency of resource allocation.

Bibliography

[1] World community grid. *http://www.worldcommunitygrid.org/*.

[2] David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Gener. Comput. Syst.*, 18(8):1061–1074, 2002.

[3] David P. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[4] Owen Appleton. EGEE-II: a grid for research. *iSGTW Issue 18*, April 2007.

[5] A Archer and E Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science*, 2001.

[6] Baruch Awerbuch, Yossi Azar, and Adam Meyerson. Reducing truth-telling online mechanisms to online optimization. In *Proc. ACM Symposium on Theory of Computing (STOC'03)*, 2003.

[7] Martin Backschat, Alexander Pfaffinger, and Christoph Zenger. Economic-based dynamic load distribution in large workstation networks. In *Euro-Par, Vol. II*, pages 631–634, 1996.

[8] Albert D Baker. Metaphor or reality: A case study where agents bid with actual costs to schedule a factory. In Clearwater [17].

[9] Y Bartal, R Gonen, and N Nisan. Incentive compatible multi unit combinatorial auctions. Technical report, The Hebrew University of Jerusalem, 2003.

[10] Russell Bent and Pascal Van Hentenryck. The value of consensus in online stochastic scheduling. *In Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 219–226, 2004.

[11] S. Bikhchandani, S. Chatterjee, and A. Sen. Incentive compatibility in multi-unit auctions. Technical report, Working Paper, 2004.

[12] Avrim Blum, Vijar Kumar, Atri Rudra, and Felix Wu. Online learning in online auctions. In *Proceedings of the 14th Annual ACM-SIAM symposium on Discrete algorithms*, 2003.

[13] A. Broder and M.Mitzenmacher. Network applications of bloom filters: A survey. *Internet Math*, 1, no. 4, 2003.

[14] J. Brunelle, P. Hurst, J. Huth, L. Kang, C. Ng, D. Parkes, M. Seltzer, J. Shank, and S. Youssef. Egg: An extensible and economics-inspired open grid computing platform. In *Proceedings of the 2006 Grid Asia*, Singapore, May 2006.

[15] R. Buyya and S. Vazhkudai. Compute power market: Towards a market-oriented grid. *The First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, 2001.

[16] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1507–1542, 2002.

[17] Scott H Clearwater, editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[18] Scott H. Clearwater, Rick Costanza, Mike Dixon, and Brian Schroeder. Saving energy using market-based control. In Clearwater [17], chapter 10, pages 253–273.

[19] G. F. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. In *Machine Learning*, pages 309–347, 1992.

[20] Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. MIT Press, January 2006.

[21] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence Journal*, 49:61–95, 1991.

[22] Joan Feigenbaum and Scott Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, 2002.

[23] Amos Fiat, Andrew Goldberg, Jason Hartline, and Anna Karlin. Competitive generalized auctions. In *Proc. 34th ACM Symposium on Theory of Computing (STOC'02)*, 2002.

[24] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of SuperComputer Applications*, 15(3), 2001.

[25] A. Garcia-Camino, P. Noriega, and J. A. Rodriguez-Aguilar. Implementing norms in electronic institutions. In *Proceedings of the 4th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS-05)*, 2005.

[26] A. Gibbard. Manipulation of voting schemes: a general result. *Econometrica*, 41:587–601, 1973.

[27] Andrew Goldberg and Jason Hartline. Envy-free auctions for digital goods. In *Proc. ACM E'Commerce*, 2003.

[28] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. In *Combinatorica*, pages 301–337), volume = 20(3), 2000.

[29] J. Gomoluch and M. Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Proceedings of the First International Workshop on Middleware for Grid Computing (MGC '03)*, 2003.

[30] F. Guerin and J. V. Pitt. Guaranteeing properties for e-commerce systems. In O. Shehory and N. Sadeh J. Padget, D. Parkes, editor, *LNAI 2531: Agent-Mediated Electronic Commerce IV, pages 253-272. Springer-Verlag*, 2002.

[31] Honwei Gui, Rudolf Muller, and Rakesh V Vohra. Dominant strategy mechanisms with multidimensional types. Technical report, Northwestern Univ., 2004.

[32] Mohammad T. Hajiaghayi, Robert Kleinberg, Mohammad Mahdian, and David C. Parkes. Online auctions with re-usable goods. In *Proc. ACM Conf. on Electronic Commerce*, pages 165–174, 2005.

[33] Holger H Hoos and Craig Boutilier. Solving combinatorial auctions with stochastic local search. In *Proc. 17th National Conference on Artificial Intelligence (AAAI-00)*, July 2000.

[34] Matthew O. Jackson. Mechanism theory. In *The Encyclopedia of Life Support Systems*. EOLSS Publishers, 2000.

[35] J Johnston. *Econometric Methods*. McGraw-Hill, 1984.

[36] P.R. Jordan, C. Kiekintveld, and M.P. Wellman. Empirical game-theoretic analysis of the tac supply chain game. *In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1188–1195, 2007.

[37] Robert Kleinberg and Tom Leighton. The value of knowing a demand curve: Bounds on regret for online posted-price auctions. In *Proc. 44th Annual Symposium on Foundations of Computer Science*, 2003.

