

The Positronic Economist

A Computational System for Analyzing Economic Mechanisms

by

David R. M. Thompson

B.Sc., University of Guelph, 2004

M.Sc., University of British Columbia, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

May 2015

© David R. M. Thompson, 2015

Abstract

A mechanism is a formal protocol for collective decision making among self-interested agents. Mechanisms model many important social processes from auctions to elections. They are also widely studied in computer science: the participants in real-world mechanisms are often autonomous software systems (e.g., algorithmic bidding and trading agents), and algorithmic problems (e.g., job scheduling) give rise to mechanisms when users have competing interests.

Strategic behavior (or “gaming”) poses a major obstacle to understanding mechanisms. Although real-world mechanisms are often fairly simple functions (consider, e.g., plurality voting), a mechanism’s outcome depends on both this function and on agents’ strategic choices. Game theory provides a principled means for analyzing such choices. Unfortunately, game theoretic analysis is a difficult process requiring either human effort or very large amounts of computation.

My thesis is that mechanism analysis can be done computationally, due to recent advances in compact representations of games. Compact representation is possible when a game’s description has a suitable independence structure. Exploiting this structure can exponentially decrease the space required to represent games, and exponentially decrease the time required to analyze them.

The contributions of my thesis revolve around showing that the relevant kinds of structure (specifically, the structure exploited by action-graph games) are present in important mechanisms of interest. Specifically, I studied two major classes of mechanisms, position auctions (as used for internet advertising) and voting rules. In both cases, I was able to provide exponential improvements in the space and time complexity of analysis, and to use those improvements to address important open questions in the literature. I also introduced a new algorithm for analyzing action-graph games, with faster empirical performance and additional benefits over the previous state-of-the-art. Finally, I created the Positronic Economist system, which consists of a python-based descriptive language for mechanism games, with automatic discovery of computationally useful structure.

Preface

This thesis contains four previously published sections, each of which involved collaboration with other researchers.

Chapter 3 was co-authored with Samantha Leung and Kevin Leyton-Brown and was published at the Workshop on Internet and Network Economics [100]. I designed the algorithm with some input from Samantha, who did the majority of the software implementation. I did the theoretical and experimental evaluation of the algorithm and wrote the majority of paper. Kevin provided guidance throughout and wrote the remainder of the paper.

Chapter 4 was co-authored with Kevin Leyton-Brown and was published in the ACM Conference on Electronic Commerce [98]. Kevin and I did the high-level designing of the representation and experiments. I did the low-level design, implementation, experiments and analysis. Kevin and I wrote the paper.

Chapter 5 was co-authored with Kevin Leyton-Brown and was published in the ACM Conference on Electronic Commerce [99]. I provided the initial impetus for this project, and did the implementation, experiments and analysis. Kevin provided guidance, and he and I wrote the paper.

Chapter 6 was co-authored with Omer Lev, Jeffrey Rosenschein and Kevin Leyton-Brown and was published at the International Conference on Autonomous Agents and Multiagent Systems [101]. I designed the algorithm and the experiments (with some guidance from Omer Lev). Omer and I shared in the analysis of the data. Omer wrote the majority of the paper, with substantial contributions from Jeffrey Rosenschein, Kevin Leyton-Brown and I.

Table of Contents

Abstract	ii
Preface	iii
Table of Contents	iv
List of Tables	ix
List of Figures	xi
Glossary	xiv
Acknowledgments	xvi
Dedication	xvii
1 Introduction	1
2 Background	6
2.1 Compact Games	6
2.2 Nash-Equilibrium-Finding Algorithms	8
2.2.1 The Global Newton Method and Simplicial Subdivision	9
2.3 Other Computational Mechanism Analysis	9
3 The Support-Enumeration Method for Action-Graph Games	11
3.1 Introduction	11
3.2 Technical Background	12
3.3 SEM for AGGs	13
3.3.1 Conditional Dominance	14
3.3.2 TGS Feasibility Program	16
3.3.3 Asymptotic Analysis of SEM for AGGs	17
3.3.4 Further Speedups for k -Symmetric Games	18
3.3.5 Examples	19
3.4 Experimental Evaluation	21
3.4.1 Experimental Setup	22
3.4.2 Results	23

3.5	The Hardness of Generalizing AGG-SEM	24
3.6	Conclusion	29
4	Application: Position Auctions for Internet Advertising	31
4.1	Introduction	31
4.2	Background	33
4.2.1	Metrics for Evaluating Auction Outcomes	34
4.2.2	Models of Bidder Valuations	35
4.3	Representing Position Auctions	38
4.3.1	Representing No-Externality GFPs as AGGs	38
4.3.2	Representing No-Externality uGSPs and wGSPs as AGGs	39
4.3.3	Representing Auctions with Externalities	41
4.4	Experimental Setup	41
4.4.1	Problem Instances	42
4.4.2	Equilibrium Computation	44
4.4.3	Benchmarks: VCG and Discretized VCG	44
4.4.4	Statistical Methods	45
4.5	Results	45
4.5.1	Main Comparison	46
4.5.2	Basic Models: EOS and V	46
4.5.3	Position-Preference Models	53
4.5.4	Externality Models	58
4.6	Sensitivity Analysis	65
4.6.1	Sensitivity to Bid Increment Size	65
4.6.2	Sensitivity to Tie-Breaking Rules	68
4.6.3	Sensitivity to Rounding Rules	68
4.6.4	Sensitivity to the Number of Bidders	68
4.6.5	Sensitivity to the Number of Slots	69
4.7	Conclusions and Future Work	72
4.8	Summary Tables and Statistical Comparisons	75
5	Application: Maximizing Internet Advertising Revenue	94
5.1	Introduction	94
5.2	Background	95
5.2.1	Model	96
5.2.2	Related Work	97
5.3	First Analysis: Single-Slot Auctions with Known Quality Scores	98
5.4	Second Analysis: Multiple Slots, All Pure Nash Equilibria	102
5.5	Third Analysis: Multiple Slots, Equilibrium Refinement	105
5.6	Conclusions and Future Work	111
6	Application: Strategic Voting in Elections	113
6.1	Introduction	113
6.1.1	Related Work	115
6.2	Definitions	116
6.3	Method	118

6.4	Pure-Strategy Nash Equilibrium Results	119
6.4.1	Selectiveness of the Truthfulness Incentive	119
6.4.2	Equilibrium Outcomes	120
6.4.3	Scaling Behavior and Stability	122
6.4.4	Richer Mechanisms	123
6.5	Bayes-Nash Equilibria Results	125
6.6	Discussion and Future Work	128
7	The Positronic Economist	129
7.1	Introduction and Motivation	129
7.1.1	Related work	131
7.2	Representing Games in PosEc	132
7.2.1	Mechanisms and Settings	132
7.2.2	The PosEc Modeling Language	133
7.2.3	Special Functions in PosEc	134
7.2.4	Projection	135
7.3	Structure Inference in PosEc	135
7.3.1	White-Box Structure Inference	136
7.3.2	Black-Box Structure Inference	137
7.4	Experimental Results	138
7.5	Conclusion and Future Work	142
	Bibliography	144
A	Using the PosEc API	151
A.1	Representing Settings	151
A.2	Set-Like Classes for Outcome Spaces	153
A.3	Representing Mechanisms	154
A.4	Accessor Methods	155
B	Documentation of the PosEc API	160
B.1	Module posec.bbsi	161
B.1.1	Functions	161
B.1.2	Variables	161
B.2	Module posec.mathtools	162
B.2.1	Functions	162
B.2.2	Variables	162
B.3	Module posec.posec_core	163
B.3.1	Functions	163
B.3.2	Variables	163
B.3.3	Class ActionProfile	163
B.3.4	Class TypeProfile	165
B.3.5	Class RealSpace	166
B.3.6	Class CartesianProduct	167
B.3.7	Class Setting	168
B.3.8	Class ProjectedSetting	168

B.3.9	Class BayesianSetting	169
B.3.10	Class ProjectedBayesianSetting	170
B.3.11	Class Mechanism	171
B.3.12	Class ProjectedMechanism	171
B.3.13	Class Distribution	172
B.3.14	Class UniformDistribution	172
B.4	Module posec.pyagg	174
B.4.1	Functions	174
B.4.2	Variables	174
B.4.3	Class AGG_File	175
B.4.4	Class BAGG_File	175
B.4.5	Class AGG	176
B.4.6	Class BAGG	179
C	Documentation of the Included PosEc Applications	181
C.1	Package applications	182
C.1.1	Modules	182
C.1.2	Variables	182
C.2	Module applications.basic_auctions	183
C.2.1	Functions	183
C.2.2	Variables	183
C.2.3	Class SingleGoodOutcome	183
C.2.4	Class ProjectedOutcome	185
C.2.5	Class FirstPriceAuction	186
C.2.6	Class AllPayAuction	187
C.3	Module applications.position_auctions	188
C.3.1	Functions	188
C.3.2	Variables	188
C.3.3	Class Permutations	189
C.3.4	Class NoExternalitySetting	189
C.3.5	Class NoExternalityPositionAuction	190
C.4	Module applications.position_auctions.externalities	192
C.4.1	Functions	192
C.4.2	Variables	192
C.4.3	Class HybridSetting	192
C.4.4	Class ExternalityPositionAuction	193
C.5	Module applications.voting	195
C.5.1	Functions	195
C.5.2	Variables	196
C.5.3	Class AbstractVotingMechanism	196
C.5.4	Class VoteTuple	197
C.5.5	Class Plurality	198
C.5.6	Class Approval	199
C.5.7	Class kApproval	200
C.5.8	Class Veto	201
C.5.9	Class Borda	202

C.5.10 Class InstantRunoff 203

List of Tables

Table 3.1	Mean runtimes of AGG-SEM and the three incumbents algorithms	25
Table 4.1	The distributions we considered for our position-auctionnn experiments	43
Table 4.2	Comparing Efficiency (EOS-UNI distribution)	75
Table 4.3	Comparing Revenue (EOS-UNI distribution)	75
Table 4.4	Comparing Envy (EOS-UNI distribution)	76
Table 4.5	Comparing Efficiency (EOS-LN distribution)	77
Table 4.6	Comparing Revenue (EOS-LN distribution)	77
Table 4.7	Comparing Envy (EOS-LN distribution)	77
Table 4.8	Comparing Efficiency (V-UNI distribution)	78
Table 4.9	Comparing Revenue (V-UNI distribution)	78
Table 4.10	Comparing Relevance (V-UNI distribution)	79
Table 4.11	Comparing Envy (V-UNI distribution)	79
Table 4.12	Comparing Efficiency (V-LN distribution)	80
Table 4.13	Comparing Revenue (V-LN distribution)	80
Table 4.14	Comparing Relevance (V-LN distribution)	81
Table 4.15	Comparing Envy (V-LN distribution)	81
Table 4.16	Comparing Efficiency (BHN-UNI distribution)	82
Table 4.17	Comparing Revenue (BHN-UNI distribution)	82
Table 4.18	Comparing Relevance (BHN-UNI distribution)	83
Table 4.19	Comparing Envy (BHN-UNI distribution)	83
Table 4.20	Comparing Efficiency (BSS distribution)	83
Table 4.21	Comparing Revenue (BSS distribution)	84
Table 4.22	Comparing Envy (BSS distribution)	84
Table 4.23	Comparing Efficiency (CAS-UNI distribution)	84
Table 4.24	Comparing Revenue (CAS-UNI distribution)	85
Table 4.25	Comparing Relevance (CAS-UNI distribution)	85
Table 4.26	Comparing Efficiency (CAS-LN distribution)	86
Table 4.27	Comparing Revenue (CAS-LN distribution)	86
Table 4.28	Comparing Relevance (CAS-LN distribution)	87
Table 4.29	Comparing Efficiency (HYB-UNI distribution)	87
Table 4.30	Comparing Revenue (HYB-UNI distribution)	88
Table 4.31	Comparing Relevance (HYB-UNI distribution)	88
Table 4.32	Comparing Efficiency (HYB-LN distribution)	89
Table 4.33	Comparing Revenue (HYB-LN distribution)	89

Table 4.34	Comparing Relevance (HYB-LN distribution)	90
Table 4.35	Comparing Efficiency (GIM-UNI distribution)	90
Table 4.36	Comparing Revenue (GIM-UNI distribution)	91
Table 4.37	Comparing Relevance (GIM-UNI distribution)	91
Table 4.38	Comparing Efficiency (GIM-LN distribution)	92
Table 4.39	Comparing Revenue (GIM-LN distribution)	92
Table 4.40	Comparing Relevance (GIM-LN distribution)	93
Table 5.1	Comparing GSP variants for two bidders	99
Table 5.2	Comparing the various auction variants given their optimal parameter settings (best- and worst-case equilibria)	103
Table 5.3	Comparing the various auction variants given their optimal parameter settings (incentive-compatible equilibrium)	109

List of Figures

Figure 1.1	Process flow in the Positronic Economist system	3
Figure 1.2	An example of equilibrium analysis of voting	4
Figure 2.1	An example of an action graph: the ice-cream game	7
Figure 3.1	The Test-Given-Support (TGS) feasibility program for n -player games	12
Figure 3.2	Iterative Removal of Strictly Dominated Strategies	13
Figure 3.3	A visualization of Porter <i>et al</i> 's [84] tree-search through support space	14
Figure 3.4	ConditionallyDominatedAGG	16
Figure 3.5	RecursiveCD	16
Figure 3.6	Visualizing the outer-loop tree search process followed by AGG-SEM when enumerating PSNEs	20
Figure 3.7	Visualizing a ConditionallyDominatedAGG search	21
Figure 3.8	Scatterplot contrasting the runtimes of AGG-SEM and NFG-SEM	23
Figure 3.9	An example 3SAT instance reduced to dominance in an AGG-FNA	26
Figure 3.10	An example INDEPENDENTSET instance reduced to dominance in a BAGG	28
Figure 3.11	Runtime CDFs for AGG-SEM and the three incumbent algorithms	29
Figure 3.12	Per-distribution runtime CDFs for all four algorithms	30
Figure 4.1	A weighted GFP represented as an AGG	39
Figure 4.2	An algorithm for converting a no-externality auction setting into an action graph representing a (weighted) GFP	40
Figure 4.3	A weighted GSP represented as an AGG	41
Figure 4.4	An algorithm for converting an auction setting into an action graph representing a wGSP	42
Figure 4.5	Creating the action graph for a GIM position auction	43
Figure 4.6	Performance of different position auction types, averaged across all 13 distributions	47
Figure 4.7	Empirical cumulative probability distributions over the number of equilibria in EOS and V models	48
Figure 4.8	Empirical box plot of wGSP's envy under different equilibrium selection criteria	48
Figure 4.9	Empirical box plot of wGSP's social welfare under different equilibrium selection criteria	49
Figure 4.10	Empirical box plot of wGSP's revenue under different equilibrium selection criteria	50
Figure 4.11	Empirical box plot of uGSP and wGSP's revenue under different equilibrium selection criteria	50
Figure 4.12	Empirical box plot of uGSP and wGSP's social welfare under different equilibrium selection criteria	51

Figure 4.13	Comparing the average performance of different position auction types in EOS and V settings	52
Figure 4.14	Comparing wGFP to wGSP	53
Figure 4.15	Empirical CDF of economic efficiency in BHN models	54
Figure 4.16	Comparing the average performance of different position auction types in BHN settings	55
Figure 4.17	Empirical CDF of economic efficiency in BSS models	56
Figure 4.18	Comparing the average performance of different position auction types in BSS settings.	57
Figure 4.19	Empirical CDF of economic efficiency in the cascade models.	58
Figure 4.20	Empirical CDF of revenue in the cascade models.	59
Figure 4.21	Comparing the average performance of different position auction types in cascade settings	59
Figure 4.22	Comparing the average performance of wGSP and cwGSP in cascade settings	60
Figure 4.23	Empirical CDF of revenue in the hybrid model.	61
Figure 4.24	Empirical CDF of economic efficiency in the hybrid model.	61
Figure 4.25	Comparing the average performance of different position auction types in hybrid settings	62
Figure 4.26	Empirical CDF of economic efficiency in the GIM model.	63
Figure 4.27	Empirical CDF of revenue in the GIM model.	63
Figure 4.28	wGSP tended to have good worst-case efficiency when the top position produced the majority of the surplus	64
Figure 4.29	Comparing the average performance of different position auction types in GIM settings	64
Figure 4.30	Comparing different auction designs as the number of bid increments varies.	66
Figure 4.31	Comparing different auction designs as the number of bid increments varies (continued).	67
Figure 4.32	How tie breaking affects auction outcomes	69
Figure 4.33	How price rounding affects auction outcomes	70
Figure 4.34	Comparing different auction designs as the number of agents varies.	71
Figure 4.35	Comparing different auction designs as the number of agents varies (continued).	72
Figure 4.36	Comparing different auction designs as the number of slots varies.	73
Figure 4.37	Comparing different auction designs as the number of slots varies (continued).	74
Figure 5.1	Visualizing the allocation functions of VCG, anchoring and squashing	100
Figure 5.2	Visualizing the allocation functions of UWR and QWR	101
Figure 5.3	Visualizing the allocation functions of UWR+Sq and QWR+Sq	101
Figure 5.4	Visualizing the allocation function of the Myerson auction (log-normal distribution)	102
Figure 5.5	Revenue, parameter settings and equilibrium selection for six GSP variants	104
Figure 5.6	Visualizing-incentive compatible price computation in position auctions	105
Figure 5.7	Effect of squashing on revenue in incentive-compatible equilibrium	107
Figure 5.8	Effect of UWR and QWR on revenue in incentive-compatible equilibrium	107
Figure 5.9	Marginal effect of squashing on revenue in incentive-compatible equilibrium (with UWR present)	108
Figure 5.10	Marginal effect of squashing on revenue in incentive-compatible equilibrium (with QWR present)	108
Figure 5.11	Effect on revenue when squashing is only applied to QWR's reserve	109
Figure 5.12	Effect on revenue when squashing is only applied to QWR's post-reserve allocation	110
Figure 5.13	Comparing GSP variants while varying the number of bidders	110

Figure 6.1	An action graph game encoding of a simple two-candidate plurality vote. Each round node represents an action a voter can choose. Dashed-line boxes define which actions are open to a voter given his preferences; in a Bayesian AGG, an agent’s type determines the box from which he is allowed to choose his actions. Each square is a sum node, tallying the number of votes a candidate received.	119
Figure 6.2	Even with the truthfulness incentive, many different outcomes were still possible in equilibrium.	120
Figure 6.3	With the truthfulness incentive, some outcomes occur much more frequently than others. . . .	121
Figure 6.4	CDF of social welfare.	122
Figure 6.5	The average proportion of equilibria won by candidates with average rank of 0–1, 1–2, etc. . . .	123
Figure 6.6	Percentage of games with a Nash equilibrium of a given type with 3 candidates and varying number of voters.	124
Figure 6.7	Varying the number of voters and candidates.	124
Figure 6.8	Empirical CDF of counts of equilibria.	125
Figure 6.9	Every instance had many equilibria, most of which only involved a few candidates.	127
Figure 7.1	The Three Laws of Positronic Economics	130
Figure 7.2	White-Box Structure Inference	136
Figure 7.3	Measuring the performance of WBSI on efficiently-represented games	139
Figure 7.4	Measuring the performance of the BBSI-ILS algorithm on tuned inputs	140
Figure 7.5	Measuring the performance of the BBSI-ILS algorithm on untuned inputs	141
Figure 7.6	Measuring the scales of BAGGs using structure learned by BBSI	143

Glossary

- AGG – Action-graph game
- AGG-SEM – The support-enumeration algorithm, using action-graph-game structure
- BBSI – Black-box structure inference
- BNE – Bayes-Nash equilibrium
- BNIC – Bayes-Nash incentive compatible
- GFP – The generalized first-price position auction
- GNM – Govindan and Wilson’s global Newton method for finding Nash equilibria
- GSP – The generalized second-price position auction
- IC – Incentive compatible (a property of some mechanisms)
- IRSDS – Iterative removal of strictly dominated strategies
- MSNE – Mixed-strategy Nash equilibrium
- NE – Nash equilibrium
- NFG – Normal-form game
- NFG-SEM – The support-enumeration algorithm, using normal-form-game structure
- NP – The set of all computational problems that can be solved in polynomial time by a non-deterministic Turing machine
- P – The set of all computational problems that can be solved in polynomial time by a deterministic Turing machine
- PSNE – Pure-strategy Nash equilibrium
- SEM – Porter, Nudelman and Shoham’s support-enumeration method for finding Nash equilibria (see Section 3.2)
- SimpDiv – The simplicial subdivision method for finding Nash equilibria

- TGS – Test given support subroutine
- uGSP – The unweighted generalized second-price position auction
- VCG – The Vickrey-Clark-Groves mechanism
- WBSI – White-box structure inference
- wGSP – The (quality-)weighted generalized second-price position auction

Acknowledgments

I would like to thank my supervisor, Kevin Leyton-Brown, and my committee members, Sergei Severinov and Holger Hoos, for their many hours of support and guidance. I also owe a debt to the rest of the faculty at UBC, and at the University of Guelph, who helped to shape and inspire me as a researcher.

I am grateful to my collaborators and peers, and for the great communities of AAAI, AAMAS, ACM EC, IJCAI, INFORMS, and the Algorithmic Game Theory semester at the Hebrew University of Jerusalem.

My research was generously supported by grants from Google, Microsoft and NSERC.

I am thankful for my friends and family, who were a constant source of support and joy.

Lastly, I would like to thank my beloved wife, Fern.

Dedication

To Fern

Chapter 1

Introduction

A mechanism is a formal protocol that a group of agents can use to arrive at a decision, even when those agents disagree about which choice is best. A simple example is a committee voting to choose a single recommendation from a small set of candidates. In computer science terms, a mechanism can be thought of as an algorithm whose inputs can be distorted by agents who want to influence the output.

Mechanism analysis is an increasingly important topic in computer science research. As the internet brings agents with competing interests together, many traditional computer science problems are now being investigated as mechanisms. Two important example problems are routing messages through a network [90], and scheduling tasks across a set of processors [78]; both are made more complicated by the presence of multiple users with competing interests. Mechanisms also provide important non-traditional application domains for artificial intelligence. One example application is AI agents that participate in mechanisms, e.g., automated auction bidders [72]. Another is AI systems that design novel mechanisms for specific, unusual settings [16].

The study of mechanisms also has many important applications outside the realm of computer science. Mechanisms are a model of many important social institutions, including auctions [58], elections [33, chapter 5] and matching systems [34] such as those used to match medical students to internships or to match organ donors with recipients.

Mechanism analysis is the investigation of what happens when a mechanism is run. However, treating a mechanism as a simple mapping from inputs to outputs misses a critical point; the inputs to the mechanism can be distorted by agents hoping to manipulate the output. To understand a mechanism one needs to understand the relationship between the outcome and the agents' *true* preferences. For example, when considering the question “who will win an election, if we use voting rule X ?,” the answer “whoever gets the most votes” is much less satisfactory than “whoever best reflects the actual preferences of the majority of voters.” This kind

of investigation is especially difficult because agents will not merely try to outwit the mechanism designer, but will also try to outwit each other.

Mechanism analysis means studying the interaction between a setting, i.e., the agents and their preferences, and a mechanism, but also involves a third element: a formal model of the strategic choices made by the agents. The models of choice come from game theory: the mechanism and setting specify a game, and game-theoretic solution concepts, e.g., Nash equilibrium, are used to identify the strategies that rational agents would follow in that game. Thus, mechanism analysis means answering the question “Given mechanism M and setting S , what happens in an equilibrium of type E ?”

One could object to this way of framing the problem, arguing that researchers should focus on mechanisms where agents do not have any incentive to manipulate the outcome. Such mechanisms are called incentive compatible (IC) or truthful. In fact, the revelation principle shows that incentive compatibility is without loss of generality, in the sense that every mechanism is equivalent to some incentive compatible mechanism [75]. However, the revelation principle does not eliminate the need for study of non-IC mechanisms for several reasons. One reason is that non-IC mechanisms are widely used in practice, e.g., simultaneous ascending auctions [18], plurality votes [33], the generalized second-price auction [103]. To replace any of these mechanisms with an equivalent IC mechanism first requires that one completely understand how agents behave in the original mechanism. IC mechanisms also have many inherent weakness that could make them impractical for many applications, including exponentially large computation and communication costs and a need for the revelation of confidential information. Another reason is that IC mechanisms are often only IC under restrictive assumptions about the agents’ preferences and abilities. When these assumptions are relaxed, the mechanism may be as manipulable as any other non-IC mechanism, e.g., when the Vickrey auction is used to sell to agents with common values [58], when multiple single-good Vickrey auctions sell goods to agents with combinatorial preferences [10], or when the combinatorial Vickrey auction is used to sell to agents who are capable of bidding under multiple identities [112].

My thesis is that mechanism analysis can be done computationally, leveraging compact representations of games. (See Figure 1.1.) Equilibrium computation has made substantial strides in the last decade. Importantly for my thesis, two major areas of progress are (1) designing and identifying algorithms that are fast in practice [80, 84], and (2) compactly representing games, along with algorithms that exploit this compactness for more efficient computation [51, 55, 91]. Using a compact game representation allows me to decouple the problem of representation, i.e., storing the game in a computationally useful form, from the problem of solving the game, i.e., computing a solution concept such as a Nash or correlated equilibrium. It also allows me to leverage existing algorithms and implementations for solving compact games, as well as any new algorithms that are developed concurrently. For a compact representation, I chose action-graph games (AGGs) [51], and their incomplete information counterpart Bayesian action-graph games (BAGGs) [47], because they are more compact than nearly any other representation¹ and there are preexisting fast

¹MAIDs [57] are more compact than AGGs for some games. However, MAIDs are lacking in practical implemented algorithms,

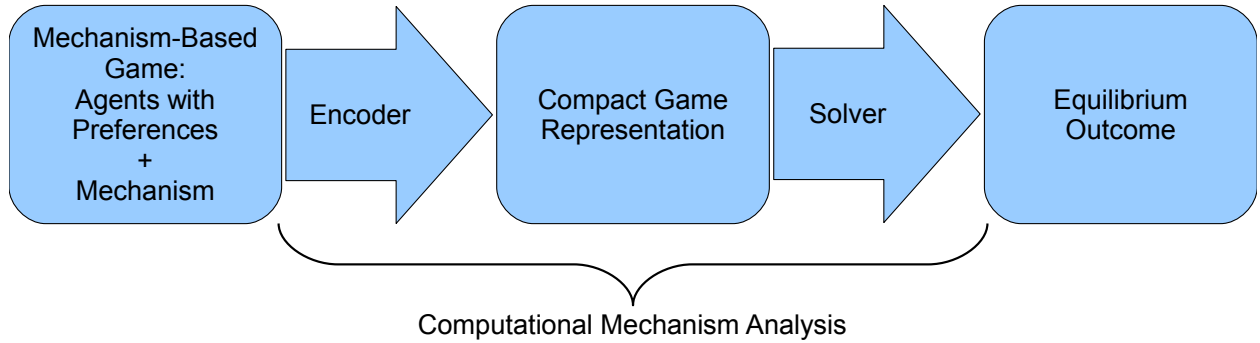


Figure 1.1: My thesis is that it is possible to do mechanism analysis, i.e., identifying equilibrium outcomes of non-truthful mechanisms, computationally. Computational mechanism analysis (CMA) is different from most other mechanism analysis in that the equilibrium-finding is done entirely by computer. To do this, I use compact game representations, action-graph games [51] and Bayesian action-graph games [47], as intermediate data structures.

implementations of state-of-the-art solvers for AGGs.

Beyond the freedom to choose among solver algorithms, our two-stage design also provides freedom to choose among solution concepts. To date, (Bayes) Nash equilibrium has tended to dominate the game-theoretic analysis of mechanisms. (For example, see the works cited in Chapter 4, which cover a broad range of analyses of internet advertising auctions.) However, arguments can be made for many other solution concepts such as correlated equilibrium, perfect equilibrium and rationalizability, as well as behavioural solution concepts such as quantal-response equilibrium or cognitive hierarchy. Computing these solution concepts often requires computing game-theoretic “building blocks” such as expected utility, best response and dominance. Thus, the (B)AGG-representation language, which allows for exponential speedup of these building blocks, can provide a substantial computational advantage. (See Roughgarden and Papadimitriou [91] for an example of how fast expected utility calculations can lead to exponentially faster correlated equilibrium computation.)

One could argue that any hard-to-compute solution concept is implausible, precisely because of its hardness. If an equilibrium is hard for researchers to find, then surely it must be hard for the agents to find as well; thus, the argument follows, researchers could better predict agent behaviour by using only easy-to-compute solution concepts, such as the limits of learning dynamics. Our approach could still be useful, even for a researcher studying such a solution concept. Learning dynamics can converge to many different outcomes depending on learning parameters (and random seeds), and it could be infeasible to bound the range of possible outcomes. However, many kinds of learning dynamics are guaranteed to converge to correlated equilibrium, and thus, an algorithm to compute optimal or extreme correlated equilibrium (e.g., the algorithm from Jiang and Leyton-Brown [49]) can be used to identify which outcomes are possible, while remaining

and even basic operations like calculating expected utility are typically NP-hard because they generalize an existing NP-hard problem: Bayes-Net inference [17].

$$\begin{array}{ll}
49 & : A \succ B \succ C \\
47 & : B \succ C \succ A \\
4 & : C \succ B \succ A
\end{array}$$

Figure 1.2: An example setting for demonstrating that truthful voting can be a Nash equilibrium in plurality voting, but that it can lead to the “wrong” outcome. The top line can be read as “49 voters prefer candidate A over candidate B , and candidate B over candidate C .” Voting truthfully, i.e., for your most preferred candidate, is a Nash equilibrium in this setting: because the final tallies separated by more than one vote, no agent has an incentive to deviate and strategically vote. In this Nash equilibrium, candidate A wins. However, candidate A is the wrong candidate in the sense that he is a “Condorcet loser,” i.e., he would lose in a pairwise vote against any other candidate.

agnostic about learning parameters.

Another significant, and fair, criticism of this approach is that it involves analyzing one game at a time, and thus, any result it produces will be sensitive to the exact parameters of the mechanism, setting and equilibrium-selection criteria that produced that specific game and equilibrium². These results can be called “existential” results; i.e., they show that a game with a particular property exists. (See Figure 1.2 for an example.) In contrast, the literature on mechanisms contains many “universal results,” i.e., results showing that some property holds for every mechanism-setting pair in some space. One example of a famous universal result is Myerson’s revenue equivalence theorem [75], which identifies a wide range of settings in which all economically efficient single-good auctions generate the same amount of revenue, in symmetric Bayes-Nash equilibrium. These results are not dependent on parameters in the same way that existential results are, but they are nevertheless limited, typically covering only a small space of mechanisms and settings. Using CMA it is often possible to explore much larger spaces. For example, the famous EOS/Varian [30, 103] results about economic efficiency in advertising auctions only applies to a very narrow range of settings, a single mechanism and a specific family of equilibria. In contrast, the CMA tools in Chapter 4 can evaluate economic efficiency for a wide range of settings, a large parameterized family of auctions, and the entire space of Nash equilibria, using the algorithm from Chapter 3.

Despite only producing existential results, CMA can make a wide range of contributions to the study of mechanisms:

- Additional analysis of even a single game can yield new insights. For example, the work in Chapter 4 includes one of the first analyses of what happens when the envy-free-ness assumption is relaxed in the

²The first part of the issue—that computational mechanism analysis (CMA) can only handle a single game at a time—seems to be intrinsic to our way of framing the problem. The second part—that CMA only works with arbitrary sample equilibria—can be overcome with better equilibrium-finding algorithms. Although most research into equilibrium-finding algorithms focuses on finding sample Nash equilibria, itself a computationally hard problem, there are algorithms that can go further, finding all equilibria of a particular type (see Chapter 3), or finding equilibria that provably optimize some objective (as in [48] for example).

advertising auction games of Edelman *et al* [30] and Varian [103].