[38] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky. SETI@home - massively distributed computing for SETI. *Computing in Science and Engineering*, 3:78–83, 2001.

[39] K Kuwabara, T Ishida, Y Nishibe, and T Suda. An equilibratory market-based approach for distributed resource allocation and its applications to communication network control. In Clearwater [17], chapter 3, pages 53–73.

[40] P. Langley, W. Iba, and K. Thomas. An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference of Artificial Intelligence*, pages 223–228. AAAI Press, 1992.

[41] Steffen L. Lauritzen. Propagation of probabilities, means, and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87(420):1098–1108, 1992.

[42] R. Lavi, A. Mu'alem, and N. Nisan. Towards a characterization of truthful combinatorial auctions. In *Proc. 44th Annual Symposium on Foundations of Computer Science*, 2003.

[43] Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proc. 2nd ACM Conf. on Electronic Commerce (EC-00)*, 2000.

[44] Daniel Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, September 2002.

[45] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auctions.

[46] Donald B. Marron and Carlton W. Bartels. The user of computer-assisted auctions for allocating tradeable pollution permits. In Clearwater [17], chapter 11, pages 274–300.

[47] Andreu Mas-Colell, Michael D Whinston, and Jerry R Green. *Microeconomic Theory*. Oxford University Press, 1995.

[48] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[49] A. Mu'alem. A note on testing truthfulness. Technical Report 130, Electronic Colloquium on Computational Complexity, 2005.

[50] Robert B Myerson. Optimal auction design. *Mathematics of Operation Research*, 6:58–73, 1981.

[51] M. Naor, B. Pinkas, and O. Reingold. Privacy preserving auctions and mechanism design. In *Proc. 1st ACM Conf. on Electronic Commerce (EC-99)*, pages 129–139, 1999.

[52] John Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences*, volume 36, pages 48–49, 1950.

[53] Chaki Ng, Philip Buonadonna, Brent N. Chun, Alex C. Snoeren, and Amin Vahdat. Addressing Strategic Behavior in a Deployed Microeconomic Resource. In *3rd Workshop on the Economics of Peer to Peer Systems*, Philadelphia, PA, 2005.

[54] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the POPCORN project. In *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*, page 592, Washington, DC, USA, 1998. IEEE Computer Society.

[55] S. Papai. Groves sealed bid auctions of heterogeneous objects with fair prices. *Social Choice and Welfare*, 20:371–385, March 2003.

[56] D. C. Parkes, M. O. Rabin, S. M. Shieber, and C. A. Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. In *Proc. 8th Int. Conf. on Electronic Commerce (ICEC'06)*, 2006.

[57] David C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[58] David C Parkes, Jayant R Kalagnanam, and Marta Eso. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 1161–1168, 2001.

[59] M. Pauly. Programming and verifying subgame perfect mechanisms. *Journal of Logic and Computation*, 15(3):295-316, 2005.

[60] M. Pauly and M. Wooldridge. Logic for mechanism design - a manifesto. *Proceedings of the Game Theoretic and Decision Theoretic Agents Workshop*, 2003.

[61] M. J. Pazzani. Search for dependencies in bayesian classifiers. In D. Fisher and H. J. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V.* Springer Verlag, 1996.

[62] Ryan Porter. Mechanism design for online real-time scheduling. In *Proc. ACM Conf. on Electronic Commerce (EC'04)*, 2004.

[63] J. Quinlan. C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, CA, 1993.

[64] Kevin Roberts. The characterization of implementable rules. In Jean-Jacques Laffont, editor, *Aggregation and Revelation of Preferences*, pages 321–348. North-Holland, Amsterdam, 1979.

[65] Michael Saks and Lan Yu. Weak monotonicity suffices for truthfulness on convex domains. In *Proceedings of the 6th ACM conference on Electronic commerce (EC '05)*, 2005.

[66] Jeffrey Shneidman and David C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC'04)*, St. John's, Canada, 2004.

[67] Jeffrey Shneidman, David C. Parkes, and Laurent Massoulie. Faithfulness in Internet algorithms. In *Proc. SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04)*, Portland, USA, 2004.

[68] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

[69] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical atl model checking. In *Proceedings of the 5th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-06)*, 2006.

[70] Sivakumar Viswanathan, Bharadwaj Veeravalli, Dantong Yu, and Thomas G. Robertazzi. Design and analysis of a dynamic scheduling strategy with resource estimation for large-scale grid systems. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 163–170, Washington, DC, USA, 2004.

[71] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992.

[72] Leon Walras. Elements of pure economics. 1877, printed 1954.

[73] Michael P Wellman. Market-oriented programming: Some early lessons. In Clearwater [17], chapter 4, pages 74–95.

[74] Makoto Yokoo. The characterization of strategy/false-name proof combinatorial auction protocols: Price-oriented, rationing-free protocol. In *Proc. 18th Int. Joint Conf. on Art. Intell.*, 2003.

[75] E. Yom-Tov and Y. Aridor. Improving resource matching through estimation of actual job requirements. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006)*, 2006.

[76] Katie Yurkewicz. Crimson grid aids campus computing and collaboration. Science Grid This Week, August 2006.