- Although CMA cannot directly produce universal results, it can help human researchers to do so, through its ability to cheaply analyze large numbers of games. CMA helps to disprove hypotheses, by enumerating many games searching for a counterexample. CMA results can also inspire researchers to find new universal results based on unexpected patterns that appear in sample games. For example, we were able to partially characterize the space of Bayes-Nash equilibria in plurality voting games (Theorem 14) based on patterns found in a small set of sample games.
- By sampling settings, it is also possible to compute expected properties of a mechanism, e.g., expected revenue given an arbitrary sample-able distribution, possibly derived from real-world data. This also makes it possible to compare the expected performance of different mechanisms, even in cases where neither mechanism dominates the other.
- Computational mechanism analysis enables automated mechanism design; in this approach, mechanism design is framed as a constrained optimization where the search space is the parameter space of a parameterized mechanism and the objective is some property of equilibrium outcomes, computed using CMA. Vorobeychick *et al* [108] have had success with this approach, even using iterative best response, an extremely simple Nash-equilibrium-finding algorithm. Chapter 5 shows an extended example of revenue optimization using CMA. As in the case of voting, surprising experimental results have lead to new theoretical insights.
- Lastly, CMA can facilitate the computation of empirical bounds, such as on the price of anarchy.³

This document proceeds as follows. Chapter 2 contains formal models of the concepts described above, as well as relevant background. Chapter 3 describes a novel solver algorithm that became a key tool for my CMA techniques. Chapters 4 and 5 describe my first application domain, internet advertising auctions. Chapter 4 shows how my CMA approach can be used to compare different advertising auctions across a wide range of models. Chapter 5 demonstrates how mechanism analysis can facilitate mechanism design, searching for a profit-maximizing mechanism from a parameterized space of auction designs. Chapter 6 describes my second application domain, strategic voting in elections. Each application domain required building novel encoder algorithms, which involved substantial cognitive effort. Chapter 7 describes the Positronic Economist system, which simplifies, unifies and generalizes the CMA software used in the preceding chapters.

³ “Price of anarchy,” a worst-case bound on economic efficiency has recently become popular as a way of doing mechanism analysis. (See Chapters 17–21 of [79].) Price of anarchy results typically consist of a existential result, i.e., an example of a setting and equilibrium where this mechanism is a factor of x away from economic efficiency, and corresponding universal result, i.e., a proof that no setting-equilibrium pair exists where this mechanism is more than x away from economic efficiency.

Chapter 2

Background

A full review of game theory and mechanisms is far beyond the scope of this document; this chapter is written with the assumption that the reader is familiar with both topics. (See Chapters 3–6 and 9–11 of [96] for a detailed treatment.) Thus, this chapter discusses only more specialized areas of game theory. The first area is compact game representations, particularly action-graph games. These are essential for computational mechanism analysis: without them I would not be able even to store many games of interest. The second area is solvers, specifically Nash-equilibrium-finding algorithms. One of the major advantages of using an existing compact game representation is that I can directly apply these algorithms, rather than having to rely entirely on novel solvers. The third area is computational mechanism analysis: there is a very small existing literature on making general purpose algorithms for finding equilibrium outcomes of mechanisms.

Because each application domain has its own extensive body of research, I provide application-specific literature surveys in their respective chapters.

2.1 Compact Games

The first barrier to computational mechanism analysis is game representation. Since normal-form games grow exponentially in the number of players, they yield infeasibly large encodings of all but the simplest interactions. The literature contains many compact representations for simultaneous move games, for example congestion games [89], graphical games [55] and action-graph games [51]. For my purposes, action-graph games are the most useful, because they combine two important features. They’re very compactly expressive, i.e., if other representations can encode a game in polynomial-space then so can AGGs, and there are existing empirically fast tools for working with them.

Action-graph games achieve compactness by exploiting two kinds of structure in the payoffs. (There are

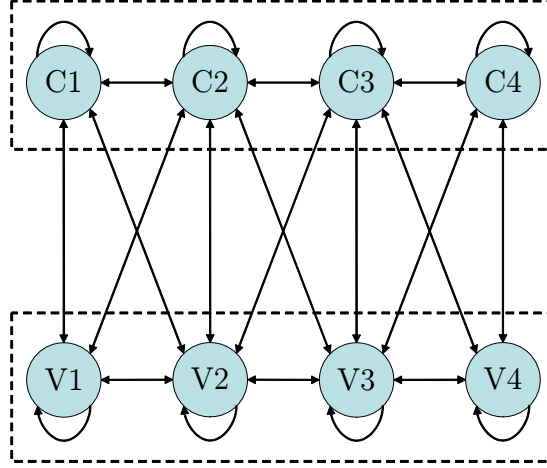


Figure 2.1: An action graph for an “ice-cream game” [51]. In an ice-cream game, each player has either chocolate or vanilla ice-cream to sell and must choose from a set of possible locations to set up his stand. The action graph encodes that a player’s payoff depends only on his location, and the type and number of competitors within one step of his location.

AGG extensions that can exploit additional types of structure.)

- *Anonymity* means that an agent’s payoff depends only on his own action, and the “configuration” induced by the other agents. A configuration is a tuple of counts of how many agents played each action. In other words, the agent’s payoff does not depend on who played which action. Therefore, instead of storing an action’s payoff for every pure-strategy profile of the other agents, AGGs only need to store one payoff for every configuration.
- *Context-specific independence* means that an agent’s payoff for playing some action only depends the counts on a subset of the actions. These independences are encoded in an action graph, a directed graph where each node corresponds to an action. The payoff for playing an action only depends on the counts on its neighboring nodes. Instead of storing an action’s payoff for every configuration, AGGs only need to store one for every “projected configuration,” a tuple of counts on the neighbors of a vertex, denoted $C^{(v)}$ for vertex v . See Figure 2.1 for an example.

Formally, an action-graph game is a 4-tuple $\langle N, A, G, u \rangle$ where N is the set of agents $\langle 1, 2, \dots, n \rangle$; $A = \prod_{i \in N} A_i$ is the set of action profiles; $G = \langle V, E \rangle$ is a directed graph with vertices V , where $V = \bigcup_{i \in N} A_i$, and edges E ; and $u = \langle u_1, u_2, \dots, u_{|V|} \rangle$ is a tuple of utility functions, where $u_v : C^{(v)} \mapsto \mathbb{R}$.

The other advantage of AGGs, beside their size, is that they can be reasoned about efficiently. In particular, given a mixed-strategy profile it is possible to compute the expected utility (for an agent i playing a particular action a_i) in polynomial time using dynamic programming [51]. This is important because expected utility is

the main bottleneck in two important equilibrium-finding algorithms. The dynamic program proceeds through n iterations, where at the k^{th} iteration, it computes the marginal distribution over the projected configurations $C^{(a_i)}$ given the strategies of the first k agents.

There are two other variants of AGGs that will be relevant to my work:

- AGGs with function nodes (AGG-FNs) [51] — These have additional nodes that correspond to functions rather than only actions. The projected configuration on a function node is a function of the projected configuration on its neighbourhood. These are sometimes exponentially smaller than AGGs.
- Bayesian AGGs (BAGGs) [47] — In Bayesian AGGs, each agent has a private random type, typically drawn independently from the other agents’ types, and each type has its own action set which is a subset of the available action nodes.

2.2 Nash-Equilibrium-Finding Algorithms

Nash equilibrium finding is one of the central problems in algorithmic game theory and as such, the literature on it is extensive, including numerous algorithms and hardness results. (See chapters 4–6 of [96] and chapters 1–9 of [79] for a overview.) Because the literature is so broad, I will focus specifically on Nash equilibrium finding in AGGs. The most powerful hardness result—i.e., that finding an sample Nash equilibrium is PPAD complete¹ [14, 20]—applies directly to all fully expressive compact representations, including AGGs. There are also applicable NP-hardness results: Finding a pure-strategy Nash equilibrium of a symmetric AGG with unbounded tree-width is NP-hard [19]. Finding a pure-strategy Nash equilibrium of a graphical game² is NP-hard [42]. Thus, it is unlikely that there exists a polynomial-time algorithm for finding Nash equilibria of general action-graph games. There have been some polynomial-time algorithms for finding Nash equilibria of restricted AGGs [19, 46] but these have gone unimplemented, and often the restrictions rule them unsuitable for my purposes, e.g., bounded treewidth, full symmetry, no function nodes.

In contrast, there have been promising results taking existing normal form algorithms and making them work on AGGs [51]. These modified algorithms have exponentially faster asymptotic runtime than their normal-form counterparts when inner loop operations are optimized for AGGs. They also often have very good performance in practice, quickly finding Nash equilibria of games that would be infeasible to even store on a modern hard drive if represented in the normal form. Below, I describe two algorithms, simplicial subdivision and the global Newton method, that were previously made exponentially faster for AGGs. In both cases, the algorithm’s bottleneck was calculating expected utility, and performance was improved by using AGG-specific expected utility routines. A third state-of-the-art Nash-equilibrium-finding algorithm, the

¹PPAD stands for “Polynomial Parity Arguments on Directed graphs.” This class also includes many other fixed-point problems, and there is no known polynomial time algorithm for any PPAD-complete problem.

²Graphical games are a specialization of AGGs. Graphical games only exploit a very strong form of independence, where two players are independent iff their actions can never affect each other’s payoffs.

support enumeration method [84], is covered in detail in Chapter 3. It has different bottlenecks from the other two algorithms, but I was still able to make it exponentially faster.

2.2.1 The Global Newton Method and Simplicial Subdivision

The global Newton method (GNM) [43] and simplicial subdivision (SimpDiv) [102] are two algorithms for finding sample Nash equilibria with many common features. Most critically, both use expected utility calculations as a critical inner-loop step, meaning that when analyzing AGGs they can be made exponentially faster by using computing expected utility according to Jiang et al’s dynamic programming method [51]. Both have been implemented in GAMBIT [68] with AGG-specific extensions by Jiang.

The global Newton method (GNM) of Govindan and Wilson is a homotopy method. This means that it works by tracing a continuous path from an easy-to-solve problem instance through to the actual problem instance of interest, maintaining a solution throughout the process. This particular algorithm works by tracing a space of games, starting from a game with strict dominant strategies and ending with the original input game, with interpolated payoffs on the games in between. GNM is a Las Vegas algorithm³: the starting game with the dominant strategy is generated at random, but the algorithm is always guaranteed to terminate eventually.

Simplicial subdivision works by exploring the space of mixed-strategy profiles in the game. Specifically, it divides that space according to successively smaller triangular grids, i.e., dividing the simplex into smaller simplexes, while tracing a fixed path along the vertices of this grid so that the current vertex is adjacent to a simplex that contains a fixed point. SimpDiv is also a Las Vegas algorithm, because the starting point of the path determines how much time the exploration will take and which fixed point will be found.

2.3 Other Computational Mechanism Analysis

There is a small existing literature on general-purpose techniques for analysing mechanisms. This is in contrast with special-purpose techniques that only work for a specific small family of mechanisms, settings and equilibrium concepts. For example, [103] and [5] include special-purpose algorithms for finding Nash equilibria of internet-advertising auctions. However, both are tied to specific auctions, settings and equilibrium refinements.

Vorobeychik et al [108] have experimented with applying iterative best response to simple two-bidder auction problems. Iterative best response is a very straightforward algorithm: each agent initially plays some pure strategy, and in each iteration one agent changes to a pure strategy that is a best response given the current pure strategies of the other agents. Unfortunately, this algorithm can only find pure-strategy Nash equilibria. Further, even in games in which such equilibria exist, iterative best response can cycle forever without finding

³A Las Vegas algorithm is an algorithm that has random behavior and runtime, but that never produces incorrect output [6].

one.

Gerding et al [35, 86, 95] have explored refinements of the fictitious play algorithm applied to simultaneous auction games and double auctions. Fictitious play is similar to iterative best response; however, when players update their strategies, they best respond to the empirical distribution over previous iterations. In the limit, these empirical distributions can converge to mixed Nash equilibria. However, fictitious play is not guaranteed to converge.

Neither of these approaches, nor my approach, dominates the others in terms of expressiveness: Each can model games that are impossible to model for the other two. One advantage of using AGGs, apart from being able to model scenarios that are impossible for the other two, is that it allows for more sophisticated algorithms, i.e., ones that are guaranteed to converge to equilibrium on any input, and that can be targeted to search for specific types of equilibria.

Vorobeychik et al, have also studied interactions such as position auctions, approximating Nash equilibria by simulation [105–107]. In principle, a simulator can model an arbitrarily complicated game. However, in practice their approach involves sampling a small number of strategies (and a small number of random seeds capturing any randomness in the game) and then finding an equilibrium of a game defined by those samples. This approach is currently only practical when the number of players and samples is small. Unfortunately, a game defined by a small number of samples is unlikely to encode the full complexity of the simulator.

Chapter 3

The Support-Enumeration Method for Action-Graph Games

3.1 Introduction

In this chapter, we introduce a novel equilibrium-finding algorithm for AGGs, based on the support-enumeration method (SEM) of Porter, Nudelman and Shoham [84]. Similarly to the SimpDiv and GNM variants produced by Jiang *et al* [51], our SEM variant works by replacing the main inner-loop operations with faster AGG-specific alternatives. SEM has two distinguishing features compared to those other algorithms: 1) for many important families of games, SEM is much faster at finding sample Nash equilibria, and 2) SEM can enumerate Nash equilibria, rather than just finding a single sample equilibrium. (Further, this search can be directed towards specific types of Nash equilibria, particularly small-support Nash equilibria. No other algorithm can do this.) Although the first feature originally motivated us to develop SEM for AGGs, the second has proved even more valuable. As subsequent chapters will show, we often found that games of interest have many Nash equilibria, and that considering only the sample equilibria found by other solvers could be misleading. Thus, SEM’s equilibrium enumeration ultimately became the main game-solving algorithm for all our subsequent analyses.

A key reason that SEM has not been extended to work with compact game representations is that it operates very differently from other equilibrium-finding algorithms, and hence existing techniques could not be applied to it directly. In this chapter we show how SEM can be extended to work with AGGs (and hence with other game families compactly encodable as AGGs, such as graphical games and congestion games). Specifically, we show how three of SEM’s subroutines can be made exponentially faster. Experimentally, we show that these optimizations dramatically improve SEM’s performance, rendering it competitive with, and often faster than, other state-of-the-art algorithms for computing equilibria of AGGs.

$$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) = v_i \quad \forall i \in N, \forall a_i \in S_i \quad (3.1)$$

$$\sum_{a_{-i} \in S_{-i}} p(a_{-i}) u_i(a_i, a_{-i}) \leq v_i \quad \forall i \in N, \forall a_i \in A_i \setminus S_i \quad (3.2)$$

$$\sum_{a_i \in S_i} p_i(a_i) = 1 \quad \forall i \in N \quad (3.3)$$

$$p_i(a_i) \geq 0 \quad \forall i \in N, \forall a_i \in S_i \quad (3.4)$$

Figure 3.1: The Test-Given-Support (TGS) feasibility program for n -player games. For any given support profile $S \in \prod_{i \in N} 2^{A_i}$, we can construct a TGS feasibility program where any feasible solution (p, v) is a Nash equilibrium with support S ,¹ where the players randomize according to the probabilities in p and get the payoffs specified by v . The constraints on line 3.1 specify that each player is indifferent between all the actions in the support of his strategy. Those on line 3.2 specify that each player weakly prefers the actions in the support of his strategy. The remaining lines specify that each mixed strategy is a probability distribution. Note that $p(a_{-i}) = \prod_{j \in N \setminus \{i\}} p(a_j)$ because agents randomize independently in Nash equilibria.

3.2 Technical Background

The support-enumeration method (SEM) is a brute-force-search method for finding Nash equilibria. Rather than searching through all mixed strategy profiles, it searches through support profiles (specifying which actions each agent plays with positive probability) and tests whether there is a Nash equilibrium with that particular support. This test can be performed using the polynomial feasibility program given in Figure 3.1, which is feasible if and only if a Nash equilibrium with that support profile exists. Though several algorithms have been proposed for searching in the space of supports to find Nash equilibria [24, 63, 65], we will focus on the most recent SEM variant, due to Porter, Nudelman and Shoham [84]. This variant introduces two important features designed to improve empirical performance. First, it uses heuristics to order its exploration of the space of supports, searching from smallest to largest, breaking ties in favor of more balanced support profiles. This order can speed up equilibrium finding for several reasons. First, there are fewer small support-size profiles to search through. Second, the corresponding feasibility programs have fewer variables and simpler constraints. And third, in many games of interest (see, e.g., [80]), Nash equilibria with small, balanced supports are common [84]. The second important feature is that instead of simply iterating through the complete set of support profiles of a given size, SEM explores this space by tree search (see Figure 3.3 for

¹Note that this formulation allows for actions in the support to be played with zero probability. This does not adversely affect SEM's behavior when looking for a single Nash equilibrium; if such a Nash equilibrium existed, SEM would have found it already. However, this can have adverse affects when searching for multiple mixed Nash equilibria, e.g., on the game below:

	A	B
A	0,0	-1,0
B	0,-1	0,0

Although this game has no mixed Nash equilibria, TGS is feasible given full supports. This issue is easily addressed by adding an objective $\max x$ subject to $x \leq p_i(a_i)$ for all actions in the support. Although supports strictly smaller than S are still feasible, the optimal solution will have smaller support iff no MSNE with support S exists (which is indicated by x being equal to zero).

Input: $D = (D_1, \dots, D_n)$: profile of domains
Output: Updated domains, D , or failure

```

repeat
  changed  $\leftarrow$  false
  foreach  $i \in N$  do
    foreach  $a_i, a'_i \in \cup_{d_i \in D_i} d_i$  do
      if  $\text{ConditionallyDominated}(a_i, a'_i, D_{-i})$  then
         $D_i \leftarrow D_i \setminus \{d_i \in D_i : a_i \in d_i\}$ 
        changed  $\leftarrow$  true
        if  $D_i = \emptyset$  then
          return failure
until  $\neg \text{changed}$  ;
return  $D$ 

```

Figure 3.2: Iterative Removal of Strictly Dominated Strategies

an example). This search works by, at each level, selecting a support for a single additional player. At the leaves of the tree, the support is specified for every agent, and that support profile can be tested using TGS for the existence of a Nash equilibrium. (See Section 3.3.5 for an example of this.) The advantage of using this tree search comes from pruning: after an agent's support is selected, SEM performs iterative removal of strictly dominated strategies (IRSDS — see Algorithm 3.2), conditional on agents only playing actions in their supports. This has the effect of eliminating many support profiles from consideration. The search backtracks whenever an agent has an empty support or the TGS feasibility program is infeasible.

While SEM can find (or enumerate) exact PSNEs, but there are two important caveats when it comes to mixed-strategy Nash equilibria. The first is that mixed-strategy Nash equilibria can involve irrational probabilities, making them impossible to store in any rational representation. Thus, the best any implementation working with rational or floating-point representations can guarantee is to find ε -Nash equilibria. (For our experiments, we used an additive ε of 10^{-10} .) Second, games can have multiple, even infinitely many, Nash equilibria for a single support profile; enumerating all MSNEs requires enumerating all feasible solutions to the TGS system. For our enumeration experiments, we restricted our attention to PSNEs, which are finite in number and can be represented exactly using rational numbers.

3.3 SEM for AGGs

Observe that we can trivially make a version of SEM that takes an AGG as input, simply replacing the normal form game (NFG) utility lookups ($u_i(a)$) with the AGG equivalents ($u_i(a) = u_{a_i}(c^{(a_i)})$), where $c^{(a_i)}$ is the projected configuration given a). We denote this algorithm NFG-SEM, because its behavior is exactly the same as that of SEM for strategically equivalent normal-form games. However, while SEM and NFG-

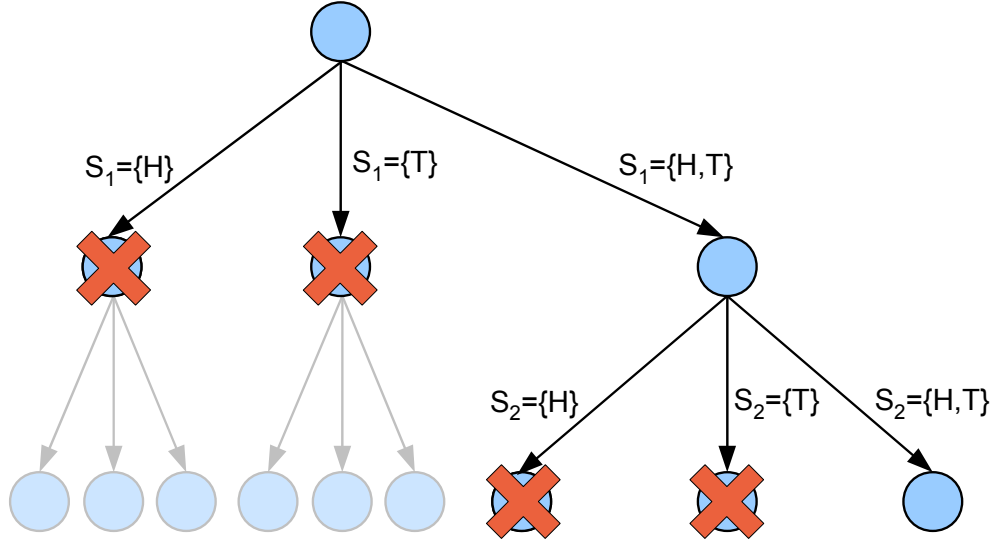


Figure 3.3: Porter *et al.*'s [84] tree-search works by instantiating strategies from agents' supports, one agent at a time, and removing strategies that are strictly dominated conditional on the agents playing within their given supports. The search backtracks whenever an agent has an empty support or the TGS feasibility program is infeasible.

SEM will do the same sequence of operations, their asymptotic performance analyses are not equivalent. This is because NFG-SEM's inputs can be exponentially smaller, due to the representational efficiency of AGGs. Specifically, we show that NFG-SEM's inner-loop operations—iterative removal of strictly dominated strategies and the TGS feasibility program—are at least worst-case exponential in the AGG input length (denoted ℓ).² The outer-loop search over supports also requires exponential time, even for games with PSNEs.

However, we can do better if we construct a version of SEM that explicitly takes AGG structure into account. We present such an extension of SEM, denoted AGG-SEM, and its asymptotic analysis. Overall, we show that AGG-SEM's worst-case performance is exponentially faster than that of NFG-SEM.³

3.3.1 Conditional Dominance

Because SEM makes extensive use of iterative removal of strictly dominated strategies, efficiently identifying dominated strategies is critical. For normal-form games, testing whether or not some pure strategy a_i is

²AGG's representation sizes are usually dominated by the size of their payoff tables. Intuitively, one might conclude that each table has size $O(n^d)$ (where d is the in-degree of a the corresponding action node). However, this bound is often extremely loose for AGGs, and does not apply to AGG-FNs, at all. Thus, it is both simpler and more accurate to express asymptotic performance in terms of actually input size, ℓ , rather than in terms of properties of the graph such as in-degree.

³Note that our theoretical results only show that AGG-SEM has exponentially faster worst-case runtime than NFG-SEM. There may exist other SEM-based algorithms that are as fast (or faster than) AGG-SEM.

dominated by some other a'_i is straightforward: one can exhaustively search through the pure strategy profiles of the other agents, looking for the existence of some a_{-i} to which a_i is a weakly better response. This trivial algorithm only requires time linear in the size of a normal-form game. However, it can require time exponential in the size of an action-graph game.

Lemma 1. *NFG-SEM's dominance check has a worst-case running time of $\Theta(2^\ell)$.*

Proof. Consider the family of action-graph games with two actions per player and no edges. For this family, there are at most $2n$ nodes, the payoff table for each of which contains only a single value. Thus, ℓ is $\Theta(n)$, while $|A_{-i}| = 2^{n-1}$ and therefore is $\Theta(2^\ell)$. In the worst case (where an agent has a dominated strategy), exhaustive search iterates over every $a_{-i} \in A_{-i}$ to confirm that a_i is not a best response to any action profile. \square

However, we can do better: a straightforward, polynomial-time algorithm for AGG dominance checking can be derived using an idea similar to that of Jiang *et al*'s [51] dynamic-programming algorithm. To determine whether or not a_i is dominated by a'_i , we do not need to search through the entirety of A_{-i} ; we only need to search over the set of possible projected configurations on the joint neighborhoods of a_i and a'_i . This adaptation guarantees polynomial runtime. However, empirically we observed that it often gave rise to poor performance, compared to exhaustive search over A_{-i} . We attribute this to stopping conditions: the exhaustive search can stop as soon as it finds any case where a_i is a better response, while the dynamic-programming algorithm must build up all configurations first before it ever encounters a better response, effectively performing a breadth-first search. Based on this insight, we created a depth-first-tree-search-based algorithm that combines the best of both approaches: like exhaustive search, it can find a better response without needing to compute the entire set of projected configurations; like our adaptation above, it exploits AGG structure and so needs only to evaluate a polynomial number of projected configurations. It works as follows. At each level, the search fixes the action of some agent, giving a search tree that potentially includes every A_{-i} . (Recall that m denotes the maximum number of possible actions for any agent.) However, we also perform multiple-path pruning: a search refinement in which previously visited nodes are recorded, and the search backtracks whenever it re-encounters a node along a different search path [83]. In our case, the algorithm backtracks whenever it encounters a previously visited projected configuration, based on a lookup from a trie map. (See Algorithm 3.4 and see Section 3.3.5 for an example the algorithm's execution.)

Lemma 2. *AGG-SEM's dominance check has a worst-case running time of $O(nm\zeta^3)$.*

Proof. The search traverses a tree with a depth of n and a branching factor of m . However, at every level, at most ζ^2 nodes are expanded (where ζ denotes the largest set of possible projected configurations for any node), because there are at most ζ^2 distinct projected configurations on the neighborhood of a_i, a'_i . In the worst case, when it traverses the whole tree, the search must follow each of m arcs from $O(\zeta^2)$ nodes at each of n levels, or $O(nm\zeta^2)$ arcs. For each arc, the search may perform a trie-map lookup and insert; these operations each require runtime that grows like the maximum in-degree of the graph, t , and so the total cost

is $O(nm\zeta^2\iota)$. Because ℓ is $\Omega(\zeta + \iota)$, $nm\zeta^2\iota$ is $O(nm\ell^3)$. □

Input: Two actions for player i : a_i, a'_i
A domain for each of the other players: D_{-i}
Output: True iff a'_i dominates a_i conditional on $-i$ only playing actions from D_{-i}
 $T \leftarrow$ an empty trie
 $c \leftarrow$ an empty projected configuration on neighbors of a_i, a'_i
return *RecursiveCD*($a_i, a'_i, D_{-i}, T, c, 1$)

Figure 3.4: ConditionallyDominatedAGG

Input: Two actions for player i : a_i, a'_i
A domain for each of the other players: D_{-i}
A trie map T (passed by reference)
A projected configuration c on the neighbors of a_i and a'_i
A player j
Output: True iff a'_i dominates a_i conditional on $-i$ only playing actions from D_{-i}
if $j = i$ **then**
 return *RecursiveCD*($a_i, a'_i, D_{-i}, T, c, j + 1$)
if $j = n + 1$ **then**
 if $u_i(a_i, c) \geq u_i(a'_i, c)$ **then**
 return *false*
 return *true*
foreach $a_j \in \{a_j \in A_j : \exists s_j \in D_j \text{ s.t. } a_j \in s_j\}$ **do**
 $c' \leftarrow c \oplus a_j$
 if $\neg(j, c') \in T$ **then**
 $T \leftarrow T \cup \{(j, c')\}$
 if $\neg \text{RecursiveCD}(a_i, a'_i, D_{-i}, T, c', j + 1)$ **then**
 return *false*
return *true*

Figure 3.5: RecursiveCD

3.3.2 TGS Feasibility Program

SEM's asymptotic performance is dominated by the Test-Given-Support feasibility program. (Polynomial feasibility is NP-hard; e.g., polynomial constraints generalize 0–1 integrality constraints [104].) For NFG-

SEM, this complexity obstacle is particularly severe: directly representing the TGS feasibility program requires space exponential in the size of the AGG. Thus, TGS could require doubly exponential time.

Lemma 3. *The NFG-SEM TGS feasibility program has worst-case size of $\Theta(nm2^\ell)$.*

Proof sketch, similar to Lemma 1's proof. TGS can have $|A_{-i}|$ terms in each constraint, and this quantity is $\Theta(2^\ell)$ in the worst case. There are $O(nm)$ such constraints. \square

The essential challenge is in the expected utility constraints (lines 1 and 2 of Figure 3.1) which can be exponentially long. We already have Jiang et al's [51] dynamic-programming algorithm for computing expected utility given a specific mixed-strategy profile. Now we want to compute expected utility symbolically, without specifying the probabilities beforehand. This can be accomplished by “unrolling” the dynamic program: every update in the dynamic program is expressed as a polynomial equality constraint in the TGS program. This set of new constraints is polynomial in the size of the AGG.

Lemma 4. *The AGG-SEM TGS feasibility program has worst-case size of $O(n^2m^2\ell^2)$.*

Proof. For each $j \in \{1, \dots, n\}$ we introduce $O(\zeta)$ new constraints, each corresponding to the probability of a projected configuration given the strategies of the first j agents. This gives $O(n\zeta)$ constraints. Each contains at most $O(\zeta m)$ terms, corresponding to the possible projected configurations and actions that could lead to some new projected configuration when another agent is added. Since ζ is $O(\ell)$, the output requires $O(nm\ell^2)$ space. It must be run once for each agent i and for each action in A_i : $O(nm)$ times in total. Thus, the TGS feasibility program requires $O(n^2m^2\ell^2)$ space. \square

Although this optimization speeds up the worst case exponentially, it is not guaranteed to be helpful on average. This is because the symbolic representation of the TGS system is made exponentially smaller by replacing each exponentially long expected-utility constraint with multiple small constraints. How this change affects runtime in the average case depends on the (black-box) feasibility solver.

3.3.3 Asymptotic Analysis of SEM for AGGs

We are now ready to compare the overall asymptotic runtime of AGG-SEM and NFG-SEM. We assume that both algorithms make use of a polynomial feasibility solver with worst-case runtime $O(2^x)$ where x is the length of the feasibility program. Unfortunately, without knowing which instances are hard for a given polynomial feasibility solver, we cannot know if these inputs produced by either SEM variant will actually require that much time. Thus, we can only upper-bound the runtime of each algorithm.

Theorem 5. *NFG-SEM's worst-case runtime requires the solving of $O(2^\ell)$ feasibility programs of size $O(nm2^\ell)$. Thus, NFG-SEM's worst-case runtime is bounded above by $O(2^{nm2^\ell + \ell})$, assuming a feasibility solver with worst-case runtime of $O(2^x)$ where x is the length of the feasibility program.*

Proof sketch. In the worst case, AGGs can have $\Omega(2^\ell)$ support profiles (as in Lemma 1), even for symmetric

games. Thus, the search must traverse a tree with $O(2^\ell)$ leaf nodes where TGS is solved, and $O(2^\ell)$ interior nodes where iterative removal of strictly dominated strategies (IRSDS) is performed. Each TGS system requires $O(nm2^\ell)$ space by Lemma 3. The time to solve each one is bounded above by $O(2^{nm2^\ell})$, which dominates the time required by IRSDS. Thus the total runtime bounded by $O(2^{nm2^\ell+\ell})$. \square

Theorem 6. *AGG-SEM's worst-case runtime requires the solving of $O(2^\ell)$ feasibility programs of size $O(n^2m^2\ell^2)$. Thus, AGG-SEM's worst-case runtime is bounded above by $O(2^{n^2m^2\ell^2+\ell})$, assuming a feasibility solver with worst-case runtime of $O(2^x)$ where x is the length of the feasibility program.*

Proof sketch. AGG-SEM still searches $O(2^\ell)$ interior nodes and leaf nodes. At each leaf node, a TGS system requiring $O(n^2m^2\ell^2)$ space (by Lemma 4) must be solved. The time to solve each one is bounded above by $O(2^{n^2m^2\ell^2})$, which dominates the time required by IRSDS. Thus the total runtime bounded by $O(2^{n^2m^2\ell^2+\ell})$. \square

Given complexity results known in the literature, it is unsurprising that AGG-SEM requires exponential time in the worst case. In particular, finding even PSNEs of AGGs in polynomial time would imply $P=NP$. The reasoning behind this observation is that AGG-SEM always returns a PSNE, if one exists. Thus, if AGG-SEM could find a sample Nash equilibrium in polynomial time, it could be used to determine whether a given AGG has any PSNEs in polynomial time. AGGs generalize graphical games (and have the same asymptotic size), and finding a PSNE of an arbitrary graphical game is NP-hard [42]. Further, finding a PSNE of a symmetric AGG with unbounded m is NP-hard [19].

3.3.4 Further Speedups for k -Symmetric Games

We now show that the search over supports can be sped up in the case of AGGs with k -symmetry, i.e., where the players can be partitioned into k classes such that all players in a class are identical. (We describe the algorithm for the case of $k = 1$, or full symmetry. The generalization is straightforward.) We saw in the proof of Theorem 5 that symmetry does not help for NFG-SEM. Here we strengthen that result, showing that NFG-SEM can take exponential time even when PSNEs exist in k -symmetric AGGs with bounded m and k .

Theorem 7. *NFG-SEM requires exponential time to find a sample PSNE in a k -symmetric AGG with bounded m and k .*

Proof sketch. For games with PSNEs, we never need to solve TGS: any support profile that survives IRSDS is a Nash equilibrium. The tree search must still expand $O(2^\ell)$ nodes to explore all $O(2^\ell)$ pure support profiles. At each interior node, IRSDS is called, requiring $O(n^2m^3)$ calls to the conditional dominance test, which requires $O(2^\ell)$ time (by Lemma 1). For bounded m , the total runtime to find a PSNE is $O(n^22^{2\ell})$. Further, recall from Lemma 1 that NFG-SEM's conditional dominance test requires $\Theta(2^\ell)$ when the strategy is dominated. Thus, NFG-SEM's worst-case run-time is $\Omega(2^\ell)$. \square

Next, we show that we *can* achieve an improvement on such games for AGG-SEM. This optimization works

by skipping any support profile that is a permutation of a previously explored support profile. At every stage of the tree search, we explore a support S_i iff $S_i \succeq S_j$ where j is any player with support selected higher in the tree, and where \succeq is the order in which supports are explored at each level of the tree.

Lemma 8. *AGG-SEM's search evaluates $\text{poly}(n)$ support profiles in the worst case, even for games without PSNEs, given a k -symmetric AGG with bounded k and m .*

Proof. Every distinct support profile can be identified by a vector of $O(k2^m)$ integers in the range $[0, n]$, where each element indicates how many agents of a given class have a given support. There are at most $O(n^{k2^m})$ such vectors. For bounded k and m , this quantity is $\text{poly}(n)$. \square

Theorem 9. *AGG-SEM requires $\text{poly}(\ell)$ time to either*

1. *find a sample PSNE, or*
2. *determine that no PSNE exists*

in a k -symmetric AGG with bounded m and k .

Proof sketch. For bounded m and k , AGG-SEM's search expands polynomially many nodes (by Lemma 8), each of which requires running IRSDS. IRSDS performs $O(n^2 m^3)$ conditional dominance tests, requiring $O(nm\ell^3)$ time (by Lemma 2). Thus, AGG-SEM has $\text{poly}(\ell)$ runtime on such games. \square

3.3.5 Examples

In this section, we show two examples of key operations of AGG-SEM.

Outer-Loop Search Example

First, we show a example of AGG-SEM enumerating PSNEs, for the following simple symmetric game (with PSNEs entries marked in bold):

	A	B	C
A	1,1	1,0	1,0
B	0,1	2,2	3,3
C	0,1	3,3	2,2

AGG-SEM's enumeration of supports proceeds in a depth-first fashion. Each arc represents choosing the support of a single agent. (See Figure 3.6.) For PSNEs, only supports of size one are considered. First, the algorithm selects that agent one plays action A (node 2), then IRSDS is performed, which removes B and C

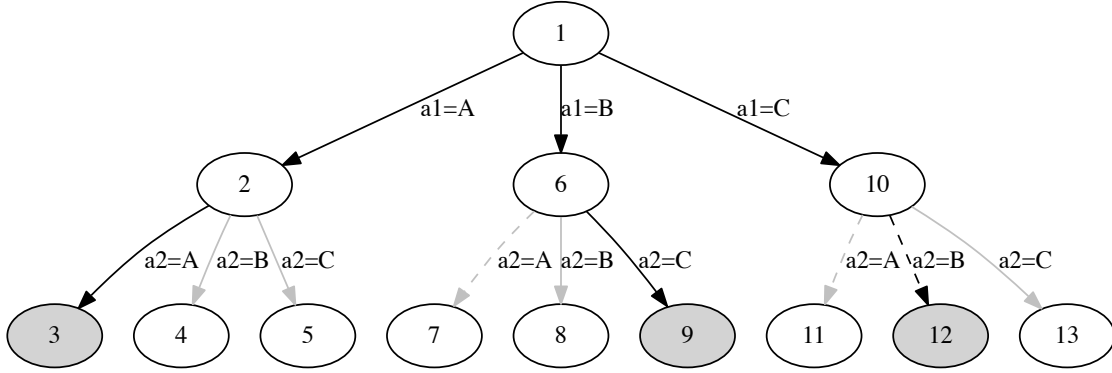


Figure 3.6: Visualizing the outer-loop tree search process followed by AGG-SEM when enumerating PSNEs. Each edge specifies that one more agent’s support has been determined. Edges that can be pruned by IRSDS are marked in gray, while edges that can be pruned by symmetry are marked by dashes. Shaded (leaf) nodes correspond to PSNEs. Note that AGG-SEM does not need to visit node 12, because this is just a permutation of the equilibrium identified at node 9.

from the space of possible actions for agent two. The algorithm proceeds to node 3, and identifies a PSNE, (A, A) . Because the other children of node 2 have been pruned by IRSDS, the algorithm backtracks and goes to node 6. Here, IRSDS can prune both A and B from the space of possible actions. $a_2 = A$ will also not be explored because this is a permutation of a profile that was previously explored or ruled out. The algorithm proceeds to node 9, another PSNE, (B, C) . Then the algorithm backtracks and proceeds to node 10. However, all the children of node 10 can be pruned: A and C are dominated given that $a_1 = C$, and profile (C, B) can be pruned by symmetry. With the space of pure-strategy profiles exhausted, the algorithm ends.

Conditional Dominance

Next we show a worked example of conditional dominance testing in a simple 3-agent, 4-resource simple congestion game with resources denoted by (A, B, C, D) , where each agent can only choose a single resource, and where the payoff for using resource x is $6/c_x$ where c_x denotes the number of agents using that resource. Note that this corresponds to a symmetric AGG where there are 4 nodes, and the only edges are self-edges.

ConditionallyDominatedAGG works by searching for partial strategy profile a_{-i} where a_i is a weak best response, which would prove that a'_i does not strictly dominate a_i , or by confirming that no such profile exists (meaning that a_i is strictly dominated and can be pruned without any risk of failing to find a Nash equilibrium).

This search proceeds in a depth-first fashion, fixing the actions of all the other agents, one at a time. (See

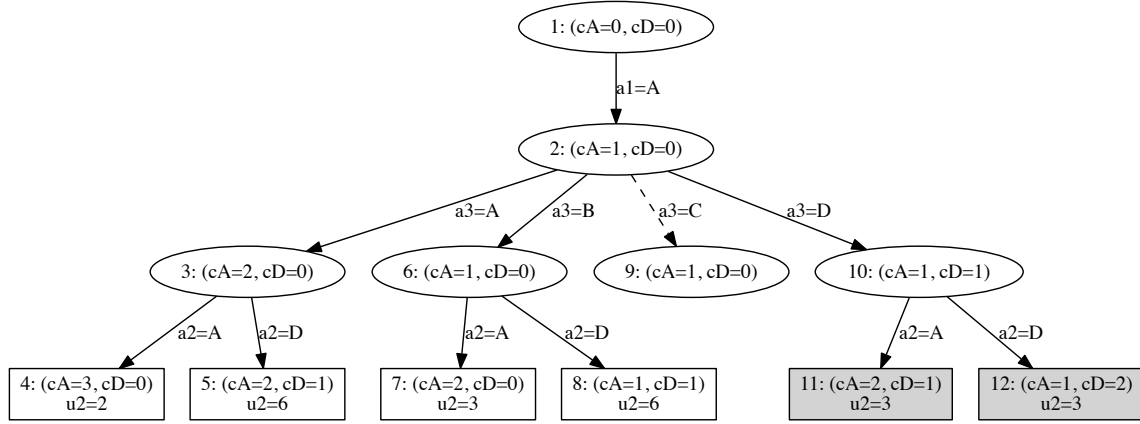


Figure 3.7: Visualizing a ConditionallyDominatedAGG search. Each edge is labeled with an assignment of an action to an agent. Each node is labeled with an integer identifying the order in which ConditionallyDominatedAGG explores, and with a 2-tuple, identifying the projected configuration given the actions leading to that node. Leaf nodes are square and come in pairs, representing actions a_i and a'_i , and are also labeled with agent i 's utility. If i 's utility is weakly greater for action a_i (as in the shaded nodes), then a_i is not dominated by a'_i and the search can exit successfully.

Figure 3.7.) At node 2, agent 1 plays A. This increases the projected configuration on A, denoted c_A to 1. At node 3, agent 3 does the same, increasing c_A to 2. Nodes 4 and 5 correspond to agent 2's choice of A and D respectively, which increases c_A and c_D respectively. Because D has strictly higher utility for agent 2, the algorithm cannot yet confirm that D does not dominate A. Backtracking, to node 6, we consider the case where agent 3 plays B. Because B is not a neighbor to either of the actions under consideration, the projected confirmation does not change between node 2 and node 6. As in nodes 4 and 5, agent 2 has a higher payoff for D in nodes 7 and 8. Thus, the search backtracks to node 9. Because node 9 has the same projected configuration as node 6, it can be pruned and the search backtracks to nodes 10 (where agent 3 plays D) and it's children, nodes 11 and 12. In this case (where agents 1 and 3 play A and D respectively), agent 2 gets equal utility for A and for D, showing that D does not strictly dominate A.

3.4 Experimental Evaluation

So far, our analysis has concentrated on the worst case. However, improvements to the worst case do not necessarily imply improvements on instances of interest. As we are motivated by developing *practical* methods for computing Nash equilibria, we conducted an experimental evaluation to compare the performance of NFG-SEM and AGG-SEM.

3.4.1 Experimental Setup

We sampled 50 instances from each of 11 different game distributions. Nine distributions were from GAMUT [80]; each had $n = 10$ players, $m = 10$ actions per player, and action graphs with in-degree at most five. The remaining two distributions were over position auction games with $n = 10$ players and up to $m = 11$ actions per player (though weakly dominated actions, which occurred frequently, were omitted by the generator). The two position auctions distributions based on 1) the generalized first-price auction (GFP), which is often has no pure-strategy Nash equilibrium, and 2) the weighted generalized second-price auction (GSP), which often has many pure-strategy Nash equilibrium [98]. (See Table 3.1.)

On each game, we compared AGG-SEM to three other algorithms: NFG-SEM, and the two existing state-of-the-art Nash-equilibrium-finding algorithms: GNM, the global Newton method [43], and SimpDiv, simplicial subdivision [102], both using Gambit implementations [68] extended to work efficiently with AGGs by Jiang *et al*[51]. All algorithms were given error tolerance of 10^{-10} . For AGG-SEM and NFG-SEM, we used MINOS [74] to solve the TGS feasibility problems.

We performed a blocking mean-of-means test [15] (with $p \leq 0.05$) to compare mean runtimes across game distributions. This is a blocking test (i.e., rather than testing whether the mean runtime of algorithm A was significantly less than the mean runtime of algorithm B, we tested whether zero was significantly greater than the mean difference between the runtimes of algorithms A and B), to account for the fact that some instances may be harder than others, e.g., because they are lacking in small-support equilibria. The bootstrapping process for testing these differences was run with 20,000 samples per test. Because the same instances were used in multiple tests (e.g., comparing algorithm A to algorithm B and to algorithm C), we applied a Bonferroni correction.

In three distributions (see Table 3.1), we were not able to conclude that differences were significant because of high runtime variation. Such problems can be overcome by obtaining additional data; thus, for these distributions we generated an additional 150 instances (i.e., 200 total). We included the resulting additional tests in our Bonferroni correction. In the end, we were able to identify a significantly faster algorithm for every distribution, except for D1, the coffee-shop game.

Our experiments were performed on machines with dual Intel Xeon 3.2GHz CPUs, 2MB cache and 2GB RAM, running Suse Linux 11.1 (Linux kernel 2.6.27.48-0.3-pae). Each run was limited to 12 CPU hours; we report runs that did not complete as having taken 12 hours. In total, our experiments required about 420 CPU days.

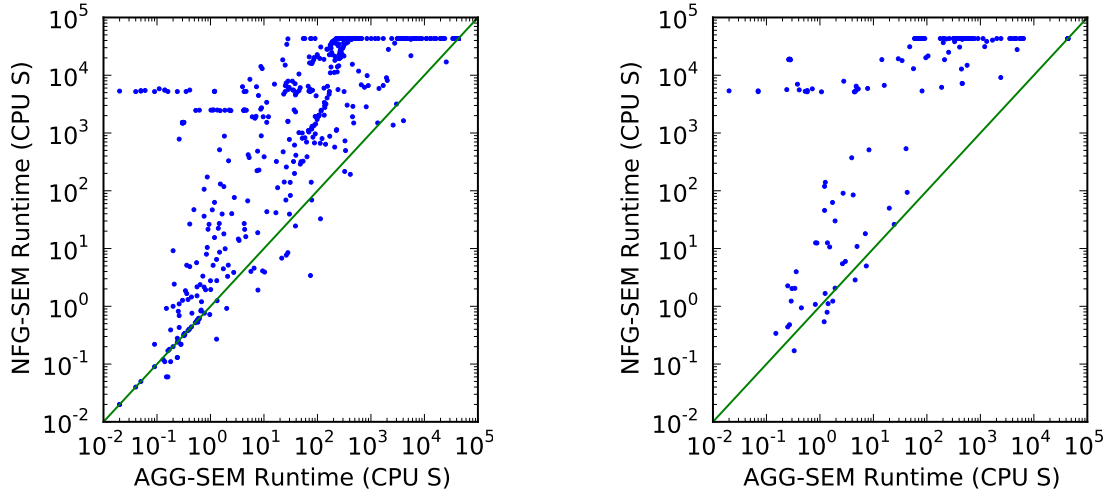


Figure 3.8: Scatterplot contrasting the runtimes of AGG-SEM and NFG-SEM. The left plot shows runtimes for computing a sample Nash equilibria; the right plot shows runtimes for computing all pure strategy Nash equilibria.

3.4.2 Results

Overall, we found that AGG-SEM provided a substantial performance improvement over NFG-SEM, outperforming it on the vast majority of instances. (See Figure 3.8). As we expected, AGG-SEM was not faster on absolutely every instance. Nevertheless, AGG-SEM achieved significantly faster mean performance in every game distribution. Its largest speedup over NFG-SEM was $280\times$ (on D1 — Figure 3.12a), its smallest speedup was $1.45\times$ (on D4 — Figure 3.12d), and its median speedup was $10\times$ (on D10 — Figure 3.12j). While the biggest speedups were on games where AGG-SEM could leverage k -symmetry, ranging from $280\times$ (on D1 — Figure 3.12a) to $7\times$ (on D2 — Figure 3.12b), we still achieved substantial speedups on asymmetric games (those with n player classes), ranging from $10\times$ (on D10 — Figure 3.12j) to $1.45\times$ (on D4 — Figure 3.12d). AGG-SEM stochastically dominated NFG-SEM overall (see Figure 3.11), but on a per-distribution basis, it only stochastically dominated on four distributions (D3 and D9–11 — Figure 3.12c and i–k). AGG-SEM’s failure to stochastically dominate on the remaining seven distributions was due to the fact that all contained instances that both methods solved extremely quickly (in less than one second), but that NFG-SEM finished more quickly. Considering runtimes over a second, AGG-SEM stochastically dominated NFG-SEM on every distribution.

AGG-SEM significantly outperformed SimpDiv and GNM on 8 of the 11 game distributions (see Table 3.1), and furthermore stochastically dominated GNM overall (see Figure 3.11). Comparing the runtimes of AGG-SEM and SimpDiv, we found that the two were correlated: they both solved many (338) of the same instances in under 600s, with SimpDiv having better mean runtime on these instances ($\mu = 12.71s$ vs $\mu = 108.32s$). On the remaining instances, SEM finished far more often (87.74% vs 23.11%). Thus, SimpDiv was only fastest

on D2 (Ice-cream games — Figure 3.12b), which contained almost exclusively instances that were easy for both algorithms. AGG-SEM only stochastically dominated SimpDiv on D10 and D11 (Figure 3.12j and k), which contained no instances that were easy for SimpDiv. AGG-SEM’s runtime was less correlated with that of GNM than it was with that of SimpDiv. For example, GNM solved every instance in D4 (GFP position auctions — Figure 3.12d), which contained instances that were not solved by either SimpDiv or AGG-SEM. Overall, however, GNM had the worst performance (and was stochastically dominated by AGG-SEM on every distribution but D3 and D4). Although AGG-SEM had the fastest overall average runtime, there were at least a few instances for which each of AGG-SEM, SimpDiv and GNM was hundreds of times faster than the others. The best practical approach may thus be a portfolio of all three algorithms, following e.g. [111].

Like Porter, *et. al.*, [84], we found that in many (7 of 11) distributions, most games (over 90%) had PSNEs. AGG-SEM finished on every such game. Thus the four distributions (D4 and D9–11 – Figure 3.12d and i–k) in which PSNEs were least common were also those on which AGG-SEM was mostly likely to time out. (We verified that AGG-SEM terminated on every game with PSNEs by checking the support size AGG-SEM was considering when it timed out. In every case, it ruled out all pure support profiles before running out of time.)

One advantage of SEM over other Nash-equilibrium-finding algorithms is its ability to enumerate all Nash equilibria (or all equilibria with support sizes not more than some constant). This is particularly useful when we want to understand the range of possible outcomes. For example, in [98] one of the goals was to identify the minimum and maximum revenue possible in equilibrium of position auction games. At the time, we were only able to compute empirical bounds on revenue in any given auction game (e.g., asserting the best-case equilibrium revenue must be at least \$5 because we found an equilibrium where the revenue is \$5). Tools that could search through or enumerate the full set of equilibria (e.g., to find the equilibrium that provably maximizes revenue) did not exist at the time. We have since tested equilibrium enumeration by searching for all PSNEs on a representative subset of these games (20 from each distribution). We found that AGG-SEM was significantly faster than NFG-SEM (see Figure 3.8) at enumerating the set of all PSNEs. Notably, for every position auction game, AGG-SEM was able to find all PSNEs in under one CPU minute. (These games each have ten bidders, eight positions and eleven bid increments per bidder.)

3.5 The Hardness of Generalizing AGG-SEM

In the previous sections, we have focused on AGGs with function nodes (or AGG-FNs). We introduced the AGG-SEM algorithm for AGG-FNs, and demonstrated its advantages both theoretically and experimentally. However, this algorithm depends on solving many conditional-dominance problems as a key inner-loop step. In this section, we show that conditional dominance is computationally hard for two key extensions of AGGs: AGGs with function nodes and additivity (or AGG-FNAs) and Bayesian AGGs (BAGGs, with or without function nodes). Thus, SEM (and other algorithms depending on dominance checking) may not be feasible for such generalized AGGs.

No.	Game type	Player Classes	Mean runtime (CPU s)			
			AGG-SEM	NFG-SEM	GNM	SimpDiv
D1	Coffee shop	1	18.00	5032.38*	3309.73*	362.63 [†]
D2	Ice cream	3	131.64*	957.42*	151.59*	0.39
D3	Job market	1	249.02	6070.61*	372.96* [†]	1536.45* [†]
<i>Position Auctions:</i>						
D4	GFP	n	7519.90*	10878.19*	75.73	10750.93*
D5	Weighted GSP	n	45.10	96.78*	723.19*	734.56* [†]
<i>Random AGGs:</i>						
D6	Random graph	1	68.02	7005.34*	10580.58*	5188.02*
D7	Road graph	1	441.11	32103.15*	41814.79*	9507.58*
D8	Small-world graph	1	596.75	31750.79*	28195.09*	4665.58*
<i>Random Graphical Games:</i>						
D9	Random graph	n	11953.48	20469.50*	24337.47*	27002.81*
D10	Road graph	n	3244.50	32052.36*	43200.00*	43200.00*
D11	Small-world graph	n	11356.47	29861.96*	43200.00*	40677.67*
<i>Overall:</i>			3244.28	16520.09*	18265.93*	13176.94*

Table 3.1: Mean runtimes. * denotes significantly slower than fastest solver, $\alpha = 0.05$. Capped runs count as 43200s. [†] denotes distributions where, due to high variance, more instances (200) were necessary for statistical significance.

Formally, an AGG-FNA is similar to an AGG-FN, except for the space of pure strategies: in an AGG-FNA, each agent chooses a pure strategy from a list of subsets of action nodes. His payoff of for a given pure strategy is simply the sum of the payoff values for all the action nodes in his selection.

We define the decision problem as follows: the input is an AGG-FNA, an agent i , and a pure strategy s_i for that agent; the decision property is that s_i is not strictly dominated by any pure strategy.

Theorem 10. $\text{UNDOMINANCE}(\text{AGG-FNA})$ is NP-complete.

Proof. Part 1: $\text{UNDOMINANCE}(\text{AGG-FNA})$ is in NP: A certificate for this problem is a pure-strategy profile for all the agents except i . A certificate can be checked by simply enumerating all of i 's pure strategies, computing the expected utility for each. The certificate is valid iff s_i 's utility is weakly maximal.

Part 2: $\text{UNDOMINANCE}(\text{AGG-FNA})$ is NP-hard: by reduction from 3SAT In 3SAT, the input a set of k clauses of length at most three, and involving m boolean variables denoted x_1, \dots, x_m . The decision property is that there exists a satisfying assignment, i.e., an assignment to all m variables where every clause is satisfied.

This reduction creates an $m + 1$ player AGG-FNA (as in Figure 3.9) where player i (for $i \leq m$) corresponds to variable x_i . Each of these players has two pure strategies, corresponding to the possible assignments of his variable. Each of these pure strategies corresponds to a single action node, not shared with any other player. The payoffs of these players are not important, and can therefore be constant (meaning the corresponding action nodes have no in-arcs). Player $m + 1$ also has two pure strategies, which we call SAFE and SAT. SAFE consists of a single action node with no in-arcs; player $m + 1$ gets a payoff of $k - 1/2$ whenever he plays SAFE. SAT consists of k action nodes, corresponding to the k clauses in the 3SAT problem instance. Each of these action nodes at most three in-arcs corresponding to single-variable assignments that could satisfy the clause. The payoff for each of these action nodes is one if the assignment would satisfy the clause and zero

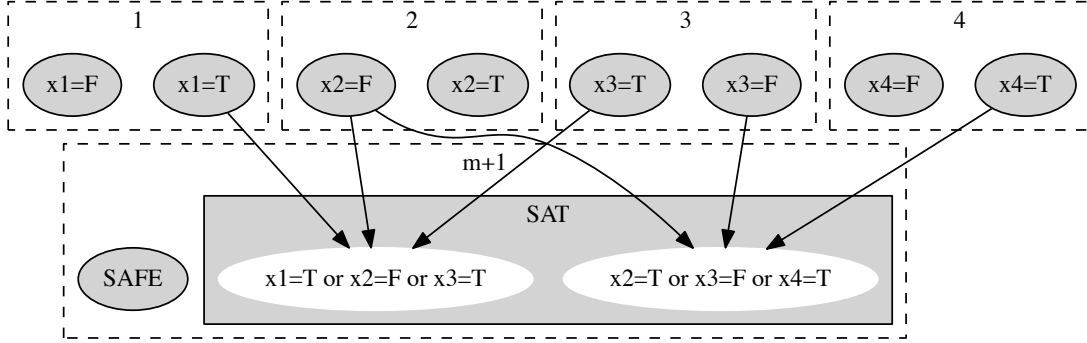


Figure 3.9: An example 3SAT instance reduced to dominance in an AGG-FNA. Dashed boxes denote action sets for agents, ellipses denote action nodes and shaded regions denote pure-strategies. Every pure strategy except SAT has a single action node; the payoff of SAT is the sum of the payoffs of its two action nodes.

otherwise.

Sub-claim (Testing for undominance can be used to test for satisfiability): There is a one-to-one mapping between assignments in the 3SAT problem instance and pure-strategy profiles for the first m players. Given such a pure-strategy profile, the payoff for player $m + 1$ playing SAT is equal to the number of clauses satisfied by the assignment. Thus, player $m + 1$ cannot get a payoff of k unless an assignment exists which satisfies all k clauses. Thus, SAT is dominated by SAFE (with a guaranteed payoff of $k - 1/2$) unless the 3SAT problem instance is satisfiable.

Sub-claim (The reduction is polynomial in time and size): This game has $2m$ action nodes for the variable players, and $k + 1$ action nodes for the SAFE/SAT player. Each variable-assignment action node (and the SAT node) have no in-arcs, and thus their payoff tables each have only a single entry. Each of the k clause action nodes has 3 in-arcs, which can only be zero or one (because action node is shared between any two agents). Thus, each of those action nodes has a payoff table with 2^3 entries. Each action node only appears in a single pure-strategy. The entire AGG-FNA can be created in a single pass (iterating through variables then through clauses, creating the relevant parts of the AGG-FNA as they arise). Therefore, the entire reduction can be accomplished in $O(m + k)$ time and space.

Therefore, $\text{UNDOMINANCE}(\text{AGG-FNA})$ is NP-hard. □

A similar proof can be used with Bayesian action-graph games (BAGGs).

Bayesian action-graph games are a combination of action-graph games and epistemic-type Bayesian games.

As in Bayesian games, each agent has a private epistemic type drawn from a discrete distribution with a commonly known prior. (Types need not be probabilistically independent.) Each type maps to subset of action nodes that the agent may choose from. BAGGs can also include function nodes and additivity, but these features are not necessary for our proof of hardness.

We define the $\text{UNDOMINANCE}(\text{BAGG})$ as follows: the input is an BAGG, an agent i , a type t_i for that agent and an action node a_i for that agent given that type; the decision property is that a_i is not *ex interim* strictly dominated by any pure strategy.

Theorem 11. $\text{UNDOMINANCE}(\text{BAGG})$ is NP-complete.

Our proof of NP-completeness for BAGGs follows the same structure as previous AGG-FNA proof. In particular, the hardness reduction also involves a subset of the players whose strategies correspond to certificates and player who has choose between a safe action and an action that corresponds to the decision property (which will be strictly dominated when the decision property is false).

Proof. Part 1: $\text{UNDOMINANCE}(\text{BAGG})$ is in NP: A certificate for this problem is a behavioral-strategy profile for all the agents except i . A certificate can be checked by simply enumerating all of i 's possible actions, computing the *ex interim* expected utility for each. The certificate is valid iff a_i 's utility is weakly maximal.

Part 2: $\text{UNDOMINANCE}(\text{BAGG})$ is NP-hard, by reduction from INDEPENDENTSET : In INDEPENDENTSET , the input is an undirected graph with n vertices, and a non-negative integer k . The decision property is that there exists an independent set (i.e., a set of nodes where no two nodes in that set share an edge) with cardinality k .

This reduction creates a 3 player BAGG. (See Figure 3.10.) The first two players each have n types (with uniform probability) each of which corresponds to one of the nodes v_i in the graph. Given each of these types, an agent has a choice of two action nodes, labeled IN and OUT. Again, the payoffs of these players are irrelevant, so these action nodes have no inputs. The third player has only one type, with two action-nodes, SAFE and INDSET. If he plays SAFE, he gets a constant utility of $k - 1/2$. If he plays INDSET his *ex post* utility is given by the table below:

case	t_1 and t_2	a_1 and a_2	u_3
i	equal	both IN	n^2
ii	equal	both OUT	0
iii	equal	otherwise	$-n^3$
iv	neighbours	both IN	$-n^3$
v	neighbours	otherwise	0
vi	otherwise	otherwise	0

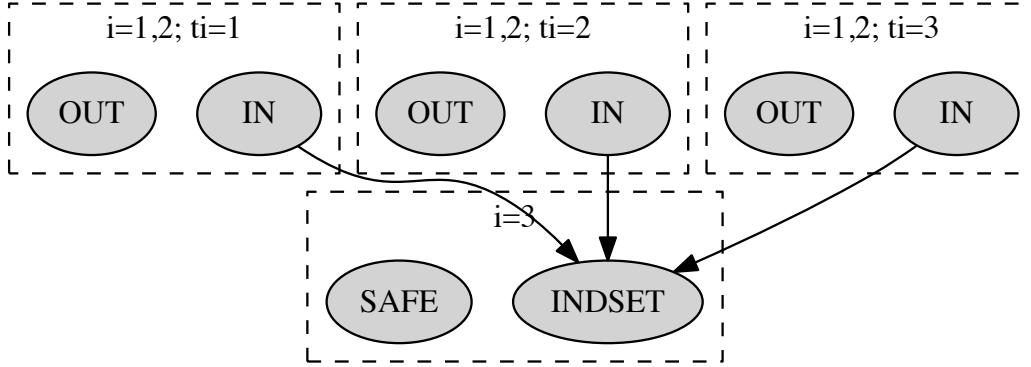


Figure 3.10: An example INDEPENDENTSET instance reduced to dominance in a BAGG. Dashed boxes denote action sets for agents given their types.

Note that these payoffs can be computed by connecting every IN node to the INDSET node.

Sub-claim (Testing for undominance can be used to test for independent set cardinality): For agents 1 and 2, every pure strategy is a mapping from the set of vertices to $\{IN, OUT\}$, and thus corresponds to a subset of the vertices. If agents 1 and 2 play strategies that are not identical, then agent 3's *ex interim* expected utility of INDSET is at most zero. (This is because agent 3 gets a payoff of $-n^3$ with probability at least $1/n^2$ (case iii); the only positive payoff agent 3 can get is n^2 and this happens with probability at most n/n^2 .) If agents 1 and 2 play strategies that are identical but that do not correspond to an independent set, agent 3's *ex interim* expected utility of INDSET is at most zero, by the same reasoning as with non-identical strategies (but using the penalty from case iv). If agents 1 and 2 play strategies that are identical, and that correspond to an independent set with cardinality j , then agent 3's *ex interim* expected utility of INDSET is exactly j (because he gets payoff n^2 (case i) with probability j/n^2). Thus, INDSET is dominated by SAFE (which gets a payoff of $k - 1/2$) unless there is an independent set with cardinality k .

Sub-claim (The reduction is polynomial in time and size): This BAGG has $2n + 2$ action nodes, and all but INDSET have no inputs (and therefore have payoff tables of size 1). INDSET has n inputs, but those inputs must be non-negative integers and must sum to at most two. Thus, the payoff table for INDSET can have only n^2 possible configurations, and can only require $O(n^3)$ space. This $O(n^3)$ factor dominates the other costs in both time and space.

Therefore, UNDOMINANCE(BAGG) is NP-hard. □

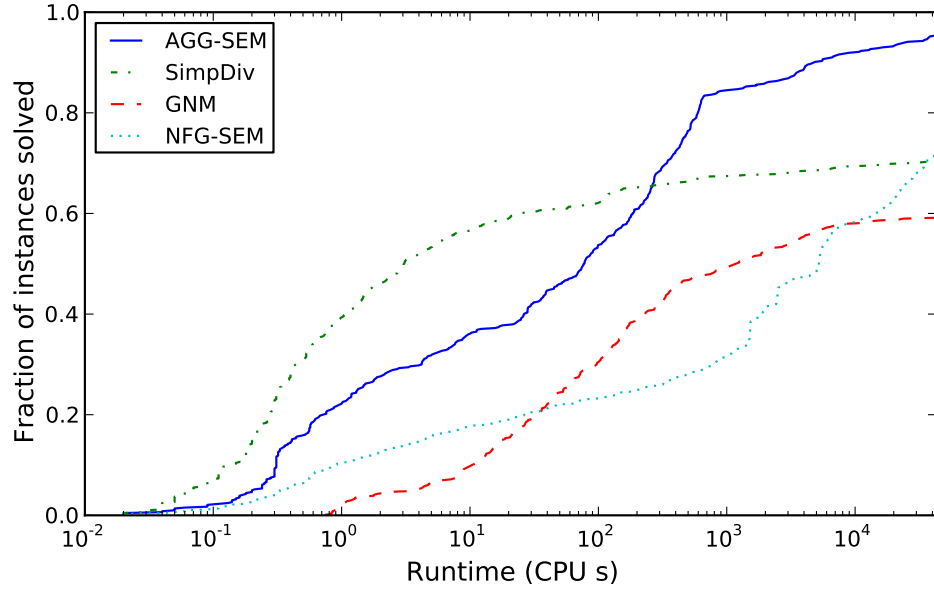


Figure 3.11: Runtime CDFs for AGG-SEM and the three incumbent algorithms

3.6 Conclusion

We have shown that the support enumeration method can be extended to games compactly represented as AGGs. Our approach outperforms the original SEM algorithm for such games both asymptotically and in practice. Theoretically, we showed that SEM’s worst-case runtime can be reduced exponentially. Our work in this vein may also be of independent interest, as it shows novel ways of exploiting AGG structure in theoretical analyses. In particular, the polynomial-time algorithm for removing dominated strategies could be useful, e.g. as a preprocessing step for other equilibrium-finding algorithms. Empirically, we observed that our new algorithm was substantially (often orders of magnitude) faster than the original SEM algorithm, and that it almost always outperformed current state-of-the-art algorithms. Beyond this, our algorithm offers substantial advantages over existing algorithms, such as the ability to enumerate equilibria and to identify pure-strategy Nash equilibria or prove their non-existence.

We envision several extensions to AGG-SEM. One promising direction is to search for specific types of (e.g., symmetric or social-welfare-maximizing) equilibria, for example by replacing the depth-first search with branch-and-bound search. This could be particularly valuable for computational mechanism analysis in cases with many equilibria; sometimes the main bottleneck is to finding an specific equilibrium is that it belongs to exponentially large set, which SEM exhaustively enumerates. Performance could also be improved by using good heuristics to choose the order in which supports are instantiated, or even by exploring the space of supports using stochastic local search rather than tree search. (However, using local search would require giving up the ability to enumerate equilibria, one of SEM’s most unusual and valuable features.)

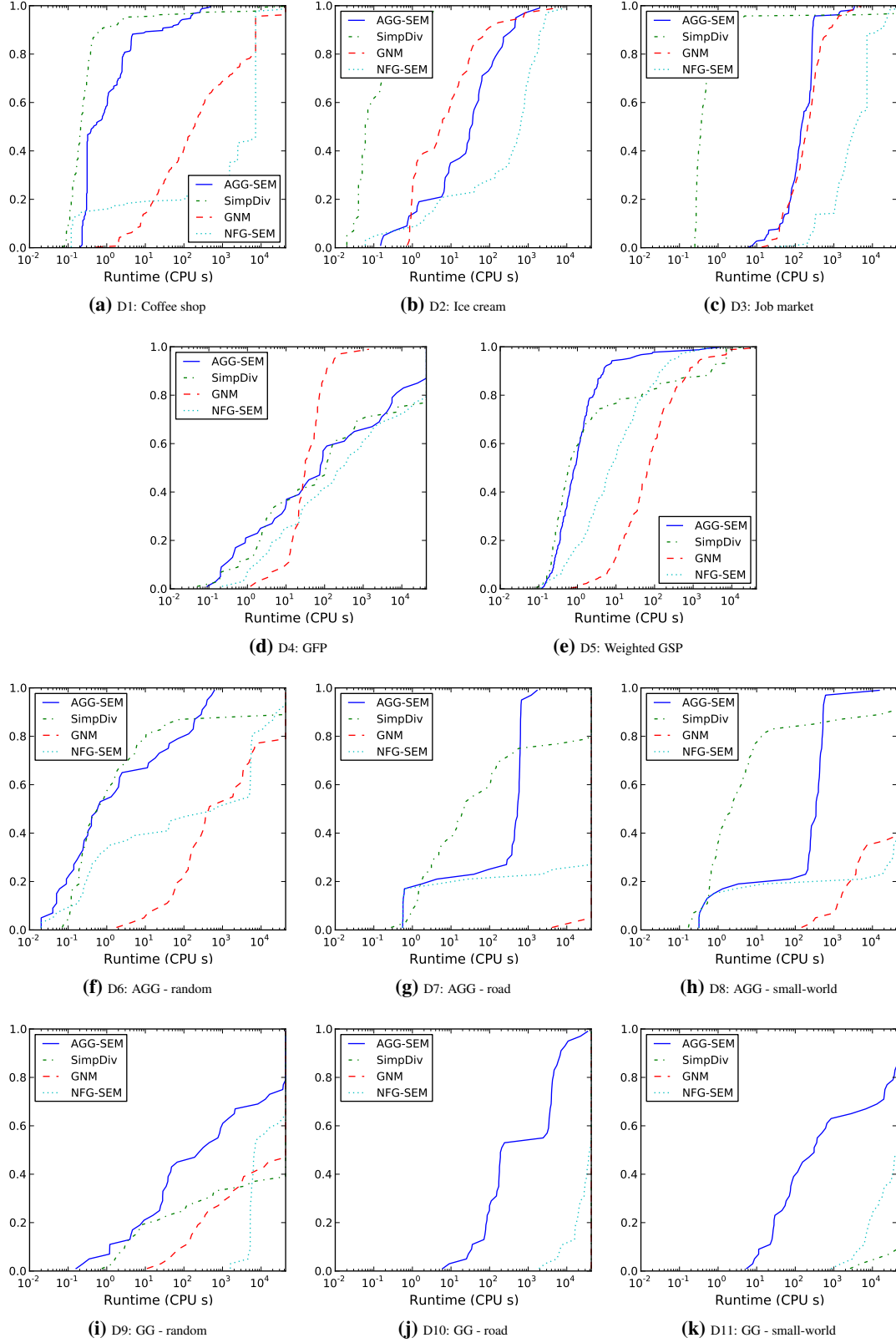


Figure 3.12: Per-distribution runtime CDFs for all four algorithms

Chapter 4

Application: Position Auctions for Internet Advertising

4.1 Introduction

This chapter covers the first application domain for my computational mechanism analysis techniques, position auctions. Position auctions are a relatively new family of mechanisms in which bidders place a single bid for a set of goods of varying quality, and the i th-highest bidder wins the i th-most-desirable good. Each year, these auctions yield billions of dollars selling advertising space on search-engine results pages. Various position auction designs have been considered over the years; e.g., advertisers who attract more clicks may be given advantages over weaker bidders, and bidders may have to pay their own bid amounts or a smaller amount computed from the bids of others. After some initial experimentation with these design dimensions, the major search engines have converged on a single design: the weighted, generalized second-price auction (which we denote wGSP; we define it formally in what follows). The main question that this chapter seeks to address is whether wGSP represents a good choice, as compared both to the auctions it has replaced and to theoretical benchmarks. Specifically, we ask whether wGSP is more economically efficient, whether it generates more revenue, whether it yields results that users find more relevant, and whether it produces low-envy allocations.

There is an enormous literature on auction analysis that seeks to answer such questions. Overwhelmingly, this literature proceeds by modeling a setting as a (Bayesian; perfect-information) game and then using theoretical analysis to describe what occurs in (Bayes–Nash; dominant-strategy; locally envy-free) equilibrium. There is much to like about this approach: it is often capable of determining that a given mechanism optimizes an objective function of interest or proving that no mechanism can satisfy a given set of properties. Indeed, most of what we know about mechanism design was established through such analysis. However, the approach

also has limitations: in order to obtain clean theoretical results it can be necessary to make strong assumptions about bidder preferences and to simplify tie-breaking and bid discretization rules. Even when considering such a simplified version of a given problem, it is often extremely difficult to make quantitative comparisons between non-optimal mechanisms (e.g., which non-revenue-optimizing mechanism yields higher revenue on expectation)? The current state of affairs in the literature is thus that we know a great deal about auctions, but that many open questions appear to be resistant to analysis by known techniques.

In the case of position auctions, most research has used perfect-information Nash equilibrium as the solution concept of choice. This choice is justified by the fact that advertisers interact repeatedly: nearly identical goods—user views of a particular search result page—are sold up to millions of times per day, and advertisers can continuously adjust their bids and observe the effects. A variety of other technical assumptions are commonly made, characterizing advertiser preferences (e.g., a click has the same value regardless of position, and regardless of which other ads are shown), user behavior (e.g., a user’s response to an ad is independent of which other ads she has seen), and advertiser behavior (e.g., an advertiser will act to reduce his envy even when doing so does not increase his utility). With all these assumptions in place, various strong results have been obtained (e.g., wGSP is efficient and generates weakly more revenue than VCG), but other important questions remain open (e.g., does wGSP generate more revenue than other position auctions?). Relaxing any one of these assumptions can lead to many further open questions.

This chapter shows that my computational mechanism analysis techniques are able to address a wide variety of open questions that have not proven amenable to theoretical analysis. We maintain the approach of modeling a mechanism as a game and of reasoning about (exact) solution concepts of interest; we depart from traditional analysis by allowing only a discrete set of bids and by answering questions by providing statistical evidence rather than theorems. Specifically, we sample advertiser preferences from a given distribution, compute an exact Nash equilibrium of the resulting game between advertisers, and reason about this equilibrium to compute properties of the outcome, such as expected revenue or social welfare. By repeatedly sampling, we can make quantitative statistical claims about our position auction setting (e.g., that one auction design generates significantly more expected revenue than another). The results of this chapter demonstrate that my methods are able to yield qualitatively new findings about a widely studied setting.

Computational mechanism analysis differs in many ways from theoretical methods. As already mentioned, one clear disadvantage is that our approach only produces statistical results (e.g., given distribution D , A performs significantly better than B on expectation) rather than theorems (e.g., A always performs better than B). Conversely, our methods have the advantage that they are able to produce results in settings for which such simple patterns do not exist. For example, we have observed distributions over advertiser preferences under which the wGSP auction sometimes generates far less revenue than its predecessor, uGSP, and sometimes generates far more. Thus, we know that any comparison of these auctions must necessarily be statistical and distribution dependent, rather than guaranteeing that one of these auctions always yields more revenue. Our computational approach also allows us to consider arbitrary preference distributions, possibly derived

from real-world data, as opposed to being restricted to distributions with convenient theoretical properties like monotone hazard rates. A further property of computational mechanism analysis is both a benefit and a weakness: bidders must be restricted to a finite set of discrete bids, unlike the vast majority of literature on auction theory, which assumes that bids are continuous. While we depart from this tradition, we do not see discreteness as necessarily disadvantageous; real-world position auctions tend to be rather coarsely discrete. For example, position auctions often clear for tens of cents per click, while bids are required to be placed in integer numbers of cents. Finally, some auction features, like rules for tie-breaking and rounding, are difficult to analyze in the continuous case, but pose no obstacle to our approach.

The bulk of this chapter shows the effectiveness of computational mechanism analysis by demonstrating what it can tell us about position auctions. In Section 4.5 we consider both simple models (such as those of Varian and Edelman *et al*) and richer models (such as cascade) in which advertisers have position-dependent valuations and externalities, in each case using our techniques to shed light on open problems.¹ Some high-level findings emerged from our analysis. Most strikingly, we found that wGSP consistently showed the best ads of any position auction, measured both by social welfare and by relevance (expected number of clicks). Further, even in models where wGSP was already known to have bad worst-case efficiency (either in terms of price of stability or price of anarchy), we found that it almost always had very good average-case performance. In contrast, we found that revenue was extremely variable across auction mechanisms, and furthermore was highly sensitive to equilibrium selection, the preference model, and the valuation distribution. In Section 4.6 we consider the extent to which our findings are sensitive to the bid discretization used, the number of bidders, and the number of slots sold.

4.2 Background

Although a variety of position auction variants have been proposed, only three have seen large-scale use in practice. We describe them here and also provide short form names that we will use throughout the chapter. All auctions are pay-per-click; that is, bidders pay every time an end-user clicks on an advertisement, not every time an advertisement is displayed.

GFP The generalized first-price auction, used by Overture and by Yahoo! from 1997–2002. Each bidder submits a single bid; the highest bidder’s advertisement is displayed in the highest position, the second bidder gets the second-highest position, and so on. Each bidder pays the amount of his bid.

uGSP The unweighted, generalized second-price auction, used by Yahoo! from 2002–2007. As before, bidders are ranked by their bids; the bidder winning the i th position pays the $i + 1$ st-highest bid.

wGSP The weighted, generalized second-price auction, used by Google AdWords and Microsoft adCenter,

¹We previously presented results for the no-externalities models in a conference paper [98]. The representation for models with externalities is new to the current paper, as are all experimental results.

and by Yahoo! since 2007. The search engine assigns each bidder a weight or “quality score”; we model this as the probability that an end-user would click on each bidder’s advertisement if it were shown in the highest position. Bidders are scored by the products of their weights and their bids, and are ranked in descending order of their scores. Each bidder pays the smallest bid amount that would have been sufficient to cause him to maintain his position.

These auctions have all received theoretical analysis under a variety of models, typically with the assumption that bidders will converge, in repeated play, to an equilibrium of the full-information, one-shot game.

4.2.1 Metrics for Evaluating Auction Outcomes

It is tempting to believe that search engines have settled on wGSP because it is a better auction design. To claim this, we need to decide what we mean by “better.” In this chapter, we will consider four such metrics.

1. Most straightforwardly, perhaps search engines gravitated to an auction design that maximizes their (short-term) interests: *revenue*.
2. Perhaps search engines are better off maximizing the *welfare* of advertisers, to ensure advertisers’ ongoing participation, and thus the search-engine’s longer-term revenue. If so, we should assess a mechanism according to the sum of advertisers’ valuations for the allocation achieved in equilibrium.
3. Both of these measures neglect a third group of agents: the search engine’s end-users, without whom there would be no profits for advertisers and thus no revenues for the search engine. Some researchers believe that our second metric, social welfare among advertisers, is a good proxy for end-user payoffs, because advertisers only get revenue from clicks that satisfy users’ needs [4]. Others have argued that click-through rates are a more direct measure of whether advertisements are interesting to users, measuring the *relevance* of a page of search ads by the expected number of clicks it receives [60]. Note that this is an expectation because even if the allocation of positions is deterministic, because whether or not a user clicks on a given ad is (for all the models covered in this paper, at least) a random event.
4. Finally, *envy* is another important measure of the quality of a multi-good allocation. One agent i envies another agent j if i ’s expected utility could be increased by exchanging i ’s and j ’s allocations and payments [44]. Allocations that do not give rise to such envy have been considered desirable in themselves. In the position auction literature, however, it is more common for envy to be used as a tool for equilibrium selection. Many researchers have restricted their attention to envy-free Nash equilibria (i.e., Nash equilibria in which the total envy across all bidders is zero) [30, 103].² In this chapter, we

²Although these researchers focused on locally envy-free equilibria—under which no bidder envies the bidders in adjacent positions—this distinction is not important for our purposes. Under their models, local envy-freeness implies global envy-freeness, using the more general definition of envy from [44]. We need this richer definition of envy—which allows for randomization—to study cases involving randomized tie-breaking or mixed strategies. Even so, this richer definition does not cover externalities—where

also use envy as a tool for equilibrium selection, contrasting envy-free and general Nash equilibria.

4.2.2 Models of Bidder Valuations

A wide range of different models have been proposed for bidder valuations in position auctions. There are broadly two classes of models: those that make a no-externalities assumption—holding an ad’s position constant, it will generate the same expected number of clicks and same expected value regardless of which ads are shown in other positions—and those that do not make such an assumption.

Models Without Externalities

We consider four no-externalities models. The first two models (which we call EOS and V, after the researchers who introduced them) have in common the assumption that each advertiser values clicks independently of their advertisement’s position. The next two models (which we call BHN and BSS) allow for “position preferences”: an advertiser might have different values for clicks when the advertisement appears in different positions. In each case, we describe important results from the literature, as well as open questions that we will address.

EOS Model. Edelman, Ostrovsky and Schwarz [30] analyzed the GSP under a preference model in which each bidder’s expected value per click is independent of position. The click-through rate is the same for all ads in a given position (making uGSP equivalent to wGSP), and decreasing in position (ads that appear lower on the screen get fewer clicks). EOS defined locally envy-free equilibria as Nash equilibria in which no bidder envies the allocation received by a bidder in a neighboring position, and showed that in such equilibria, uGSP is efficient and revenue dominates the truthful equilibrium of VCG. Caragiannis et al [12] showed that other, lower-efficiency Nash equilibria exist, but that none is worse in terms of social welfare by a factor greater than 1.259. (In other words, 1.259 is an upper bound on the *price of anarchy*.) Given that locally envy-free equilibria are only guaranteed to exist in the continuous case, while real wGSP uses discrete increments, some natural open questions about this model follow.

Question 1: Under EOS preferences, how often does wGSP give rise to envy-free (efficient, VCG-revenue-dominating) Nash equilibria? What happens in other equilibria, and how often do they occur?

V Model. Varian [103] analyzed wGSP under a more general model, in which each bidder’s value per click is still independent of position, but click-through rates are decreasing and “separable.” Separability means that for any position/bidder pair, the click-through rate can be factored into a position-specific component that is

an advertiser’s utility depends not just on his own allocation, but also on which agents receive which other goods. Thus, we do not consider envy when working with settings involving externalities.

independent of bidder identity and a bidder-specific component that is independent of position (corresponding to wGSP's weights). Varian showed that in any "symmetric" (globally envy-free) equilibrium, wGSP is efficient and revenue dominates VCG. The price of anarchy result of Caragiannis et al. also applies to the V model. Lahaie and Pennock studied the problem of what happens in this model in uGSP and wGSP (and points in between) [60]. Under additional assumptions about the valuations, they found that uGSP was less efficient than wGSP but generated more revenue. These findings suggest some natural open questions about the V model.

Question 2: Under V preferences, how often does wGSP have envy-free (efficient, VCG-revenue-dominating) Nash equilibria? What happens in other equilibria, and how often do they occur? Is uGSP generally better than wGSP for revenue, or does their relative performance depend on equilibrium selection, the valuation distribution, and/or other properties of the game? Is wGSP generally better for efficiency?

BHN Model. Blumrosen, Hartline and Nong [11] proposed a model that (like the one that follows) allows for the possibility that not all clicks are equally valuable to an advertiser. In this model, click-through rates are still decreasing and separable. However, a bidder's expected value per click increases with the bidder's rank in a separable fashion, subject to the constraint that a bidder's expected value *per impression* is weakly decreasing. The authors support their generalization by describing empirical evidence that conversions (e.g., sales) occur for a higher proportion of clicks on lower-ranked ads. They show that preference profiles exist under their valuation model in which wGSP has no efficient, pure-strategy Nash equilibrium. However, while we know that such preference profiles exist, we do not know how much of a problem they pose on average.

Question 3: Under BHN preferences, how often does wGSP have no efficient Nash equilibrium? How much social welfare is lost in such equilibria?

BSS Model. Benisch, Sadeh and Sandholm [9] proposed another position-preference model that generalizes EOS. In this model, click-through rates are decreasing in position but independent of bidder identities. However, bidders' values are single peaked in position and strictly decreasing from that peak. For example, "brand" bidders might prefer the prestige of top positions, while "value" bidders prefer positions further down. Benisch, Sadeh and Sandholm analyzed this model in an imperfect-information setting, and showed both that uGSP and wGSP ranking rules can be arbitrarily inefficient for such models and that more expressive bidding languages can improve efficiency. For different valuation distributions consistent with their model, they bounded the loss of efficiency in the best-case Bayes-Nash equilibrium. We observe that this bound derives entirely from GSP's inexpressiveness—the fact that agents lack the ability to communicate their true preferences—rather than from agents' incentives. Under the more common assumption of perfect-information Nash equilibria, expressiveness cannot cause an inefficient outcome (because for any order of the bidders, strategy profiles exist that will rank the bidders in that order), but incentives can (some bidder might want to deviate from any efficient strategy profile).

Question 4: Under BSS preferences, how often does wGSP have no efficient perfect-information Nash equilibrium? How much social welfare is lost in such equilibria?

Models with Externalities

So far we have assumed that advertisers do not care about *which* ads appear above and below their own. We now consider three models that relax this assumption: cascade and two further models that generalize cascade (hybrid, GIM).

Cascade Model. The most widely studied model of position auction preferences with externalities is the “cascade model” [2, 36, 56]. It captures the idea that users scan and click ads in the order they appear; a good ad can make lower ones less desirable, while a bad ad can cause a user to give up entirely. More specifically, users scan the ads starting from the top, and each time the user looks at (and possibly clicks on) an ad, she may subsequently decide to stop scanning. In this model, it is possible for wGSP to have low-efficiency equilibria, with price of anarchy 4 [92]. If weights are modified to take into account the probability that a user will continue reading ads, wGSP has a revenue-optimal equilibrium, but this equilibrium may require bidders to bid above their own valuations, which is a weakly dominated strategy.³

Question 5: Under cascade preferences, how often are there low-efficiency equilibria? In cases where low-efficiency equilibria exist, are these the only equilibria? When agents play only undominated strategies, how much revenue can wGSP generate? How do the modified weights affect wGSP’s efficiency?

Hybrid Model. A richer model, which we call “hybrid”, combines features of the separable (V) and cascade models. Users decide whether or not to continue scanning ads based on the number of ads they have already scanned (as in V), and the content of each of those ads (as in cascade) [56]. Under this model, auctions that allocate according to greedy heuristics—including GSP—are not economically efficient; indeed, no economically efficient, polynomial-time allocation algorithm is known [56]. Further, strategic behavior in GSP can lead to greater efficiency loss. When k slots are being sold, the worst Nash equilibrium can be as little as $1/k$ -efficient, while the best can be as little as $2/k$ -efficient [39].

Question 6: With hybrid preferences, how much social welfare is lost? On typical instances (as opposed to worst-case ones), how much difference is there between best- and worst-case equilibria?

GIM Model. Beyond their work on cascade, Gomes, Immorlica and Markakis [40] also introduced an even richer model (which we call GIM) which generalizes the hybrid model in two important ways. First, it allows for arbitrary pairwise externalities. For example, after looking at—and possibly clicking on—the first ad,

³This reweighted wGSP mechanism is only optimal among mechanisms that do not use reserve prices. Also, because some agents’ strategies may be dominated—bidding strictly more than their valuations—the equilibria tend to not be rationalizable.

a user might update her beliefs about which ads are promising, affecting some ads positively and others negatively. Second, this model allows for the possibility that a user’s behavior might depend on which sets of ads she has seen and clicked on before. Its authors argue that this richer model should have similar properties to the cascade model.⁴

Question 7: Are the efficiency and revenue achieved by wGSP substantially different under the cascade and GIM models?

4.3 Representing Position Auctions

We now present algorithms for succinctly encoding various position auction settings as AGGs. Implementations of all our algorithms are freely available at http://www.cs.ubc.ca/research/position_auctions.

4.3.1 Representing No-Externality GFPs as AGGs

We can naively represent a no-externality GFP as an AGG as follows. We begin by creating a distinct action set for each agent i containing nodes for each of his possible bids b_i . Other agents will push agent i below the top position if they bid strictly more than i ’s bid b_i or if they bid exactly b_i and are favored by the tie-breaking rule. Thus, we create directed edges in the action graph pointing to each b_i from every b_j that could potentially be awarded a higher position: every b_j for which $b_j \geq b_i$ and $j \neq i$. (If the tie-breaking rule is randomized, we consider all of the different positions the agent could achieve and return his expected utility.) The price an agent pays is simply the amount of his bid, and so can be determined without adding any more edges.

How much space does this representation require? Let the number of agents be n and the number of bid amounts for each agent be k . Consider the action node b_i^0 corresponding to i ’s lowest bid amount. This node has edges incoming from each action node belonging to another agent, meaning that the utility table corresponding to b_i^0 must store a value for every configuration over these nodes. There are k possible configurations over each agent j ’s action nodes, and $n - 1$ other agents who bid independently, so a total of k^{n-1} such configurations. This expression is exponential, and so we consider the AGG to be intractably large.

However, notice that our naive representation failed to capture key regularities of the no-externalities GFP setting. Bidder i does not care about the amount(s) by which he is outbid, nor does he care about the identities of the bidders who outbid him. We can capture these regularities by introducing function nodes. For every action node b_i , we can create one summation function node (formally, a node whose count is defined as the sum of the counts of its parents in the action graph) counting the number of other bids greater than or equal

⁴This is clearly true for the issue of whether or not clicking on an ad affects the click-through rates of subsequent ads: before the user first clicks, the auction must already have allocated all the ad space. Nevertheless, GIM’s increased range of possible externalities could lead to dramatic differences from cascade in terms of equilibrium outcomes.

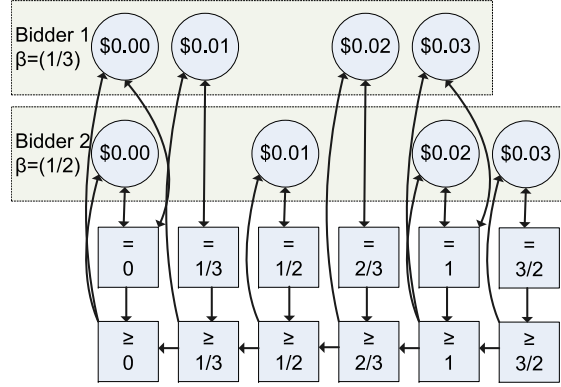


Figure 4.1: A weighted GFP represented as an AGG. (Square nodes represent summation function nodes. β denotes the quality score of each advertiser.)

to b_i , and another counting the number equal to b_i . These two quantities are sufficient for computing the position of an agent who bids b_i . Each node in the action graph is thus connected to only two other nodes. Consider again the table encoding the utility for agent i 's bid b_i^0 . There are $n - 1$ possible configurations over the \geq function node, and for each of these configurations, up to $n - 1$ possible configurations over the $=$ function node, yielding a total of $O((n - 1)^2)$ configurations—a polynomial number. There are nk nodes in the action graph (and none has a larger table), so the graph's total representation size is $O(n^3k)$. An example of such an AGG is given in Figure 4.1. (As it does not complicate the representation, and is useful both to our exposition and in Section 4.5.2, we build *weighted* GFPs, which we define in the natural way. Of course, setting all bidders' weights to 1 returns us to the unweighted case.)

More formally, we define a no-externalities position auction setting as a 4-tuple $\langle N, v, c, q \rangle$ where N is a set of agents numbered $1, \dots, n$; $v_{i,j}$ is agent i 's value per click in position j ; $c_{i,j}$ is agent i 's probability of receiving a click in position j ; and q_i is agent i 's quality (typically, the probability that i will receive a click in the top position). To specify a position auction game we additionally specify the range of allowed bid amounts K and the tie breaking rule T . (For now we will consider only the case where T is uniform-random; in Section 4.6.2 we revisit this choice.) We can now more precisely define our construction as Algorithm 4.2.

4.3.2 Representing No-Externality uGSPs and wGSPs as AGGs

GSPs are similar to GFPs in that each agent's payoff depends on a small number of values. To determine an agent's position (or possible range of positions under randomized tie breaking), we start with a graph structure similar to the one we built for GFPs. The first difference is that we must allow for bidder weights. We do this by creating function nodes for each “effective bid”—each distinct value that can be obtained by multiplying a bid amount by a bidder weight—rather than one for each bid amount.⁵ Of course, we can

⁵We can also derive AGG representations of Lahaie and Pennock's ranking rules [60] by adjusting the values of q appropriately.

```

foreach agent  $i \in N$  do
  foreach bid  $k \in K$  do
    | create an action node representing  $i$  bidding  $k$ ;
  foreach effective bid  $e \in \{k \cdot q_i \mid \forall i \in N, \forall k \in K\}$  do
    | create a summation function node  $(=, e)$  counting the bidders bidding exactly  $e$ ;
    | create a summation function node  $(\geq, e)$  counting the bidders bidding above  $e$ ;
    | add an arc from  $(=, e)$  to  $(\geq, e)$ ;
    | if  $e > 0$  then
      | | add an arc from  $(\geq, e)$  to  $(\geq, e')$  (where  $e'$  is the next largest effective bid);
  foreach action node  $a_{i,k}$  representing  $i$  bidding  $k$  do
    | add an arc from  $a_{i,k}$  to  $(=, k \cdot q_i)$ ;
    | add an arc from  $(=, k \cdot q_i)$  to  $a_{i,k}$ , and denote the value of  $(=, k \cdot q_i)$  as  $\ell$ ;
    | add an arc from  $(\geq, k \cdot q_i)$  to  $a_{i,k}$ , and denote the value of  $(\geq, k \cdot q_i)$  as  $g$ ;
    | instantiate the utility table for  $a_{i,k}$  as
      
$$u_{a_{i,k}}(\ell, g) = \frac{1}{\ell} \sum_{j=g+1}^{g+\ell} c_{i,j}(v_{i,j} - k).$$


```

Figure 4.2: An algorithm for converting a no-externality auction setting into an action graph representing a (weighted) GFP. Inputs are an AGG setting (N, v, c, q) , a set of allowable bid values K and the tie-breaking rule T ; this algorithm assumes that T corresponds to uniform randomization.

recover the unweighted case (and thus represent uGSP) by setting all weights to 1. Because effective bids are real numbers while the set of possible bids and payments K is discrete, our definition of a position auction game must now also include a “rounding rule” R that is used to determine payments. For most of the chapter we will only consider the case where R corresponds to rounding up (meaning that agents pay the minimum amount that they could have bid to maintain their positions); we will revisit this choice in Section 4.6.3.

We also need to augment the action graph to capture the GSP pricing rule. We do this by adding “price nodes”: function nodes that identify the next-highest bid below the bid amount encoded by each given action node. We use the term *argmax node* to refer to a function node whose value is equal to the largest in-arc carrying a non-zero value, based on a given, fixed ordering of the in-arcs. By ordering action nodes according to the values of their effective bids (i.e., bids multiplied by bidder weights), an argmax node identifies the highest effective bid among the subset of action nodes connected to it. Our encoding is defined more precisely in Algorithm 4.4. An example of the resulting action graph is illustrated in Figure 4.3.⁶ This representation results in a graph containing nm action nodes, each of which stores a payoff table with at most $O(n^2|E|)$ entries, where E is the set of effective bids and $|E| \leq nm$. Thus, this representation requires $O(n^4m^2)$ space.

⁶Note that although the in-degree of the argmax nodes can get large— $O(nm)$ —the computational complexity of solving an AGG only depends on the in-degrees of the *action* nodes.

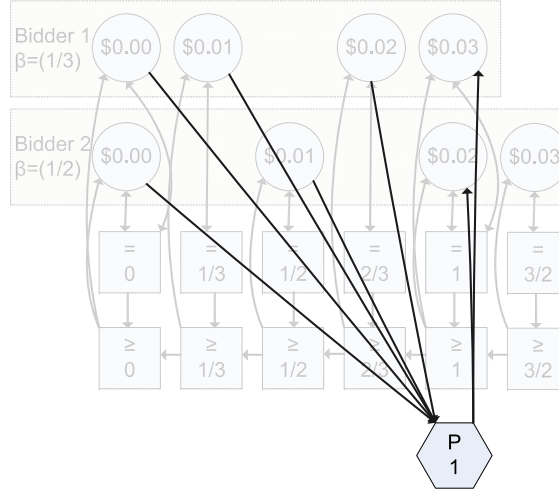


Figure 4.3: To represent a GSP as an AGG, we add price nodes (argmax nodes denoted by hexagons) to a GFP representation. For clarity only one price node is pictured; a full GSP representation requires one price node for each effective bid.

4.3.3 Representing Auctions with Externalities

We now describe an algorithm for representing GIM auctions as AGGs; this also suffices for cascade and hybrid, which GIM generalize. We define a GIM position auction setting by the 4-tuple $\langle N, v, q, f \rangle$ where N is a set of agents numbered $1, \dots, n$; v_i is agent i 's value per click; q_i is agent i 's quality (the probability that i will receive a click in the top position); and $f_i : 2^N \rightarrow \mathbb{R}$ encodes the externalities that affect i . When agent i 's ad is shown below the ads of the agents in set S , his probability of receiving a click is $q_i f_i(S)$. We assume that f is monotone decreasing (i.e., $S' \subseteq S$ implies $f(S') \geq f(S)$), and normalized so that $f(\emptyset) = 1$.

A GSP auction in a GIM setting can be converted to an AGG using Algorithm 4.5. This representation results in a graph containing nk action nodes, where each node has $2n - 1$ in-arcs. The total size of a payoff table for any node is $O(2^{2n}nk)$, so the full representation requires $O(2^{2n}n^2k^2)$ space. However, storing a single agent's preferences requires $O(2^n)$ values (to encode $f(\cdot)$), so the AGG representation is only quadratically larger than the input. Nevertheless, *GIM* settings with large n are impractical.

4.4 Experimental Setup

Broadly speaking, the method we employ in this chapter is to generate many preference-profile instances from distributions over each of the preference models, to build AGGs encoding the corresponding perfect-information auction problems for each auction design, to solve these AGG computationally, and then to compare the outcomes against each other and against VCG.

```

foreach agent  $i \in N$  do
  foreach bid  $k \in K$  do
     $\perp$  create an action node representing  $i$  bidding  $k$ ;
  foreach effective bid  $e \in \{k \cdot q_i \mid \forall i \in N, \forall k \in K\}$  do
    create a summation function node  $(=, e)$  counting the bidders bidding exactly  $e$ ;
    create a summation function node  $(\geq, e)$  counting the bidders bidding above  $e$ ;
    add an arc from  $(=, e)$  to  $(\geq, e)$ ;
    if  $e > 0$  then
       $\perp$  add an arc from  $(\geq, e)$  to  $(\geq, e')$  (where  $e'$  is the next largest effective bid);
    create a weighted argmax function node  $(\arg \max, e)$  identifying the next-highest effective bid below  $e$ ;
    foreach action node  $a$  corresponding to effective bid  $e'$  do
      if  $e' < e$  then
         $\perp$  add an arc from  $a$  to  $(\arg \max, e)$  with arc weight of  $e'$ ;
      if  $e' = e$  then
         $\perp$  add an arc from  $(\arg \max, e)$  to  $a$ , and denote the value of  $(\arg \max, e)$  as  $\rho$ ;
  foreach action node  $a_{i,k}$  representing  $i$  bidding  $k$  do
    add an arc from  $a_{i,k}$  to  $(=, k \cdot q_i)$ ;
    add an arc from  $(=, k \cdot q_i)$  to  $a_{i,k}$ , and denote the value of  $(=, k \cdot q_i)$  as  $\ell$ ;
    add an arc from  $(\geq, k \cdot q_i)$  to  $a_{i,k}$ , and denote the value of  $(\geq, k \cdot q_i)$  as  $g$ ;
    add an arc from  $(\arg \max, k \cdot q_i)$  to  $a_{i,k}$ , and denote the value of  $(\arg \max, k \cdot q_i)$  as  $\rho$ ;
    instantiate the utility table for  $a_{i,k}$  as

      
$$u_{a_{i,k}}(\ell, g, \rho) = \frac{1}{\ell} (c_{i,g+\ell}(v_{i,g+\ell} - \lceil \rho/q_i \rceil)) + \sum_{j=g+1}^{g+\ell-1} c_{i,j}(v_{i,j} - k).$$


```

Figure 4.4: An algorithm for converting an auction setting into an action graph representing a wGSP. (Specifically, this algorithm encodes wGSP with uniform random tie-breaking and prices that are rounded up to whole increments. These choices are revisited in Sections 4.6.2 and 4.6.3, respectively.)

4.4.1 Problem Instances

We generated our preference profiles by (1) imposing a probability distribution over a preference model and then (2) drawing instances from those distributions. Except for BSS, whose definition already includes a distribution, we take two approaches to imposing distributions: one is to assume that all variables are uniformly distributed over their acceptable ranges, and the other is to draw variables from the log-normal distributions of Lahaie and Pennock [60] that have been fitted to real-world data. Thus, we used a total of 13 distributions (see Table 4.1). For each we generated 200 preference-profile instances for each of the three position-auction types, GFP, uGSP and wGSP, yielding $13 \times 200 \times 3 = 7800$ perfect-information games in

```

foreach agent  $i \in N$  do
  foreach bid  $k \in K$  do
    | create an action node representing  $i$  bidding  $k$ ;
foreach pair of agents  $i, j \in N$  do
  foreach  $a_i \in A_i$  do
    | create an OR function node  $(=, a_i, j)$  representing that  $j$  having the same effective bid as  $i$ ;
    | create an OR function node  $(>, a_i, j)$  representing that  $j$  having a strictly greater effective bid than  $i$ ;
    | create a weighted argmax function node  $(\rho, a_i)$  representing the price  $i$  pays given  $a_i$ ;
    | create arcs from all three function nodes to  $a_i$ ;
    foreach  $a_j \in A_j$  do
      | if  $a_j > a_i$  then
      |   | create an arc from  $a_j$  to  $(>, a_i, j)$ ;
      | if  $a_j = a_i$  then
      |   | create an arc from  $a_j$  to  $(=, a_i, j)$ ;
      | if  $a_j < a_i$  then
      |   | create an arc from  $a_j$  to  $(\rho, a_i)$ ;
foreach agent  $i \in N$  do
  foreach action node  $a_{i,k}$  representing  $i$  bidding  $k$  do
    | Let  $L$  denote the set of agents with the same effective bid as  $a_{i,k}$ ;
    | Let  $G$  denote the set of agents with greater effective bids than  $a_{i,k}$ ;
    | Let  $\rho$  denote the next highest effective bid after  $a_{i,k}$ ;
    | instantiate the utility table for  $a_{i,k}$  as
      |
      | 
$$u_{a_{i,k}}(L, G, \rho) = \frac{f_i(G \cup L)}{2^{|L|}}(v_{i,g+\ell} - \lceil \rho/q_i \rceil) + \sum_{S \in 2^L \setminus L} \frac{f_i(G \cup S)}{2^{|L|}}(v_{i,j} - k).$$


```

Figure 4.5: Creating the action graph for a GIM position auction

Without externalities	With externalities
EOS-UNI	Cascade-UNI
EOS-LN	Cascade-LN
V-UNI	Hybrid-UNI
V-LN	Hybrid-LN
BHN-UNI	GIM-UNI
BHN-LN	GIM-LN
BSS	

Table 4.1: The distributions we considered for our experiments. UNI denotes distributions that are uniform across all values. LN denotes distributions with parameters log-normal distributed following [60], with additional parameters not discussed in that work again uniformly distributed.

total. Each of these games had 5 bidders, 5 positions and 30 bid increments. In Section 4.6 we describe further experiments in which we investigated the our findings’ sensitivity to these and other parameters.

We normalized values in each game so that the highest was equal to the highest possible bid, to ensure that the full number of bid increments was potentially useful. We also removed weakly dominated strategies from each game, both for computational reasons and to select against implausible equilibria. Specifically, we eliminated strategies in which agents bid strictly more than (the ceilings of) their valuations.⁷

4.4.2 Equilibrium Computation

We performed our experiments on WestGrid’s Orcinus cluster—384 machines with dual Intel Xeon E5450 3.0GHz CPUs, 12MB cache and 16GB RAM, running 64-bit Red Hat Enterprise Linux Server 5.3. To compute Nash equilibria, we used three algorithms:⁸ (1) `simpdiv`, the simplicial subdivision algorithm of van der Laan *et al* [102], adapted to AGGs by Jiang, Bhat and Leyton-Brown [51]; (2) `gnm`, the global Newton method of Govindan and Wilson [43], adapted to AGGs by Jiang, Bhat and Leyton-Brown [51]; and (3) `sem`, the support-enumeration method of Porter, Nudelman and Shoham [84], adapted to AGGs by Thompson, Leung and Leyton-Brown [100]. We ran `sem` to enumerate all pure-strategy Nash equilibria. For finding sample mixed-Nash equilibria, we ran `simpdiv` from ten different pure-strategy-profile starting points chosen uniformly at random, and also ran `gnm` ten times with random seeds one through ten. We limited runs of `simpdiv` and `gnm` to five CPU minutes. In total, we spent about 15 CPU months on equilibrium computation.

4.4.3 Benchmarks: VCG and Discretized VCG

As well as comparing GSP and GFP to each other, we also compared these position auctions to VCG. There are two ways of doing this. First, we considered VCG’s truthful equilibrium given agents’ actual (i.e., non-discretized) preferences. However, this prevents us from determining whether differences arise because of the auction mechanism or discretization. To answer this question, we also compared to an alternate version of VCG in which we discretized bids to the same number of increments as in the position auctions. In this case, we assume that bidders report the discrete value nearest to their true value. (Observe that this is always an ϵ -Nash equilibrium with ϵ equal to half a bid increment.)

⁷In past work, for computational reasons, we also eliminated *very weakly* dominated strategies. In particular, this included bids that led to identical outcomes regardless of the actions of other agents (e.g., from an agent with an extremely low quality score). We did not do this here because we now enumerate equilibria: although the elimination of very weakly dominated strategies does not change the set of equilibrium outcomes, it can change the relative frequency of these outcomes.

⁸Implementations of all three algorithms are available at <http://agg.cs.ubc.ca>.

4.4.4 Statistical Methods

In order to justify claims that one auction achieved better performance than another according to a given metric (e.g., revenue), we ensure that this difference was judged significant by a statistical test. Specifically, we performed blocking, means-of-means, bootstrapping tests [15] as follows:

1. For each setting instance, find the difference in the metric across that pair of auctions on that instance. Each value is normalized by the achievable social welfare in that instance. Call this set of values S .
2. Draw $|S|$ samples from S (with replacement), and compute the mean. Perform this procedure 20,000 times. Let M denote the set of means thus computed.
3. Our estimated performance difference is the mean of M (the mean-of-means of S).
4. This difference has significance level α if the α^{th} quantile of M is weakly greater than zero.

When reporting results, we use the symbol $*$ to denote that a result has a significance level of at least $\alpha = 0.05$ and $**$ to denote that a result has a significance level of at least $\alpha = 0.01$. For each group of data points (i.e., for a specific size and preference model) we perform many simultaneous tests, comparing revenue, welfare, relevance and envy between all pairs of auctions. To avoid spurious claims of significance, we thus perform Bonferroni multiple-testing correction (effectively, dividing the desired significance level by the number of tests performed) [73].

To avoid undermining the statistical reliability of our data, we did not drop individual games that we could not solve within our chosen time budget: we worried that the features that made some instances hard to solve could also make their equilibrium outcomes qualitatively different from those of easier-to-solve games. Instead, when we were not able to identify any Nash equilibria of a particular game (typically involving GFP auctions), we replaced the metric value of interest with an suitable upper or lower bound (e.g., a position auction’s revenue is trivially guaranteed to be between 0.0 and the maximum possible social welfare). When we have incomplete data about one or both of auctions A and B , we do not claim that auction A achieves significantly better performance according to some metric than auction B unless the lower bound on A ’s performance is significantly better than the upper bound on B ’s performance.

4.5 Results

We now turn to our experimental results. Our goal is to demonstrate the effectiveness of computational mechanism analysis in general and to shed light on open questions about position auctions in particular. In the latter vein our main aim is to justify the search industry’s convergence on the wGSP auction; we also consider the seven model-specific questions we asked in the introduction. This section thus begins with a broad comparison of the different position auction mechanisms, followed by a more detailed examination of

each individual model. We also describe some follow-up experiments prompted by model-specific claims in the research literature. The last section of this chapter gives all of our results in tabular form (i.e., providing numerical values rather than just graphs) and also reports the results of statistical significance tests for all comparisons.

4.5.1 Main Comparison

We begin by looking at the relative performance of GFP, uGSP, and wGSP position auctions, averaged across all of our different models and distributions, and compared to the VCG and discrete VCG baselines as appropriate (see Figure 4.6). In a nutshell, the industry’s choice of wGSP appears to be justified in terms of efficiency, relevance and (to a lesser extent) envy, but not in terms of revenue. More specifically, in terms of both efficiency and relevance, we found that wGSP was clearly the best position auction design; it also exhibited relatively little variation in these metrics (less than 10%) across equilibria. wGSP ranked lower on efficiency and relevance than VCG (with or without discretization), but was surprisingly close, given that in many of our models wGSP is known to be inefficient. wGSP also clearly outperformed GFP and uGSP in terms of envy, but also exhibited very substantial variation across equilibria. Revenue comparisons were much more ambiguous: all auctions achieved fairly similar median revenues, and variation from one equilibrium to another could be very large for both GSP variants (with best-case equilibrium revenues of about twice worst-case equilibrium revenues). We also saw substantial revenue variation across models, distributions, and equilibrium selection criteria, which we will discuss in detail in what follows.

4.5.2 Basic Models: EOS and V

We consider the EOS and V models together because they are very similar, both in terms of what was known from previous work, and in terms of our findings. The main difference between EOS and V is that V introduces quality scores, which can be used to weight advertisers’ bids. Thus, in EOS, wGSP and uGSP are identical, while in V they are not. Earlier we asked the following two questions.

Question 1: Under EOS preferences, how often does wGSP give rise to envy-free (efficient, VCG-revenue-dominating) Nash equilibria? What happens in other equilibria, and how often do they occur?

Question 2: Under V preferences, how often does wGSP have envy-free (efficient, VCG-revenue-dominating) Nash equilibria? What happens in other equilibria, and how often do they occur? Is uGSP generally better than wGSP for revenue, or does their relative performance depend on equilibrium selection, the valuation distribution, and/or other properties of the game? Is wGSP generally better for efficiency?

We begin by investigating how often envy-free equilibria exist in wGSP (with discrete bids) and how many

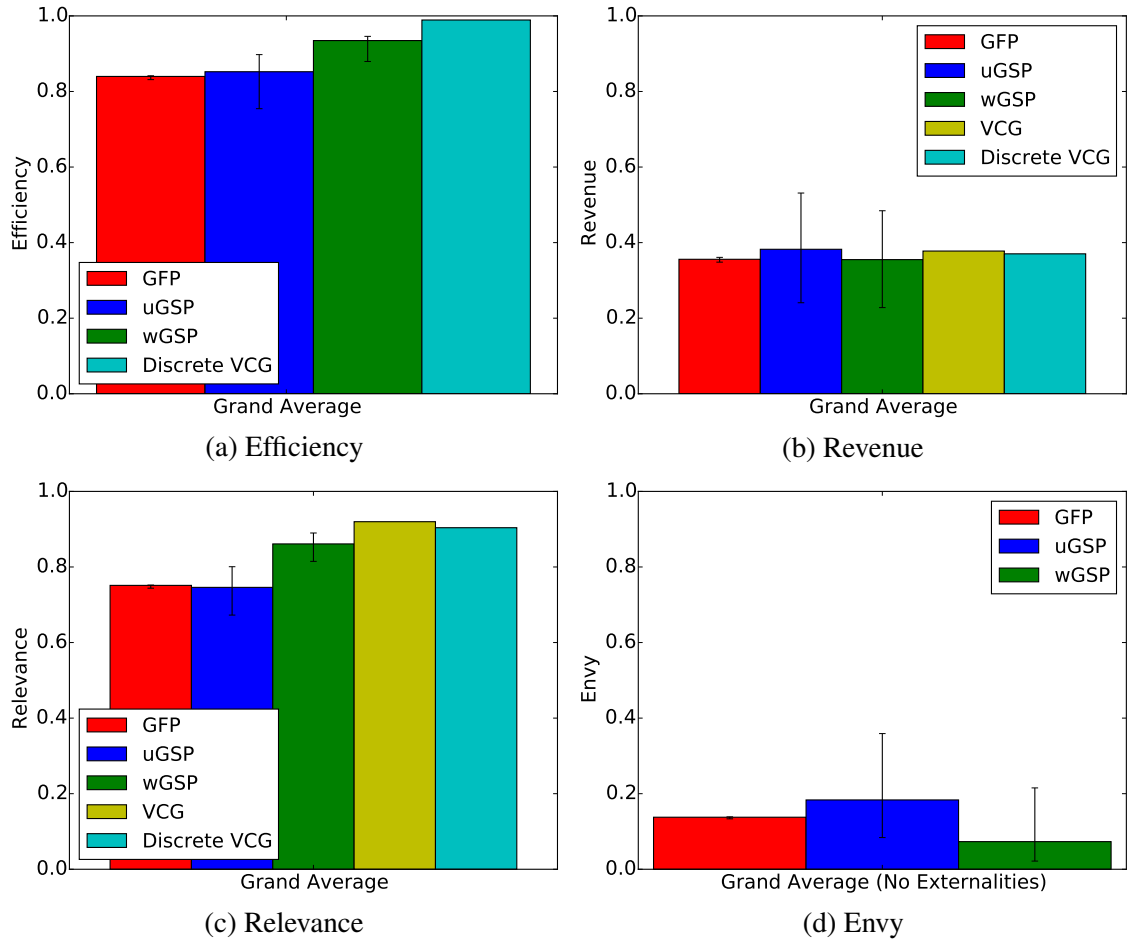


Figure 4.6: Performance of different position auction types, averaged across all 13 distributions (efficiency, revenue, and relevance) and the 7 no-externality distributions (envy, which does not apply to settings with externalities). The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

such equilibria there are, as compared to pure Nash equilibria. Surprisingly, we found that envy-free equilibria did not exist in the majority of games, despite most games having hundreds or thousands of pure-strategy Nash equilibria (see Figure 4.7), and that even when they did exist there were orders of magnitude more Nash equilibria than envy-free equilibria. This led us to investigate the *amount* of envy present in wGSP equilibria. We found that this quantity varied substantially across equilibria, but that envy-minimizing Nash equilibria tended to get very close to zero envy (see Figure 4.8).

We know that in the EOS and V models, envy-free equilibria are economically efficient and yield high revenue; however, we have just observed that such envy-free equilibria did not reliably exist in our games. We next investigated the efficiency and revenue properties of general Nash equilibria. We found that wGSP was

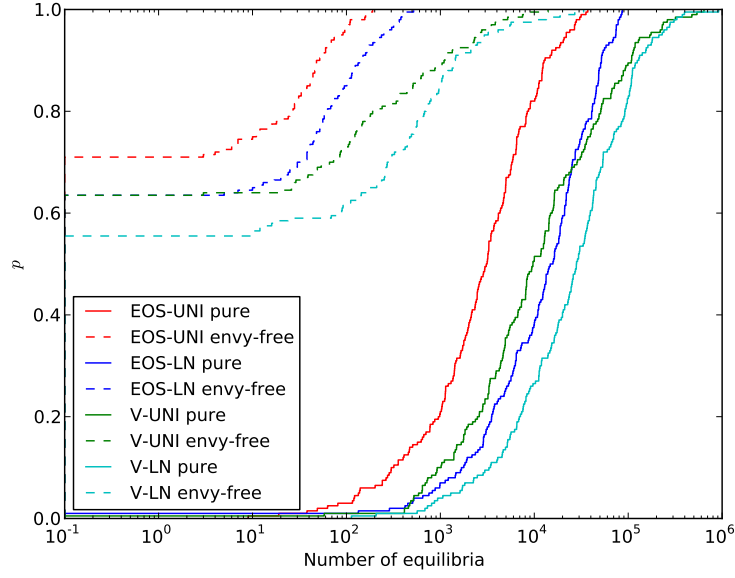


Figure 4.7: Empirical cumulative probability distributions over the number of equilibria. Many lines do not begin at $p = 0$, due to our use of a log scale; a line beginning at $p = 0.6$ indicates that 60% of games had zero envy-free Nash equilibria.

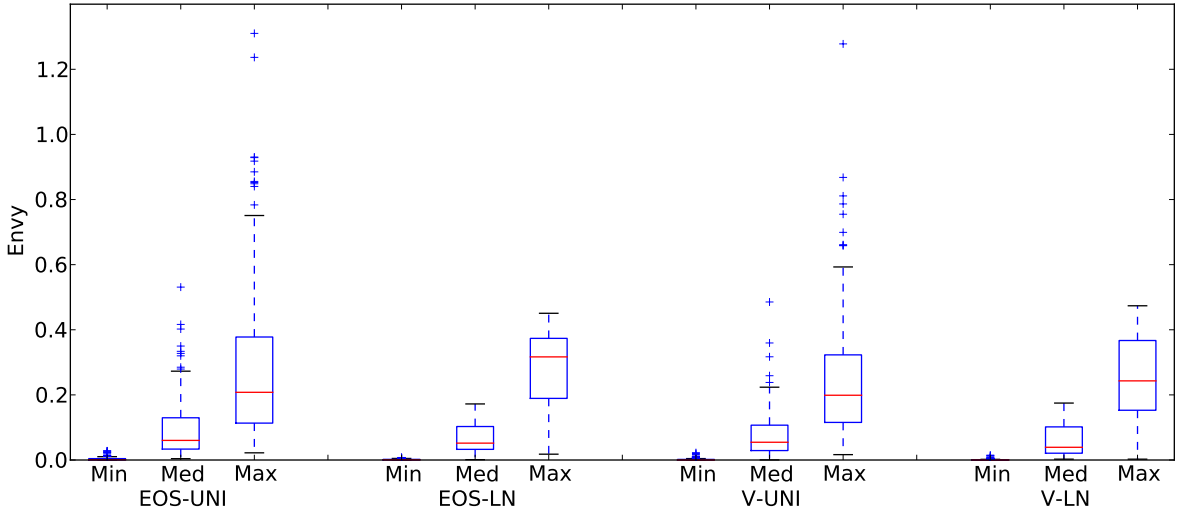


Figure 4.8: Empirical box plot of wGSP's envy under different equilibrium selection criteria (minimum, median and maximum). The envy-minimizing Nash equilibrium always had very little envy, but other equilibria can have substantial envy. Note that we report envy normalized by total possible social welfare. Even normalized, envy can occasionally exceed 1, as it is possible for an agent to be envied by many other agents.

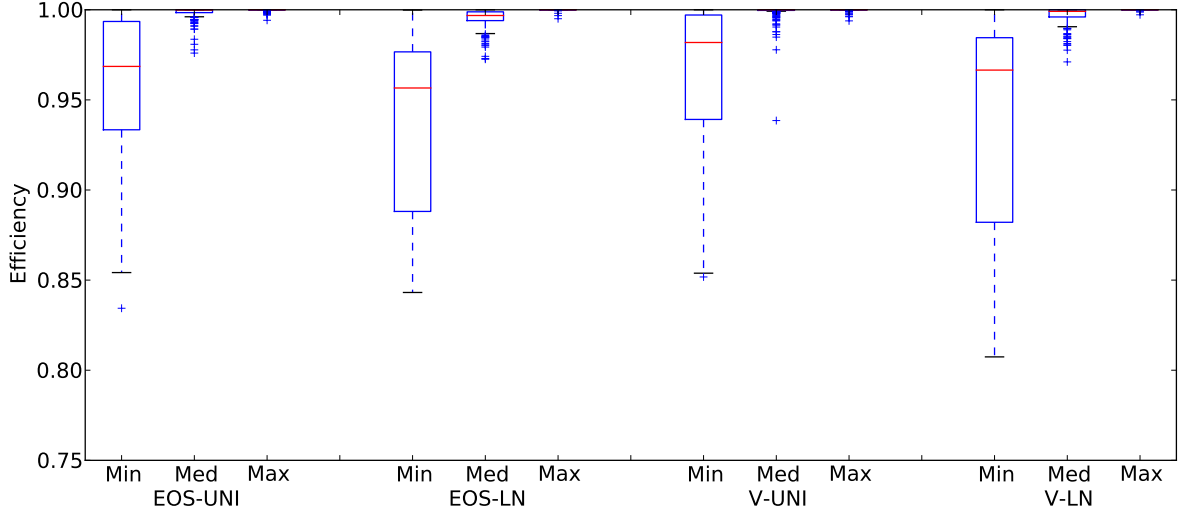


Figure 4.9: Empirical box plot of wGSP’s social welfare under different equilibrium selection criteria (minimum, median and maximum). Note that the majority of equilibria (indicated by the median) tend to be very nearly efficient.

very efficient, not just in the best-case equilibrium, but in the majority of equilibria; however, the worst-case equilibria could be substantially less efficient (see Figure 4.9).⁹

Concerning revenue, we found that equilibrium selection has a very large effect. Further, this effect is not merely confined to a few very bad equilibria. In every distribution, we found that wGSP’s best-case revenue was better than VCG, worst-case was much worse than VCG and median was slightly worse in EOS settings and approximately equal in V settings (see Figure 4.10).

It remains to compare uGSP with wGSP under the V model in terms of both revenue and efficiency. First, recall that Lahaie and Pennock [60] found that uGSP generated much more revenue but moderately less social welfare, but only for the specific case of “symmetric equilibria” and log-normal distributions. For both V distributions, we found that uGSP was better than wGSP for revenue (provided that we compared the mechanisms under the same equilibrium selection criterion, e.g., median), but that both mechanisms were very sensitive to equilibrium selection: the difference between mechanisms ($\sim 1.2\times$) was very small compared to the difference between good and bad equilibria (at least $\sim 2\times$) (see Figure 4.11). The gains from uGSP were much larger in the log-normal distribution than in the uniform distribution. Concerning efficiency, wGSP was indeed better overall. We also found that uGSP’s social welfare was far more sensitive to equilibrium selection (at least $\sim 1.2\times$) than wGSP’s ($\sim 1.04\times$), and even uGSP’s best equilibria were

⁹We note that in our previous work we found almost no instances with such low efficiency [98]; we attribute this discrepancy to the fact that we did not previously have a practical algorithm for enumerating Nash equilibria, and so we previously reported minimum and maximum efficiency based only on the most extreme equilibria our sample-Nash-finding algorithms could find. In contrast, our current work uses the SEM algorithm, and hence guarantees that we have truly found the best/worse case PSNEs, and furthermore allows us to identify the median equilibrium (which we could not do before).

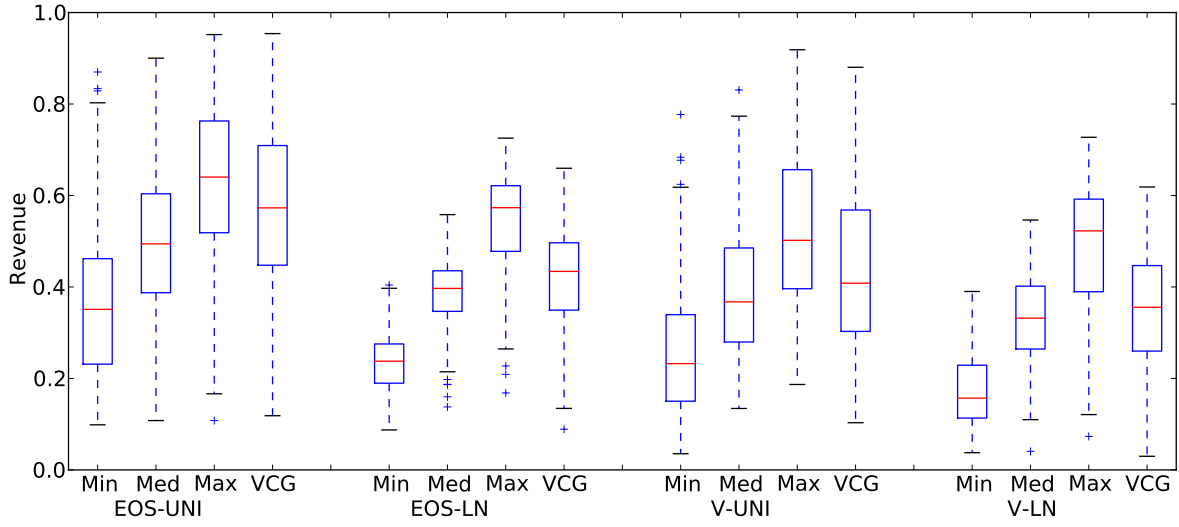


Figure 4.10: Empirical box plot of wGSP's revenue under different equilibrium selection criteria (minimum, median and maximum). The median equilibrium of wGSP performs similarly to VCG, while the worst-case and best-case equilibria are substantially different.

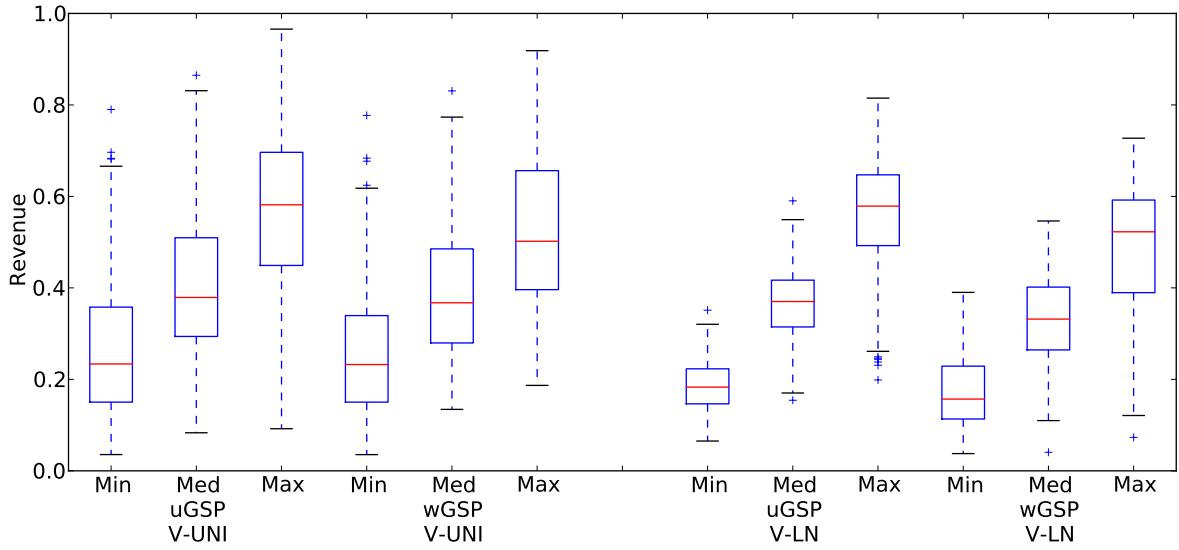


Figure 4.11: Empirical box plot of uGSP and wGSP's revenue under different equilibrium selection criteria (minimum, median and maximum).

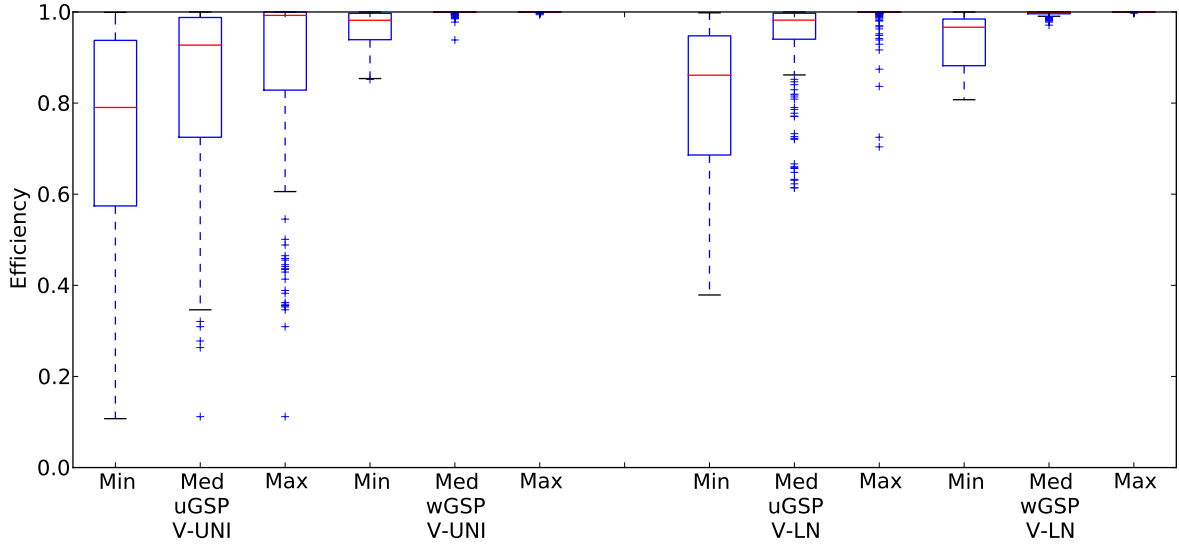


Figure 4.12: Empirical box plot of uGSP and wGSP’s social welfare under different equilibrium selection criteria (minimum, median and maximum).

sometimes very inefficient (see Figure 4.12).

Finally, we come to the overarching question of this chapter, “which position auction design is best?” We conducted a general 4-way comparison between GFP, uGSP, wGSP and VCG for each metric. In terms of social welfare, we found that wGSP was clearly the most efficient position design in the V model, regardless of equilibrium selection. In the EOS model, u/wGSP was more efficient than GFP in best- and median-case equilibria, but not in the worst case (see Figure 4.13(a)). Comparing revenue, we found that equilibrium-selection effects were very large for GSP auctions—much larger than differences between auction designs (see Figure 4.13(b)). In terms of relevance (i.e., expected number of clicks),¹⁰ we see that wGSP was consistently better than the other designs and roughly comparable to VCG (see Figure 4.13(c)). Finally, treating envy as an objective to minimize, in EOS settings we see that in GFP performed very well in all equilibria, while GSP designs were more sensitive to equilibrium selection. In the V model, unweighted designs performed very poorly compared to wGSP (see Figure 4.13(d)).

Weighted GFP

Under the EOS model, we found that GFP had high efficiency in its worst-case equilibria, compared to the other two position auctions. This finding nicely compliments another by Chawla and Hartline [13]: GFP’s worst-case Bayes-Nash equilibrium is efficient in games with independent, identically distributed private values. Because the EOS model does not include bidder-specific click probabilities, GFP’s unweighted

¹⁰We do not consider the EOS model for relevance: because it is unweighted, it predicts that every complete allocation—where all positions are sold—will produce the same expected number of clicks.

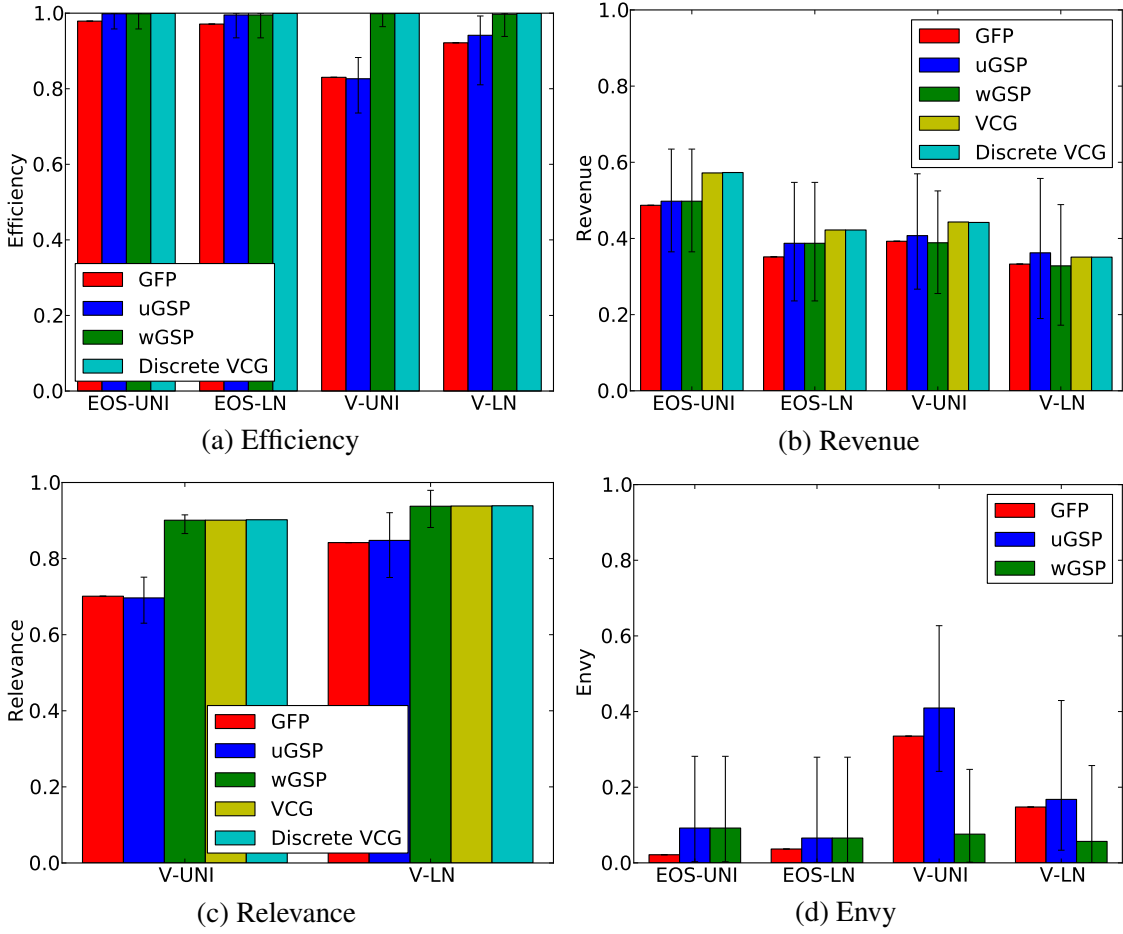


Figure 4.13: Comparing the average performance of different position auction types in EOS and V settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

allocation is not a problem. We thus wondered whether adding weights to GFP would produce a mechanism with good worst-case Nash equilibria under the V model, which does include bidder-specific click probabilities. Recall that it is straightforward to include weights in our AGG reduction for GFP position auctions. We performed two experiments, one at the same scale as our others (where enumerating all mixed-strategy Nash equilibria (MSNEs) of weighted GFP is computationally infeasible) and one at a scale small enough to permit such enumeration.

In the smaller experiment, we found that 98.5% of games had a single mixed Nash equilibrium. When multiple equilibria existed, they tended all to have nearly identical revenue and social welfare (within 4% and 1% respectively). In our full-sized experiments, we also observed little variability across equilibria. In 97% of games, revenue and social welfare varied less than 0.1% from the best case to the worst case. In the remaining 3% of games, social welfare never differed by more than 0.3% across equilibria while revenue never differed

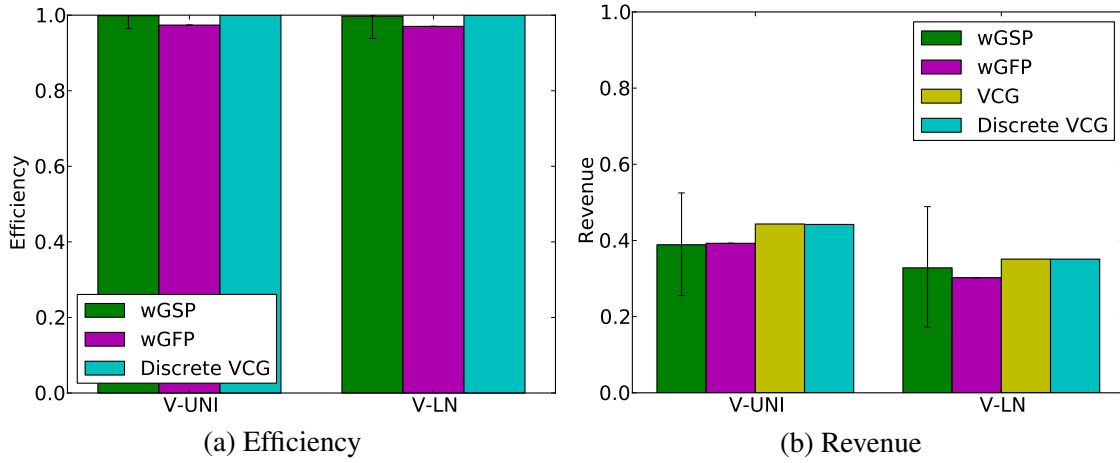


Figure 4.14: Comparing wGFP to wGSP. We found that wGFP’s efficiency varied extremely little across equilibria, but that its worst case was not better than wGSP’s. wGFP’s revenue again varied little, with worst-case revenue much larger than wGSP’s worst case and roughly comparable to wGSP’s median case.

by more than 3.5%. Comparing wGFP to wGSP, we found that wGFP *did not* yield significantly better worst-case efficiency. However, wGFP did achieve much higher worst-case revenue, roughly equal to wGSP’s median case (see Figure 4.14).

4.5.3 Position-Preference Models

We now consider the richer BHN and BSS models, which allow advertisers’ values to depend on ad positions.

The BHN Model

Blumrosen, Hartline and Nong’s [11] key motivation was that advertisers may prefer clicks in the lowest position, because the few users who actually click on low-position ads are likely to buy. They found that wGSP sometimes has no efficient Nash equilibrium under their model.

Question 3: Under BHN preferences, how often does wGSP have no efficient Nash equilibrium? How much social welfare is lost in such equilibria?

We observed two kinds of efficiency failures (see Figure 4.15). The first kind involved a complete failure to allocate, with every agent bidding zero. This equilibrium arises when the top-most position has so many spurious (non-converting) clicks that paying even one increment per click is too much. (This scenario also leads to an interesting outcome under discrete VCG: the advertiser who gets the top position can actually

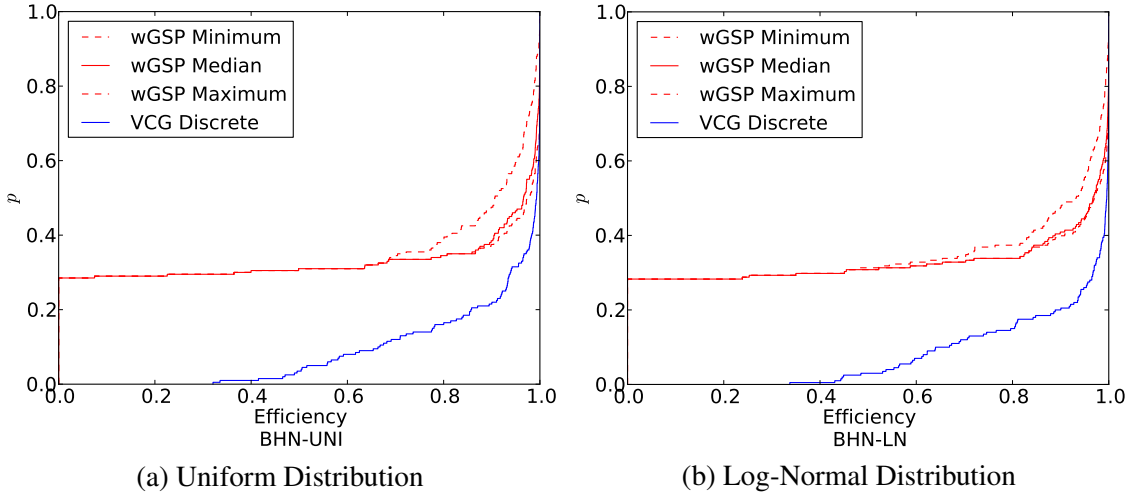


Figure 4.15: Empirical CDF of economic efficiency in BHN models.

be paid, since the externality he imposes on other bidders is to allow them to reach the higher-value lower positions.) A similar phenomenon can arise in lower positions as well: e.g., if exactly one advertiser values the second position (and by monotonicity, the first as well) at weakly more than one bid increment, then in any pure equilibrium all the other bidders bid zero. The second kind of efficiency failure involves mis-ordering the advertisers (e.g., placing the highest-valuation advertiser in some slot other than the highest), as we already saw in the V and EOS settings. However, the magnitude of inefficiency can be more than we saw in those models, because different positions have different conversion rates. In equilibria of EOS and V models, the high-valuation bidder might prefer to avoid the top position because the top positions' greater numbers of clicks are offset by disproportionately greater prices per click. In the BHN model, top-position clicks can also be too expensive, and this is compounded by the fact that top-position clicks are least likely to convert. Discretization can exacerbate this problem. Because per-click valuations for the top positions are small compared to lower positions, discretization has a greater effect on top positions, leading to inefficiencies due to rounding. Because high positions get the majority of clicks, this causes a large loss of welfare.

We now consider how different auction designs performed on BHN (see Figure 4.16). First, we observed that all position auctions designs were very inefficient ($\leq 69\%$), and substantially less efficient than discrete VCG (which achieved 91.6% and 90.5% efficiency in BHN-UNI and BHN-LN respectively.) wGSP was dramatically better than GFP; wGSP's worst-case equilibria were more efficient than GFP's best case. The comparison between uGSP and wGSP was less clear, because uGSP's efficiency varied much more across equilibria. In the case of BHN-UNI, this made uGSP's best-case equilibria slightly (but not significantly) better than wGSP's. In other distributions and for other criteria, uGSP was clearly worse.

Turning to revenue, we found that no auction design extracted more than 30% of the surplus, but that every position auction design was dramatically better at generating revenue than VCG. This finding has a

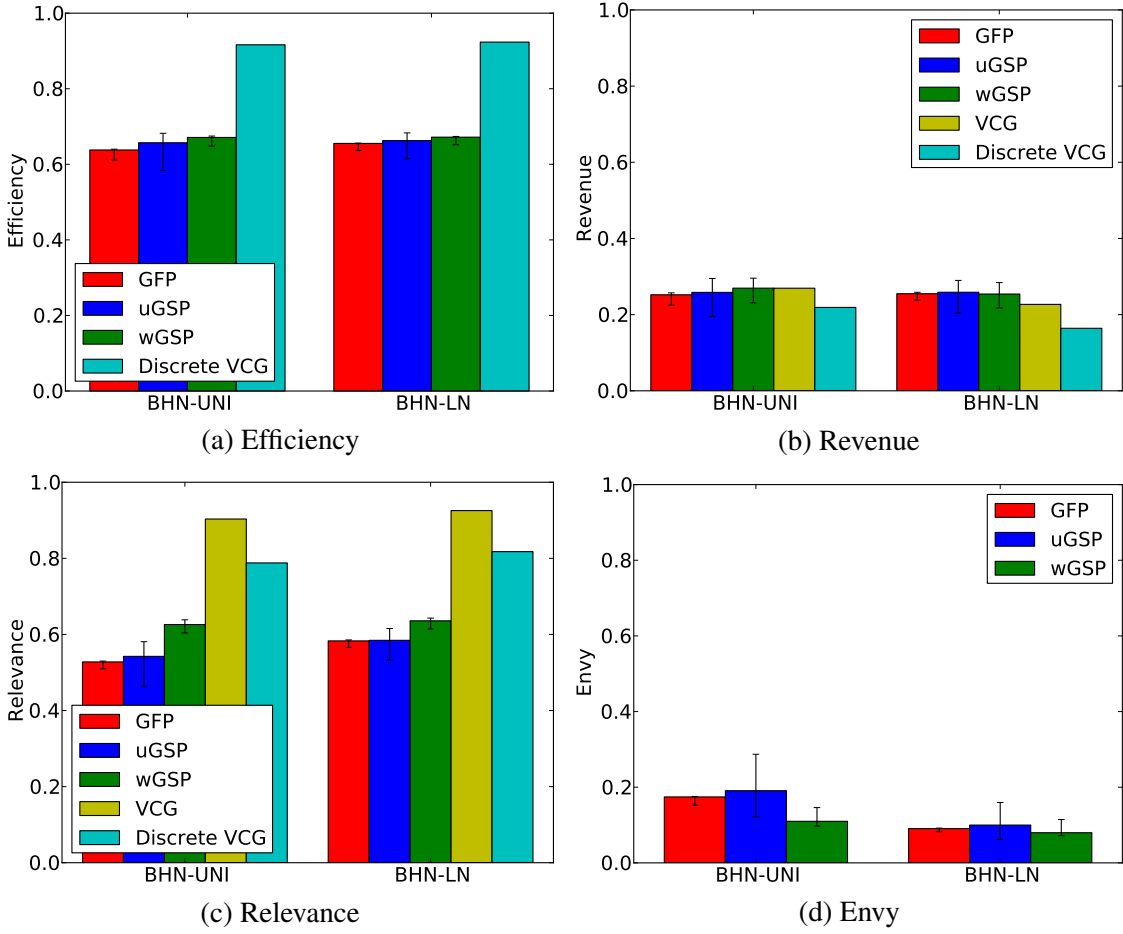


Figure 4.16: Comparing the average performance of different position auction types in BHN settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

straightforward explanation: in BHN models, conversion rates are higher in low positions. In VCG, an agent pays his externality, i.e., the amount he decreases the welfare of the lower-ranked agents by displacing them one position. In GSP, an agent pays the amount of the next highest bid, as if he had entirely removed the next highest bidder entirely. In EOS, V and BHN models, the VCG payment (for a given bid profile) is always weakly less than GSP, because the VCG payment takes into account that next highest agent gets a position of value. In BHN, the displaced agents are moved into positions that are likely to be closer to the value of pre-displacement positions (compared with EOS and V settings). We also observe that, as we saw before, revenue variation across equilibria was dramatically larger than revenue variation across position auction designs. Unlike the V model, under BHN uGSP did not provide dramatically more revenue than wGSP when values were lognormally distributed.

All position auction designs achieved dramatically worse relevance than VCG (both with and without

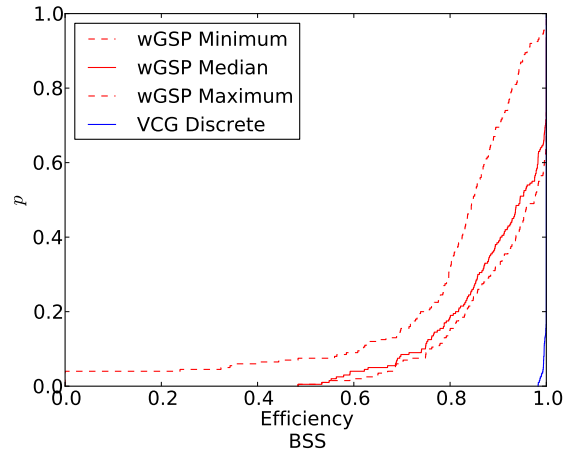


Figure 4.17: Empirical CDF of economic efficiency in BSS models

discretization). wGSP was clearly the best position auction in terms of relevance..

Envy varied substantially among both GSP designs, but nevertheless wGSP outperformed the other position auction designs in minimizing this metric.

The BSS Model

Next we investigated to the model of Benish, Sadeh and Sandholm [9]. This model is similar to the BHN model in that different advertisers can have different per-click valuations. However, in this case, valuations are “single-peaked” in the sense that each advertiser has a most preferred position and their per-click valuation decreases as the ad is moved further from that position. Recall that BSS is an unweighted model (i.e., all the advertisers have identical quality scores and click probabilities). Thus, wGSP and uGSP are equivalent in this model.

Question 4: Under BSS preferences, how often does wGSP have no efficient perfect-information Nash equilibrium? How much social welfare is lost in such equilibria?

We found that efficiency failures were very common under BSS, and could be very large, though zero-efficiency outcomes were rare and never happened in best-case equilibria (see Figure 4.17). This was due in part to randomized tie-breaking. When none of the agents values the top slot, all the agents can bid zero in equilibrium. However, they can also all bid a small but nonzero amount and tie for top bid; this leads them all to get more valuable intermediate positions often enough to justify the cost.

Comparing auction designs (see Figure 4.18), we observed that uGSP was significantly more efficient than GFP, but fell well behind discrete VCG. In revenue, uGSP and GFP both varied substantially across equilibria,

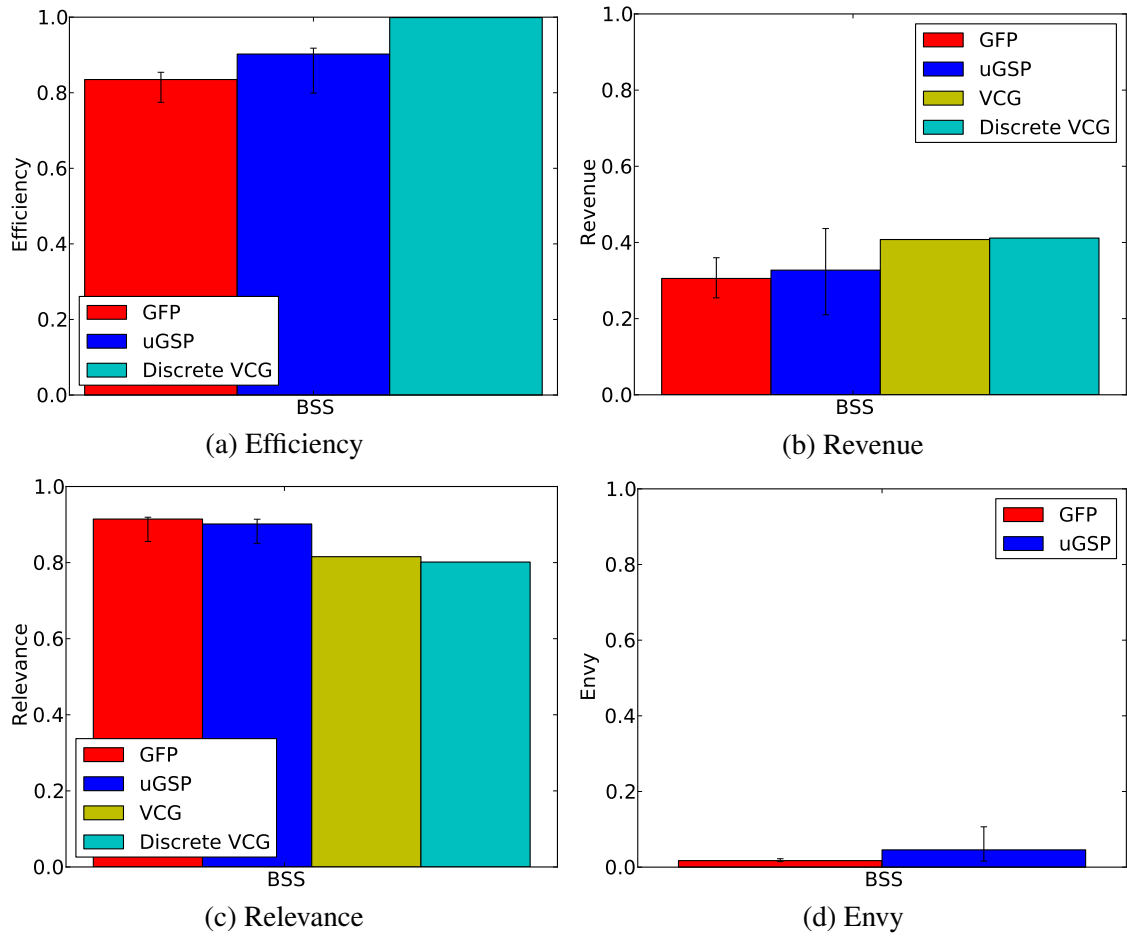


Figure 4.18: Comparing the average performance of different position auction types in BSS settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

and both were significantly worse than VCG in their median equilibria. uGSP slightly outperformed VCH in best-case equilibrium, but not significantly so. Both uGSP and GFP achieved more relevant outcomes than VCG, largely due to the tying effect described earlier, where agents were often sold clicks that they did not particularly value. Both auctions had little envy in absolute terms, though GSP was worse than GFP.

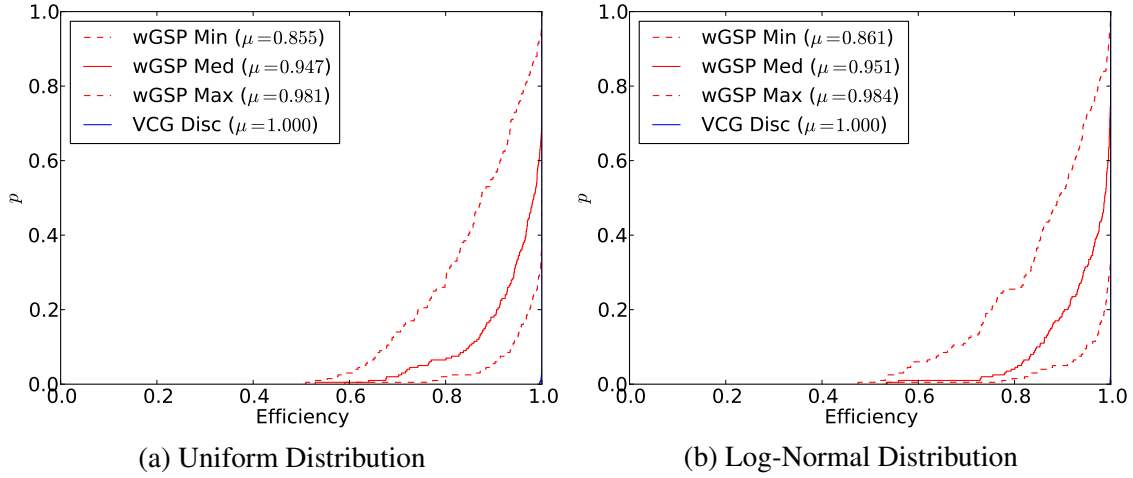


Figure 4.19: Empirical CDF of economic efficiency in the cascade models.

4.5.4 Externality Models

We now consider Cascade, Hybrid and GIM, three models in which bidders can care about the positions awarded to other agents.

The Cascade Model

Cascade is the simplest of our models involving externalities. Cascade settings are similar to V settings in that each bidder has a quality (proportional to click probability) and a per-click valuation. The difference is that each advertiser also has a continuation probability: the probability that a user looks at subsequent ads. Thus, in both models, lower positions are less valuable because they receive fewer clicks. However, in cascade models, this reduction depends on what is shown in higher positions.

Question 5: Under cascade preferences, how often are there low-efficiency equilibria? In cases where low-efficiency equilibria exist, are these the only equilibria? When agents play only undominated strategies, how much revenue can wGSP generate? How do the modified weights affect wGSP's efficiency?

We found that inefficiency was common, though the magnitudes of efficiency losses were smaller than we expected (see Figure 4.20). We found that wGSP's efficiency was always greater than 50% and was almost always greater than 80%. In contrast, Giotis and Karlin [39] showed that the price of anarchy is k and thus, for $k = 5$ there exist instances (which we evidently did not sample) with as little as 20% efficiency.¹¹ The gap between best- and worst-case equilibria was typically quite large, though a few instances had poor ($\sim 60\%$)

¹¹Although they studied the more general model that we refer to as hybrid, the construction of their worst-case example uses a part of the hybrid model space that is also consistent with the more restrictive cascade model.

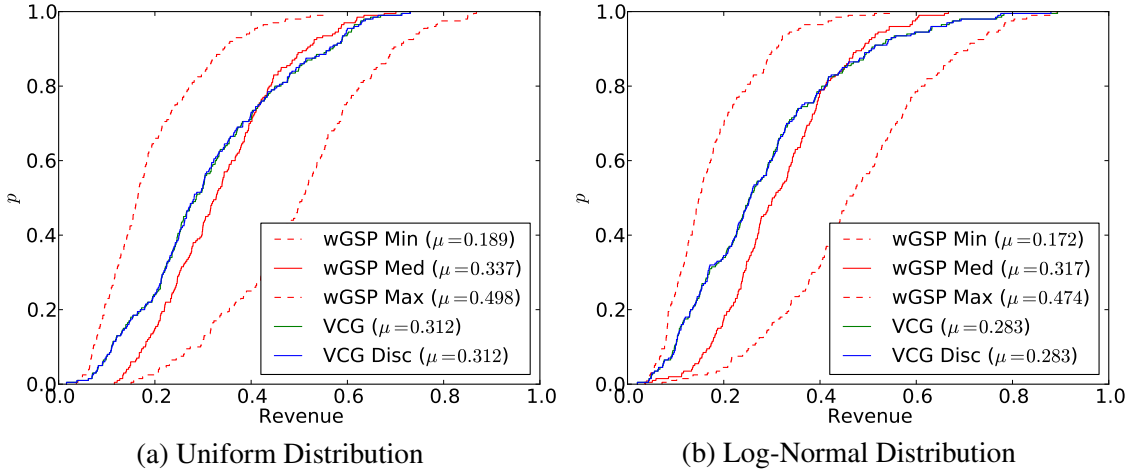


Figure 4.20: Empirical CDF of revenue in the cascade models.

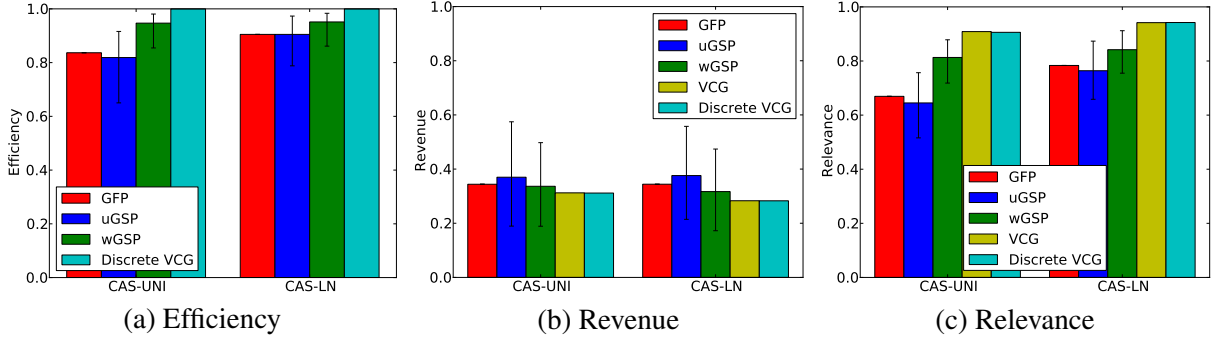


Figure 4.21: Comparing the average performance of different position auction types in cascade settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

efficiency even in the best case.

wGSP’s revenue varied substantially across equilibria (see Figure 4.20), with the best-case equilibrium generating at least 2.5 times as much revenue as the worst case.

Comparing position auctions, we observed relative performance similar to that under the no-externality models (see Figure 4.21). First, we found that wGSP was the most efficient position auction design. Surprisingly, wGSP’s best-case equilibria tended to be close to fully efficient (at least $\sim 95\%$). However, variation across equilibria was greater than in no-externality settings like EOS and V. Both uGSP and wGSP varied substantially in their revenue across equilibria, with wGSP’s median equilibrium achieving slightly more revenue than VCG. Like in the V model, wGSP was roughly comparable to uGSP in terms of revenue when values were uniformly distributed, and noticeably worse when values were log-normally distributed. Concerning relevance, we found that wGSP was clearly superior to other position auction designs, but not

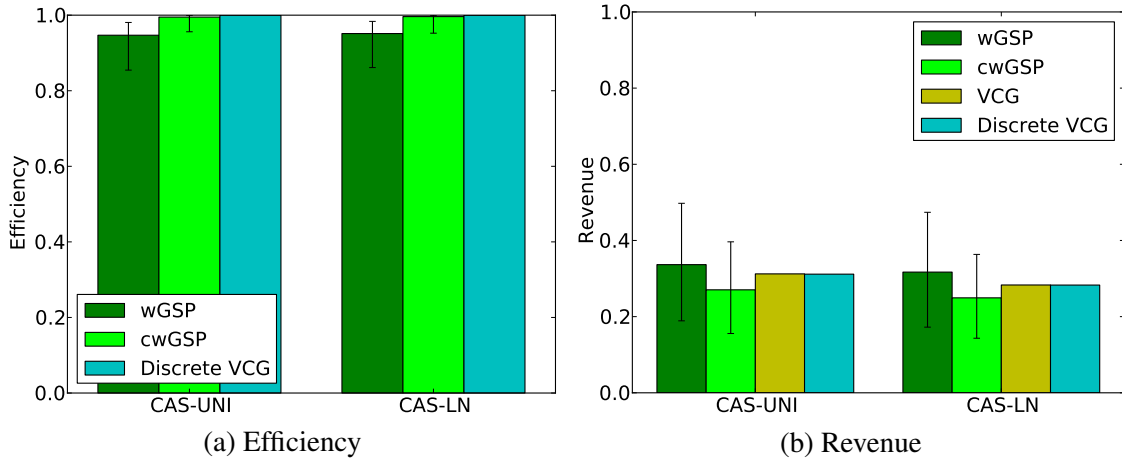


Figure 4.22: Comparing the average performance of wGSP and cwGSP in cascade settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

quite as good as VCG. Recall that envy is not well defined in settings with externalities; thus, we do not discuss it here or in what follows.

wGSP with Cascade-Specific Weights

To address some of the shortcomings of wGSP in cascade, Gomes *et al* proposed an alternative way of calculating advertiser qualities for wGSP. In their alternative weighting, an advertiser’s weight is equal to her top-position click probability divided by the probability that a user will stop looking at ads after seeing hers (i.e., one minus her continuation probability). This simple reweighting scheme has the nice property that an advertiser with a continuation probability of one will always be ranked first, a desirable property given that the efficient ranking will always rank such bidders first. Gomes *et al* also showed that this weighting gives rise to a revenue-optimal equilibrium. (There are two important caveats about this equilibrium. (1) It is only revenue-optimal among mechanisms that always allocate every position; mechanisms that use reserve prices can get strictly more revenue. (2) It might not be rationalizable, requiring some agents to play the weakly dominated strategy of bidding strictly above their own valuations.)

We experimented with this alternative mechanism, which we call cwGSP, and found that it was dramatically more efficient than wGSP (see Figure 4.22). However, we also found that it was noticeably worse than wGSP in terms of revenue. Thus, we conclude that Gomes *et al*’s revenue claims about cwGSP stem mainly from their unusual equilibrium-selection criteria.

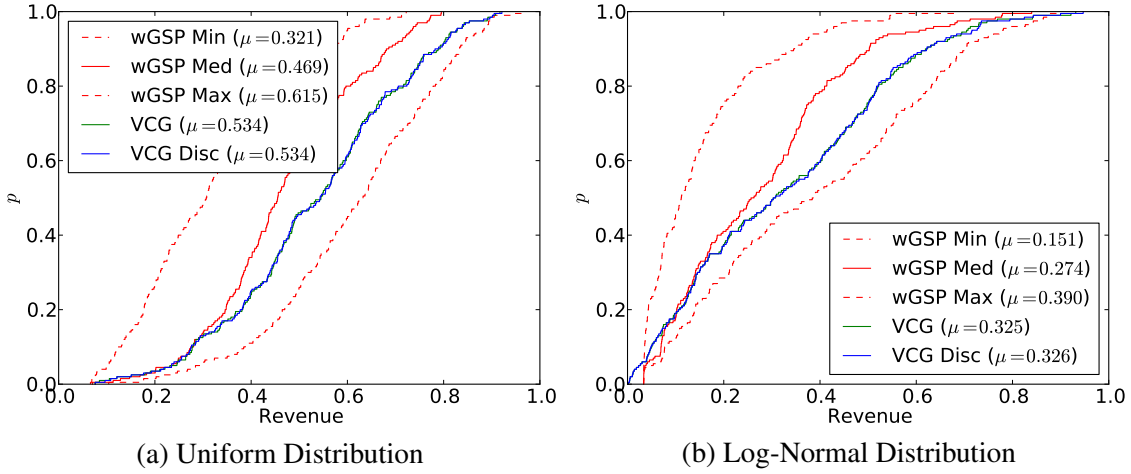


Figure 4.23: Empirical CDF of revenue in the hybrid model.

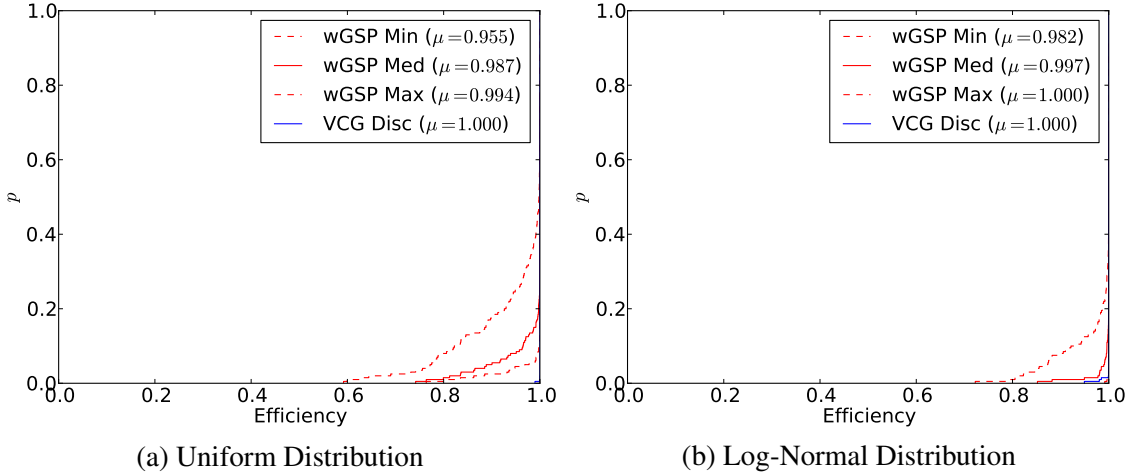


Figure 4.24: Empirical CDF of economic efficiency in the hybrid model.

The Hybrid Model

The hybrid model generalizes both the V and cascade models; lower positions can get fewer clicks either due to the number of higher-placed ads (as in V) or to the content of those ads (as in cascade). Hybrid settings can have a very large price of stability and an even larger price of anarchy [56].

Question 6: With hybrid preferences, how much social welfare is lost? On typical instances (as opposed to worst-case ones), how much difference is there between best- and worst-case equilibria?

We again found that wGSP's revenue varied substantially across equilibria (see Figure 4.23). However, we were surprised to find that while wGSP's efficiency was quite low ($\sim 60\%$) in a few games, wGSP was often

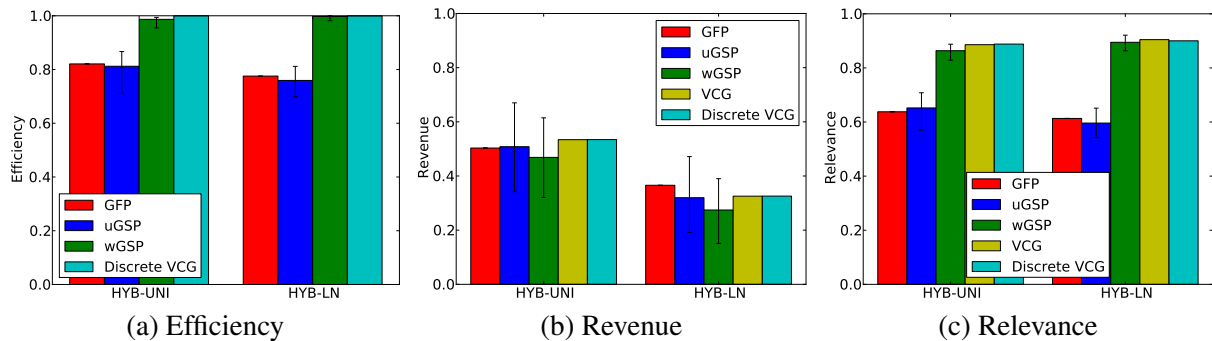


Figure 4.25: Comparing the average performance of different position auction types in hybrid settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

efficient, even in its worst-case equilibria (see Figure 4.24). This result stands in contrast with our previous findings on cascade. The very small difference between best- and worst-case equilibria did not arise because games had few Nash equilibria: wGSP still had many Nash equilibria in just about every game. In the next subsection, we compare all three externality models and investigate this anomaly more deeply.

Comparing the different position auction designs (see Figure 4.25), we again found that wGSP was the most efficient and produced the most relevant results. Concerning revenue, we found that both uGSP and wGSP varied substantially across their equilibria, but that uGSP was consistently better than wGSP. wGSP substantially outperformed GFP and uGSP in terms of relevance.

The GIM Model

Finally, we turn to the model of Gomes, Immorlica and Markakis, which is the most general of our externality models. In the GIM model, every bidder has a quality score, which corresponds to her probability of getting a click in the top position, but the click probability in lower positions can depend arbitrarily on which advertisers are shown in higher positions as long as this probability weakly decreases with position. Gomes *et al* conjectured that this model was similar to cascade, which motivated our questions.

Question 7: Are the efficiency and revenue achieved by wGSP substantially different under the cascade and GIM models?

Recall that in the cascade model, revenue and efficiency both varied substantially across equilibria, and that the majority of games exhibited moderately large inefficiency ($\sim 90\%$) in worst-case equilibrium. In GIM, we observed similar effects of similar magnitude (see Figures 4.26 and 4.27). We found it surprising that GIM, which generalizes the hybrid model (which in turn generalizes cascade), should have outcomes that are more similar to cascade than to hybrid. To explain the anomaly, we looked for features that were common

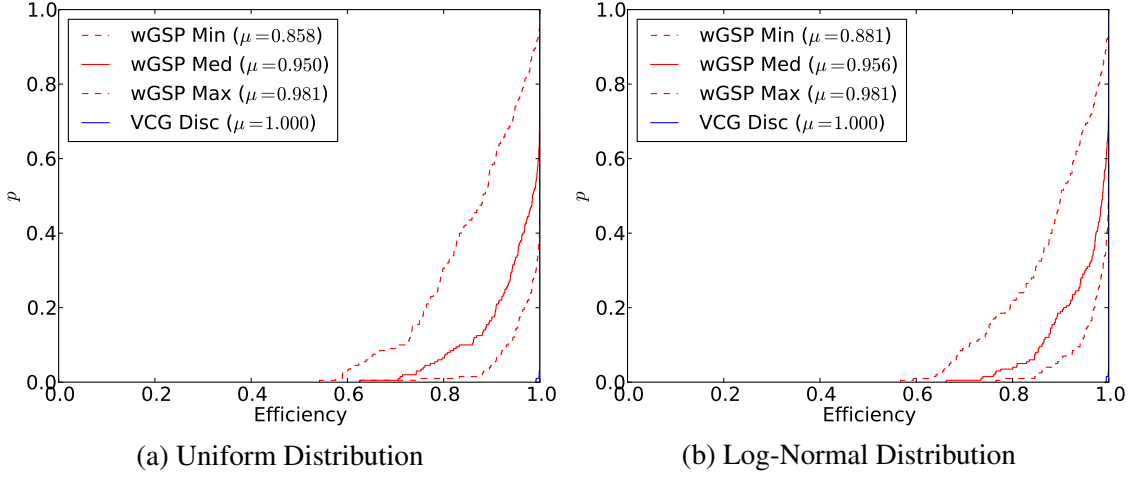


Figure 4.26: Empirical CDF of economic efficiency in the GIM model.

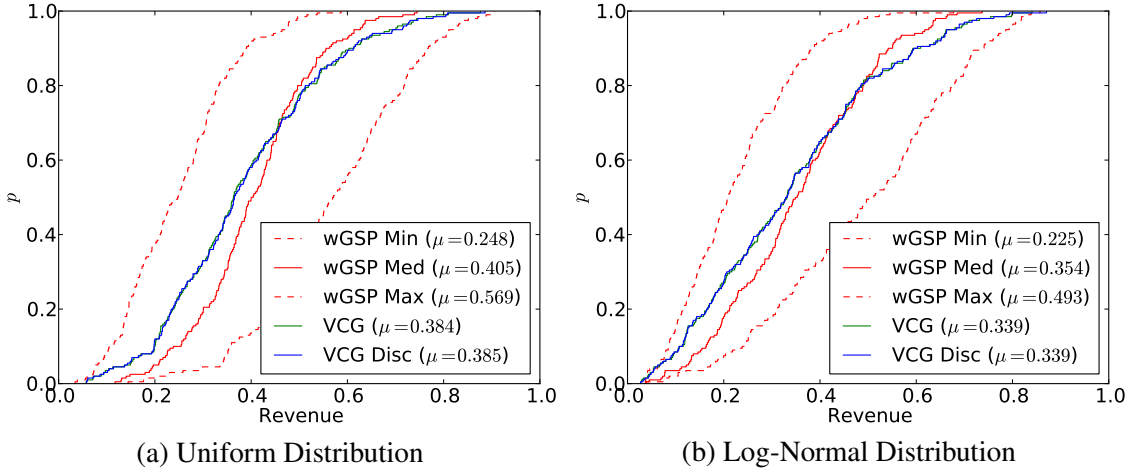


Figure 4.27: Empirical CDF of revenue in the GIM model.

to cascade and GIM, but not to hybrid. There are many such features (e.g., in wGSP’s equilibrium rankings, the effect of position on CTR is much greater under hybrid). In the end we concluded that the feature that best explains our findings is the relative value of the top position—which is higher in hybrid settings—because in all three models, instances where this fraction was large tended to have very good worst-case efficiency in wGSP (see Figure 4.28). This correlation was moderately strong ($\rho > 0.4$ using Spearman’s rank correlation) and highly significant ($p < 0.01$ with Bonferroni correction) for every distribution except BHN-LN, in which the majority of equilibria achieved almost perfect efficiency.

Lastly, we compare different position auctions in the GIM setting (see Figure 4.29). As under other externality models, we found that wGSP produced more efficient and relevant rankings than GFP and uGSP, but performed significantly worse than VCG. Again, revenue was ambiguous, with GSP revenue varying

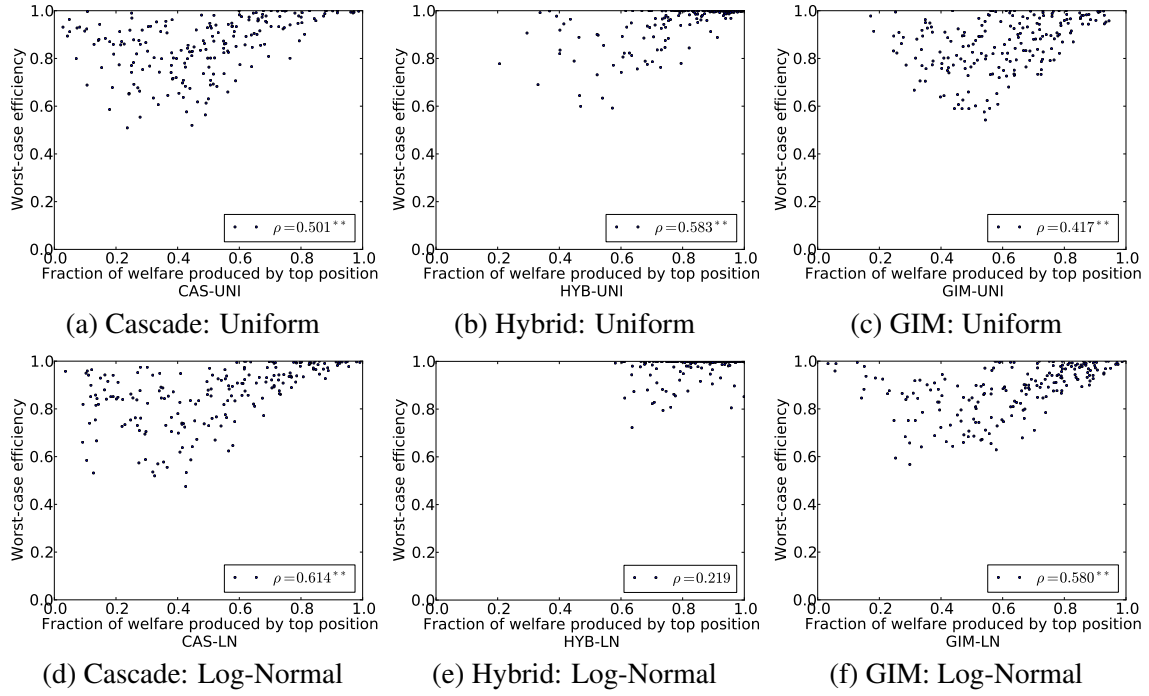


Figure 4.28: wGSP tended to have good worst-case efficiency when the top position produced the majority of the surplus (in an efficient allocation). Thus, in hybrid distributions, where this occurred extremely frequently, wGSP tended to have better worst-case efficiency than in cascade or GIM.

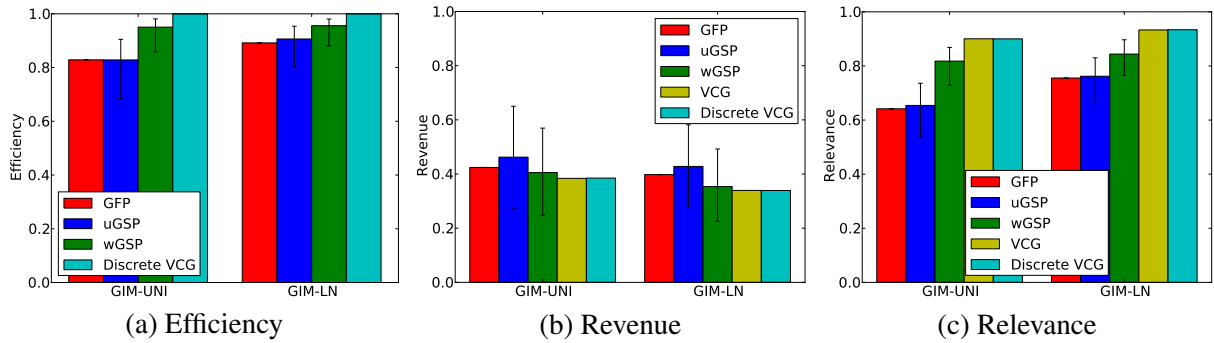


Figure 4.29: Comparing the average performance of different position auction types in GIM settings. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

substantially across equilibria and all median revenues fairly close to VCG’s revenue.

4.6 Sensitivity Analysis

Computational analysis techniques oblige us to make concrete choices about discretization and the sizes of problems that we study. This section examines the extent to which our findings varied under five such details about the position auction setting that we have held constant until now. (1) Although real-world auctions are discrete, the coarseness of the discretization can vary greatly from keyword to keyword or advertiser to advertiser, depending on the size of each advertiser’s valuation relative to the bid increment. Thus, it is important to understand how sensitive our findings are to the scale of the discretization. When bids and payments are discrete, issues also arise that are not significant in continuous games, such as (2) how ties will be broken and (3) how payments will be rounded or aggregated. Furthermore, computational analysis requires setting many parameters that can often be left unspecified in a theoretical analysis, such as (4) the number of bidders and (5) the number of slots for sale. We chose two widely studied distributions to consider in our scaling studies: one simple (V) and one with externalities (cascade). In both cases we used the more realistic valuation distribution, log-normal.

4.6.1 Sensitivity to Bid Increment Size

First, we considered the problem of increment size. In addition to V-LN and CAS-LN we also considered the BHN-UNI distribution, because we found BHN’s interaction with increment size particularly interesting. Recall that earlier we found that the single-increment reserve price was sufficient to cause dramatic efficiency loss. We anticipated that as increment size decreased, this would occur less frequently. For each distribution, we varied the number of bid increments from 5 to 40, at each step generating 100 instances and computing all relevant metrics (see Figures 4.30 and 4.31). In the V and cascade distributions, we found that (with one exception) the relative performance of different mechanisms became steady once k was greater than 10, and that absolute performance became stable once k was greater than 20. The one exception was the GFP auction; in both distributions, GFP’s performance (both in relative and absolute terms) varied dramatically as k grew. BHN’s behavior was particularly interesting. As the number of increments increased, the efficiency and relevance of all auctions dramatically increased (because the problem of unallocated slots became less common). Worst-case envy and revenue tended to get worse as the number of increments increased, while median- and best-case revenue and envy remained fairly flat.

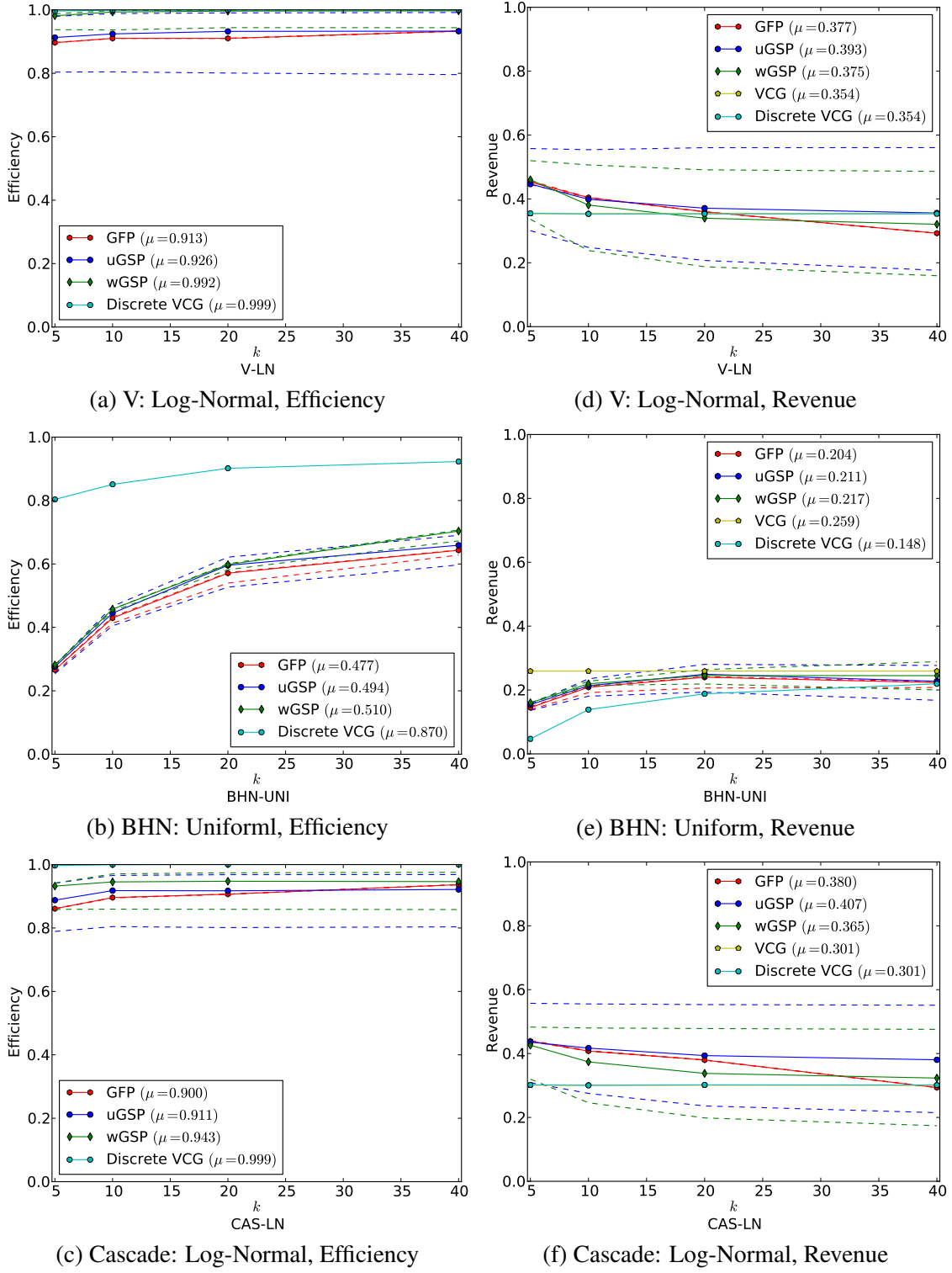
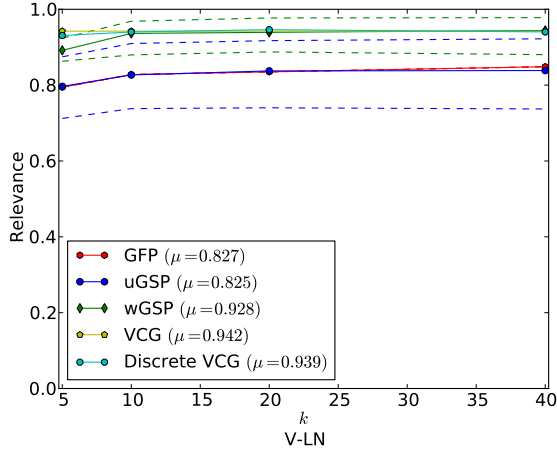
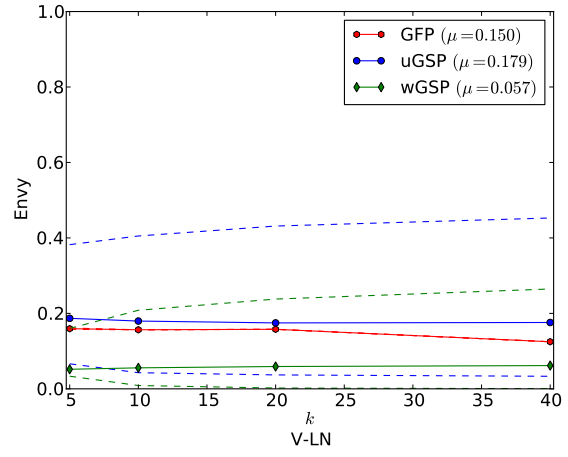


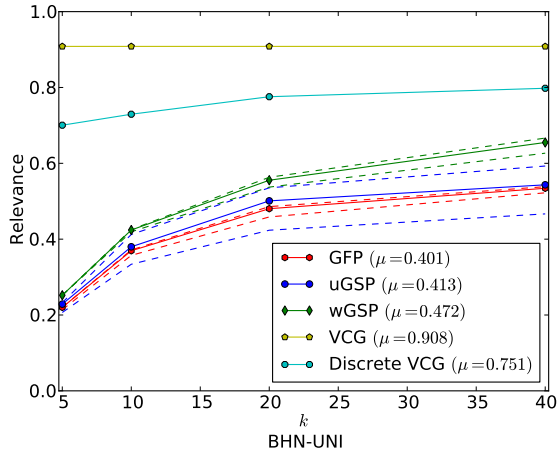
Figure 4.30: Comparing different auction designs as the number of bid increments varies.



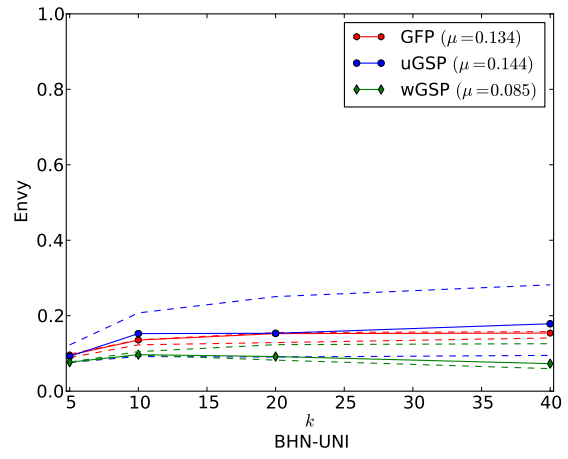
(g) V: Log-Normal, Relevance



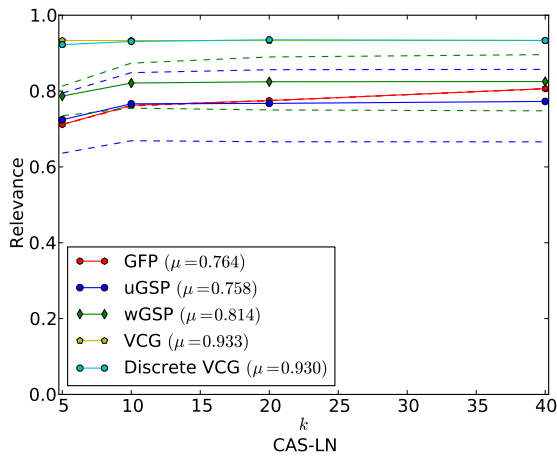
(j) V: Log-Normal, Envy



(h) BHN: Uniform, Relevance



(k) BHN: Uniform, Envy



(i) Cascade: Log-Normal, Relevance

Figure 4.31: Comparing different auction designs as the number of bid increments varies (continued).

4.6.2 Sensitivity to Tie-Breaking Rules

Discretization also invites the possibility that ties will occur between bids with positive probability. Thus, while tie-breaking rules are typically unimportant to the analysis of auctions with continuous action spaces, they can be significant in discrete settings. Observe, however, that this problem does not arise under most of our distributions; although bids are integral, each bid is weighted by a real number, with the property that for any pair of weights the probability of their ratio being integral is zero. Thus, we only needed to examine sensitivity to tie breaking in the two models that did not have weights: EOS-LN and BSS-UNI.

Broadly, we found that tie breaking made almost no difference in the EOS-LN distribution; however, lexicographic tie breaking consistently decreased performance in every metric in the BSS distribution (see Figure 4.32). This problem arose when multiple agents preferred a lower slot to a higher slot, and strictly preferred not to be shown in the high position. With random tie breaking, each agent could win their more preferred slot often enough to justify paying a high price. However, when ties are broken lexicographically, some agent must pay a higher price and get a less desirable slot, inducing him to deviate. Thus, random tie breaking achieved much higher relevance, selling clicks even in cases where continuous VCG did not.

4.6.3 Sensitivity to Rounding Rules

The last problem discretization poses for an auctioneer is the question of how to round prices in wGSP auctions. Because each bidder's integral bid is scaled by an arbitrary real constant, the next-highest bid might not correspond to any integer price. So far, we have assumed that prices are rounded up (i.e., that a bidder must pay the minimum amount that she could bid to win the position she won). In this section we also consider rounding down, rounding to the nearest integer, and rounding up plus 1 increment (which was used by Yahoo! [30]). Here our experiments consisted of 100 instances each from the V-LN and CAS-LN distributions (see Figure 4.33). We found, somewhat unsurprisingly, that more "aggressive" rounding rules (i.e., rules that favor higher prices) tended to produce higher revenue regardless of the distribution and the equilibrium-selection criterion (the difference between the best and worst rounding rules ranged from 10% to 25%), with a corresponding improvement in envy. More surprisingly, we found that rounding rules had noticeably smaller effects on economic efficiency and relevance (not more than 2.5%). Thus, the aggressive rounding rules used in practice appear to increase revenue at very little cost in terms of ad quality.

4.6.4 Sensitivity to the Number of Bidders

To study whether our findings were sensitive to the number of bidders, we used the same two distributions, V-LN and CAS-LN. For each, we solved 100 instances, and studied how each auction performed as the number of bidders varied from two to ten. Throughout, we assumed that there was no artificial limit on supply;

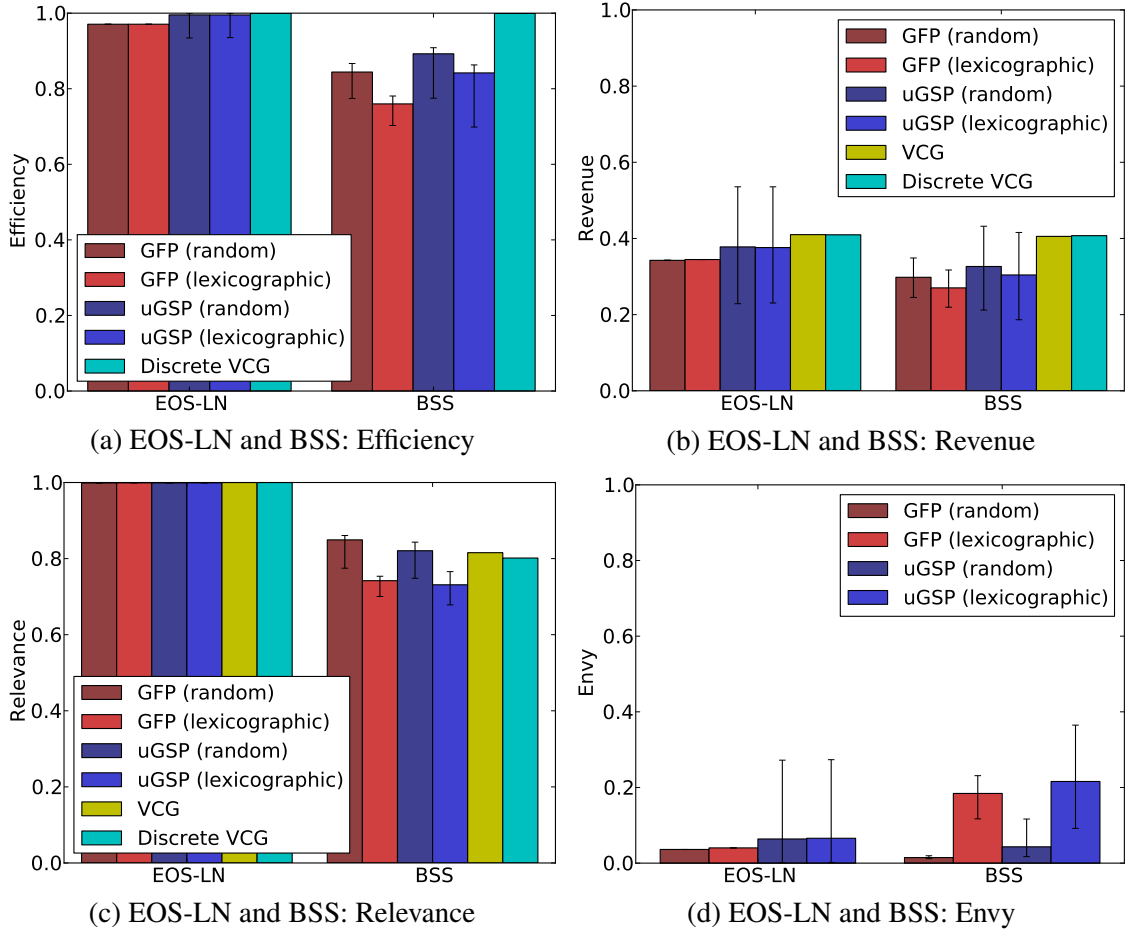


Figure 4.32: How tie breaking affects auction outcomes. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

i.e., the search engine would allocate a slot to every advertiser who bid more than zero. (We investigate the impact of this assumption next.) We found that the relative ranking between position auction variants remained consistent as the number of bidders varied (see Figures 4.34 and 4.35). As the number of bidders increased, all position auctions became progressively less efficient and relevant (although in V-LN, wGSP experienced less such decrease than the other position auctions), and all auctions (including VCG) saw increasing normalized revenue (i.e., fraction of the surplus going to the seller).

4.6.5 Sensitivity to the Number of Slots

So far, all of our experiments have assumed that the search engine can allocate space to every single advertiser (as, indeed, is often the case). Here we consider what happens when the supply of slots is limited. This is

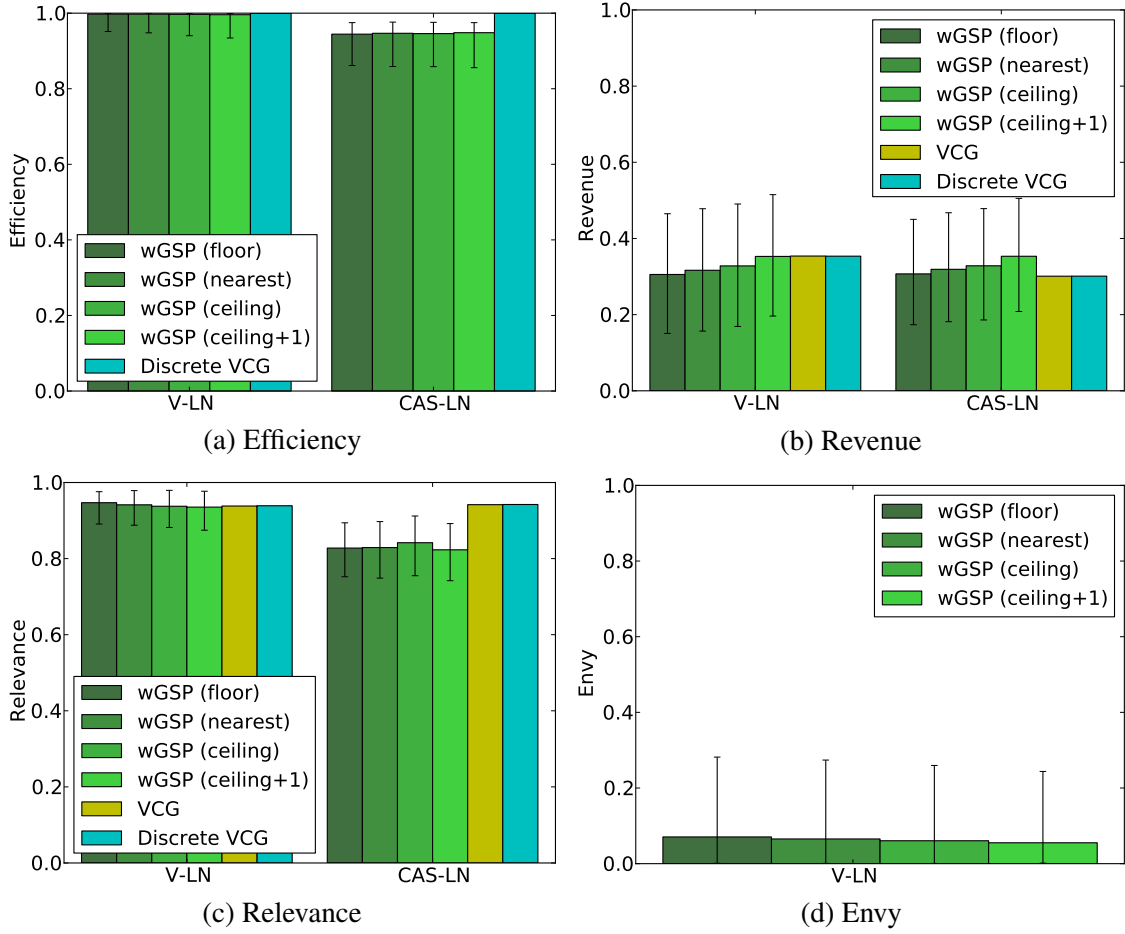


Figure 4.33: How price rounding affects auction outcomes. The “whiskers” in this plot indicate equilibrium selection effects: the bar is the median equilibrium, while the top and bottom whiskers correspond to best- and worst-case equilibria.

particularly important for cascade and hybrid models, under which both price of anarchy and computational complexity are known to depend on the total supply (m) [56]. We again considered V-LN and CAS-LN, in each case varying the number of slots from one to five.

We present our results in Figures 4.36 and 4.37. Our first striking observation is that the case of $m = 1$ (i.e., selling a single good to “quality-weighted” bidders) is distinctly different from $m > 1$. When $m = 1$ there is almost no variability in the allocation across different auctions, and therefore likewise no variability in efficiency and relevance. (The one exception is relevance in wGSP: if two bidders have quality-weighted valuations within an increment of each other but different click-through rates, either one could win in equilibrium.) However, in GSP auctions revenue (and therefore envy) was nevertheless extremely variable across equilibria. (Indeed in the continuous case, any outcome having revenue between zero and the VCG payment is possible in conservative equilibrium.) When two or more slots were available, competition for the

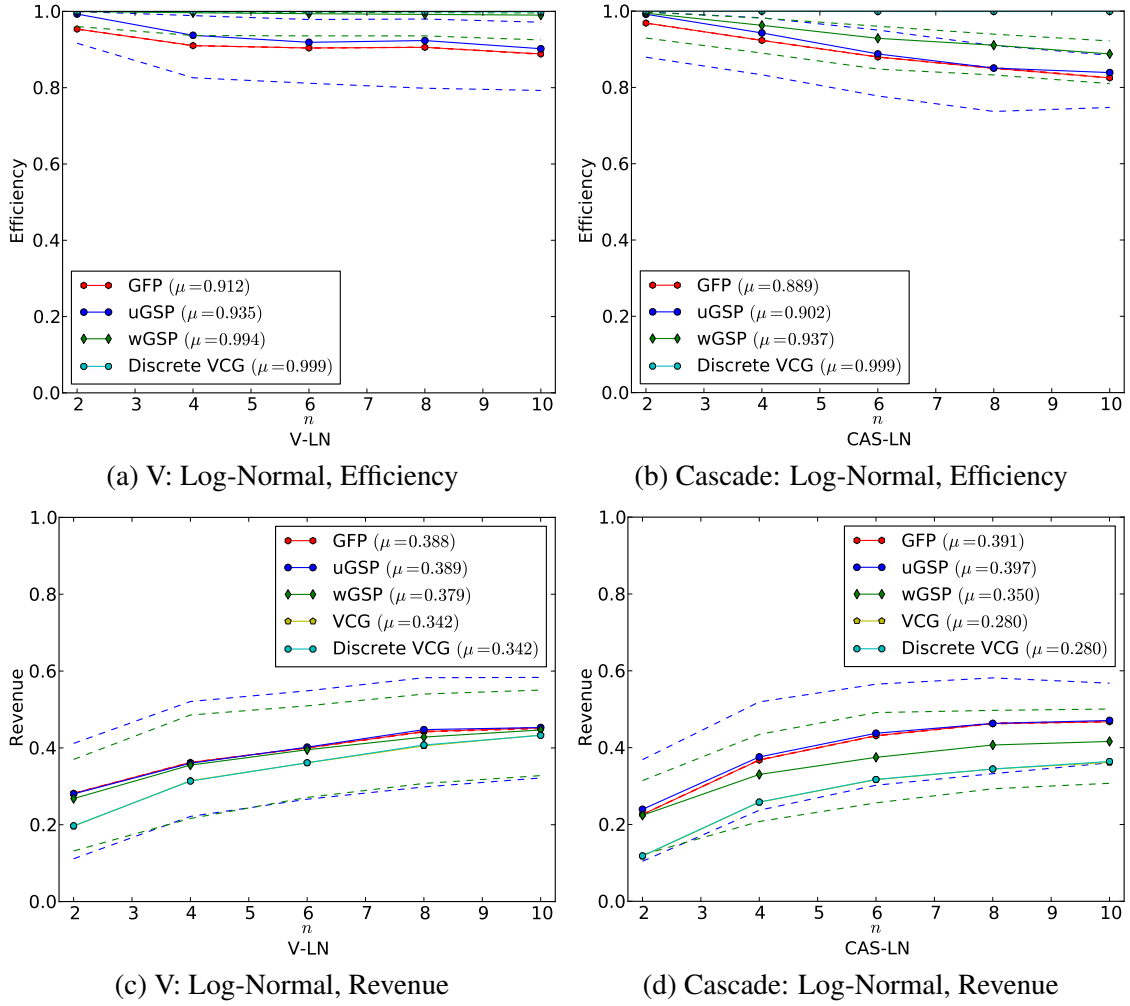


Figure 4.34: Comparing different auction designs as the number of agents varies.

lower slots dramatically decreased the range of prices that the top bidder could be charged, and hence the range of possible revenues. However, these lower slots could attract a bidder who should (in the efficient allocation) have appeared in a high position. Thus, with two or more slots, efficiency (and relevance) became more variable for all position auction designs. The relative performance of all auctions remained consistent, with wGSP being slightly worse than VCG and substantially better than uGSP and GFP. Revenue was a different story: as m increased, all auctions extracted a smaller fraction of the possible surplus because the availability of lower positions decreased competition for the higher positions. However, in GSP auctions, this decline occurred more gradually than in GFP and VCG.

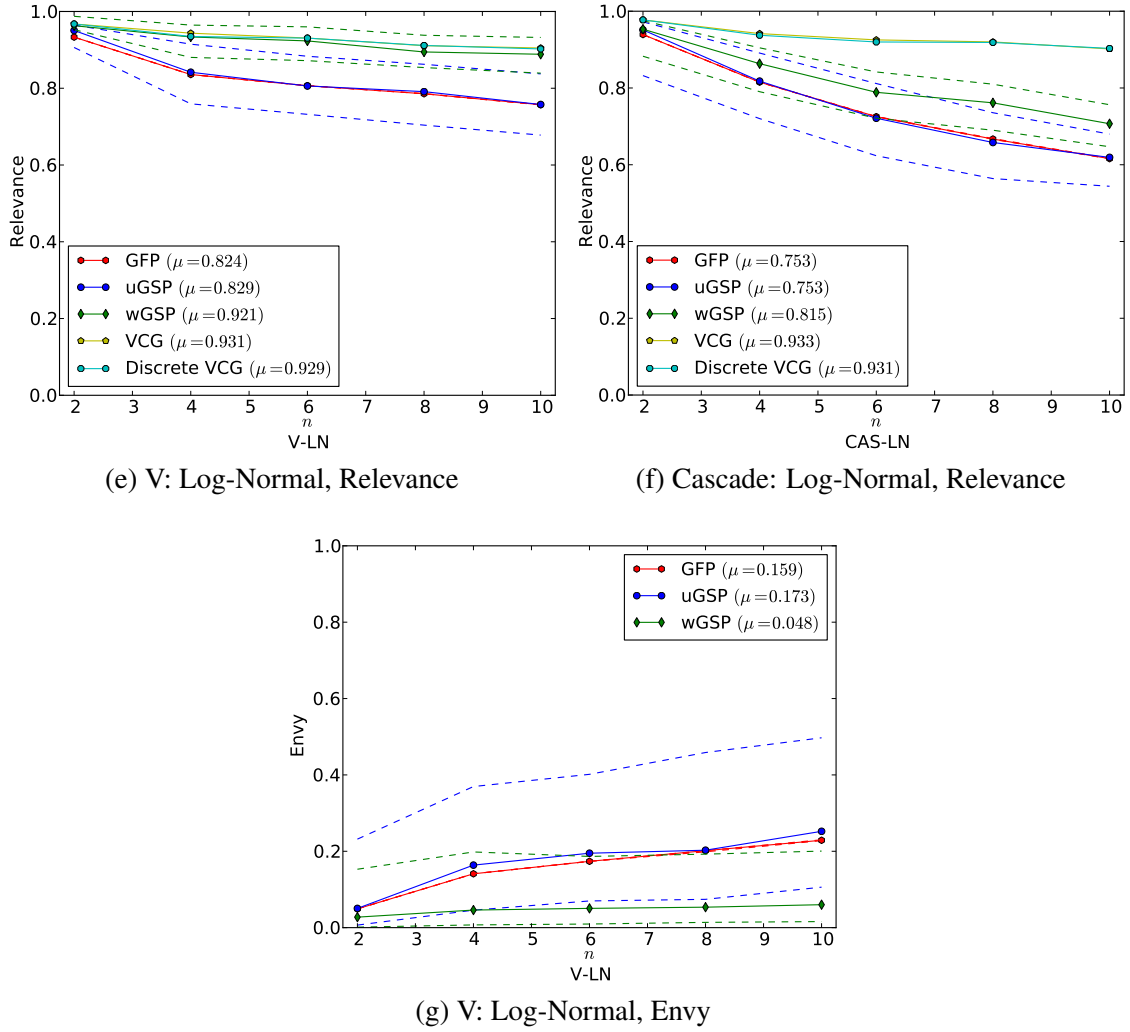


Figure 4.35: Comparing different auction designs as the number of agents varies (continued).

4.7 Conclusions and Future Work

This chapter demonstrates the feasibility of computational, rather than theoretical, mechanism analysis, applying algorithms for equilibrium computation to address a wide range of open questions about position-auction games. The main technical obstacle we faced was representing such games in a computationally usable form. We accomplished this by identifying encoder algorithms that take as input a position-auction setting (based on a wide variety of models drawn from the literature) and the parameters of the position auction (e.g., pricing rule, tie-breaking rule) and produce as output an action-graph game (AGG). These encoder algorithms, when combined with state-of-the-art equilibrium-finding algorithms, allow us to very quickly compute exact Nash equilibria of realistic position-auction games, and hence to provide quantitative statistical answers to many open questions in the literature. For example, where existing research into the

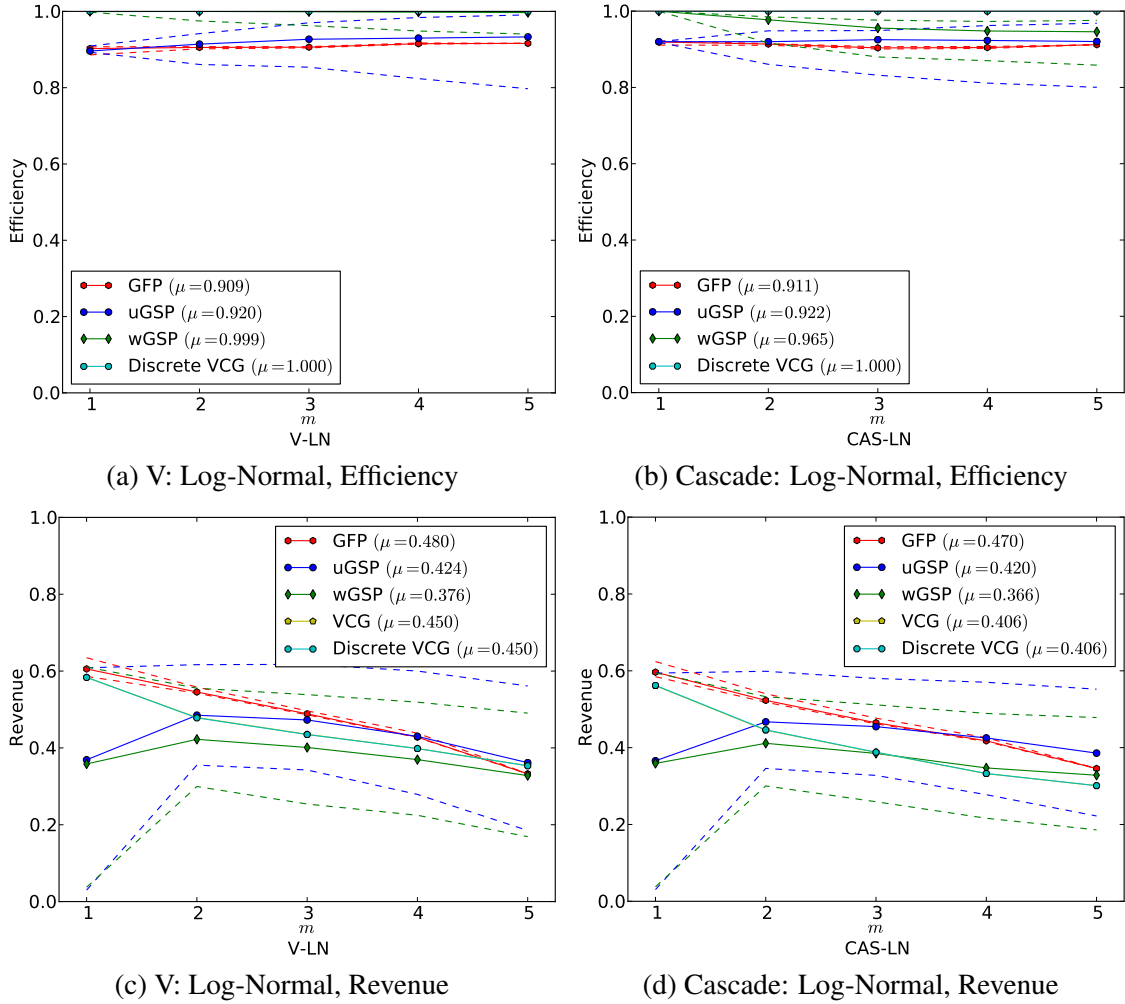


Figure 4.36: Comparing different auction designs as the number of slots varies.

widely-studied V and EOS models has focused on locally envy-free Nash equilibria, we found that such equilibria are a small minority among the set of equilibria, and often do not exist when bids are restricted to integer increments. These two sets also had interesting qualitative differences: while every envy-free equilibrium is known to yield more revenue than VCG, we found that the majority of Nash equilibria yield less revenue than VCG. These techniques also allowed us to make direct, apples-to-apples comparisons between different auction designs—varying the auction while holding the set of bidders and the equilibrium-selection criteria constant—yielding some striking results. In particular, while wGSP is known to be inefficient under many models of bidder valuations, we found that it was the most efficient position auction under nearly every model and valuation distribution, and often close to fully efficient in expectation even in models where its worst-case efficiency is very poor. Similarly, wGSP tended to outperform other position auction designs in terms of relevance and envy. On the other hand, we found that different position auction mechanisms could not easily be distinguished in terms of their revenues, with relative performance varying dramatically across

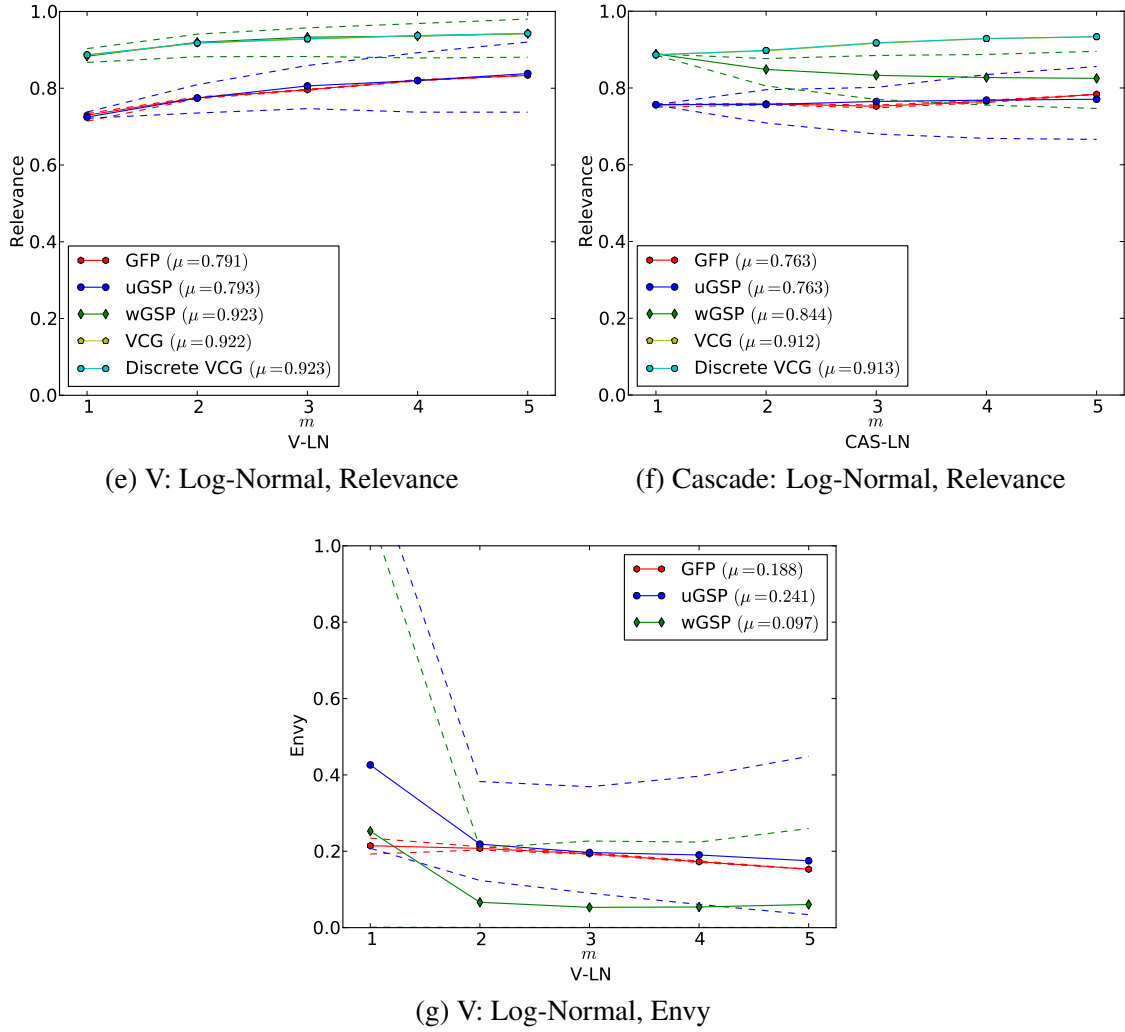


Figure 4.37: Comparing different auction designs as the number of slots varies (continued).

models, distributions, and equilibria.

Our techniques for computational mechanism analysis are applicable to many other open problems in position auctions. One very general direction is using computational mechanism analysis to facilitate mechanism design (e.g., searching through the space of reserve prices and quality-score distortions in order to find a position auction variant that optimizes some objective such as revenue or relevance, as indeed we do in Chapter 5). Other promising ideas include investigating nonlinear value for money (representing bidders who are not risk neutral, or who have budgets or return-on-investment targets), and Bayesian games. We also expect that it would be possible to extend our encoders to deal with richer auction rules (e.g., auctions where bidders specify budgets, or auctions with greater expressiveness as in [9]), or to encompass strategic interactions between multiple heterogeneous auctions (e.g., arising due to advertisers using broad match, or location targeting, or having a single budget that spans a multi-keyword campaign).

4.8 Summary Tables and Statistical Comparisons

In this section we provide detailed numerical summaries of our main experiments, as well as statistical comparisons between different auctions. For each pair of auctions A, B and metric (e.g., efficiency), we first test whether A is robustly superior to B , meaning that A 's worst case is significantly better than B 's best case. (This is signified by \dagger .) Next, we test a looser condition, whether A is better than B up to the limits of equilibrium selection, meaning that A is significantly better than B when comparing best case to best case, median to median and worst case to worst case. Next, we test whether or not one auction's performance "spans" the other, denoted by \subseteq . That is, $A \subseteq B$ indicates that A 's worst-case performance is significantly better than B 's, but that B 's best-case performance is significantly better than A 's. \sim is used to indicate that none of these conditions is true with sufficient statistical confidence.

Mechanism	Worst	Median	Best	n
GFP	0.979 ($\sigma = 0.011$)	0.979 ($\sigma = 0.011$)	0.979 ($\sigma = 0.011$)	161
uGSP	0.958 ($\sigma = 0.039$)	0.998 ($\sigma = 0.003$)	1.000 ($\sigma = 0.001$)	200
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200
	GFP	uGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	\sim
VCG				$\geq \dagger^{**}$
dVCG				

Table 4.2: Comparing Efficiency (EOS-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.487 ($\sigma = 0.168$)	0.487 ($\sigma = 0.168$)	0.487 ($\sigma = 0.168$)	161
uGSP	0.365 ($\sigma = 0.172$)	0.498 ($\sigma = 0.161$)	0.635 ($\sigma = 0.170$)	200
VCG	0.572 ($\sigma = 0.190$)	0.572 ($\sigma = 0.190$)	0.572 ($\sigma = 0.190$)	200
VCG discrete	0.573 ($\sigma = 0.191$)	0.573 ($\sigma = 0.191$)	0.573 ($\sigma = 0.191$)	200
	GFP	uGSP	VCG	dVCG
GFP		\sim	\sim	\sim
uGSP			\supseteq^{**}	\supseteq^{**}
VCG				\sim
dVCG				

Table 4.3: Comparing Revenue (EOS-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.021 ($\sigma = 0.013$)	0.021 ($\sigma = 0.013$)	0.021 ($\sigma = 0.013$)	161
uGSP	0.003 ($\sigma = 0.005$)	0.092 ($\sigma = 0.083$)	0.282 ($\sigma = 0.230$)	200

	GFP	uGSP
GFP		\sim
uGSP		

Table 4.4: Comparing Envy (EOS-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.971 ($\sigma = 0.010$)	0.971 ($\sigma = 0.009$)	0.971 ($\sigma = 0.010$)	176
uGSP	0.935 ($\sigma = 0.048$)	0.995 ($\sigma = 0.005$)	1.000 ($\sigma = 0.000$)	198
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200
	GFP	uGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\sim	\sim
VCG				$\geq \dagger^{**}$
dVCG				

Table 4.5: Comparing Efficiency (EOS-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.352 ($\sigma = 0.083$)	0.352 ($\sigma = 0.083$)	0.352 ($\sigma = 0.082$)	176
uGSP	0.236 ($\sigma = 0.062$)	0.387 ($\sigma = 0.078$)	0.547 ($\sigma = 0.114$)	198
VCG	0.422 ($\sigma = 0.111$)	0.422 ($\sigma = 0.111$)	0.422 ($\sigma = 0.111$)	200
VCG discrete	0.422 ($\sigma = 0.111$)	0.422 ($\sigma = 0.111$)	0.422 ($\sigma = 0.111$)	200
	GFP	uGSP	VCG	dVCG
GFP		\subseteq^{**}	\sim	\sim
uGSP			\supseteq^{**}	\supseteq^{**}
VCG				\sim
dVCG				

Table 4.6: Comparing Revenue (EOS-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.036 ($\sigma = 0.015$)	0.037 ($\sigma = 0.015$)	0.037 ($\sigma = 0.015$)	176
uGSP	0.001 ($\sigma = 0.002$)	0.066 ($\sigma = 0.041$)	0.279 ($\sigma = 0.109$)	198
	GFP	uGSP		
GFP		\sim		
uGSP				

Table 4.7: Comparing Envy (EOS-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.830 ($\sigma = 0.157$)	0.830 ($\sigma = 0.157$)	0.830 ($\sigma = 0.157$)	146
uGSP	0.736 ($\sigma = 0.218$)	0.826 ($\sigma = 0.208$)	0.883 ($\sigma = 0.195$)	200
wGSP	0.964 ($\sigma = 0.040$)	0.999 ($\sigma = 0.005$)	1.000 ($\sigma = 0.001$)	200
wGFP	0.974 ($\sigma = 0.014$)	0.974 ($\sigma = 0.014$)	0.974 ($\sigma = 0.014$)	181
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	$\leq \dagger^{**}$	\sim
wGFP					$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG						$\geq \dagger^{**}$
dVCG						

Table 4.8: Comparing Efficiency (V-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.393 ($\sigma = 0.132$)	0.393 ($\sigma = 0.132$)	0.393 ($\sigma = 0.131$)	146
uGSP	0.267 ($\sigma = 0.152$)	0.407 ($\sigma = 0.160$)	0.570 ($\sigma = 0.182$)	200
wGSP	0.255 ($\sigma = 0.142$)	0.389 ($\sigma = 0.152$)	0.525 ($\sigma = 0.179$)	200
wGFP	0.393 ($\sigma = 0.173$)	0.393 ($\sigma = 0.173$)	0.393 ($\sigma = 0.173$)	181
VCG	0.443 ($\sigma = 0.187$)	0.443 ($\sigma = 0.187$)	0.443 ($\sigma = 0.187$)	200
VCG discrete	0.442 ($\sigma = 0.187$)	0.442 ($\sigma = 0.187$)	0.442 ($\sigma = 0.187$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\sim	\sim	\sim	\sim	\sim
uGSP			\sim	\supseteq^{**}	\supseteq^{**}	\supseteq^{**}
wGSP				\supseteq^{**}	\supseteq^{**}	\supseteq^{**}
wGFP					\sim	\sim
VCG						\sim
dVCG						

Table 4.9: Comparing Revenue (V-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.701 ($\sigma = 0.211$)	0.701 ($\sigma = 0.211$)	0.701 ($\sigma = 0.211$)	146
uGSP	0.630 ($\sigma = 0.243$)	0.697 ($\sigma = 0.245$)	0.751 ($\sigma = 0.247$)	200
wGSP	0.866 ($\sigma = 0.136$)	0.901 ($\sigma = 0.128$)	0.915 ($\sigma = 0.127$)	200
wGFP	0.888 ($\sigma = 0.118$)	0.889 ($\sigma = 0.118$)	0.889 ($\sigma = 0.118$)	181
VCG	0.901 ($\sigma = 0.129$)	0.901 ($\sigma = 0.129$)	0.901 ($\sigma = 0.129$)	200
VCG discrete	0.902 ($\sigma = 0.128$)	0.902 ($\sigma = 0.128$)	0.902 ($\sigma = 0.128$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	\geq^{**}	\geq^{**}
wGFP					\sim	\sim
VCG						\sim
dVCG						

Table 4.10: Comparing Relevance (V-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.335 ($\sigma = 0.348$)	0.335 ($\sigma = 0.348$)	0.335 ($\sigma = 0.348$)	146
uGSP	0.242 ($\sigma = 0.350$)	0.409 ($\sigma = 0.422$)	0.627 ($\sigma = 0.452$)	200
wGSP	0.002 ($\sigma = 0.004$)	0.076 ($\sigma = 0.067$)	0.247 ($\sigma = 0.183$)	200
wGFP	0.026 ($\sigma = 0.014$)	0.026 ($\sigma = 0.014$)	0.026 ($\sigma = 0.014$)	181

	GFP	uGSP	wGSP	wGFP
GFP		\sim	\sim	$\geq \dagger^{**}$
uGSP			\geq^{**}	$\geq \dagger^{**}$
wGSP				\geq^{**}
wGFP				

Table 4.11: Comparing Envy (V-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.921 ($\sigma = 0.059$)	0.922 ($\sigma = 0.059$)	0.922 ($\sigma = 0.059$)	174
uGSP	0.810 ($\sigma = 0.157$)	0.941 ($\sigma = 0.092$)	0.993 ($\sigma = 0.033$)	200
wGSP	0.938 ($\sigma = 0.056$)	0.997 ($\sigma = 0.005$)	1.000 ($\sigma = 0.000$)	200
wGFP	0.970 ($\sigma = 0.012$)	0.970 ($\sigma = 0.012$)	0.970 ($\sigma = 0.012$)	198
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\sim	\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	\geq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	\sim	\sim
wGFP					$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG						$\geq \dagger^{**}$
dVCG						

Table 4.12: Comparing Efficiency (V-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.333 ($\sigma = 0.076$)	0.333 ($\sigma = 0.076$)	0.333 ($\sigma = 0.076$)	174
uGSP	0.190 ($\sigma = 0.054$)	0.362 ($\sigma = 0.079$)	0.558 ($\sigma = 0.128$)	200
wGSP	0.172 ($\sigma = 0.074$)	0.328 ($\sigma = 0.096$)	0.489 ($\sigma = 0.138$)	200
wGFP	0.302 ($\sigma = 0.107$)	0.302 ($\sigma = 0.107$)	0.302 ($\sigma = 0.107$)	198
VCG	0.351 ($\sigma = 0.126$)	0.351 ($\sigma = 0.126$)	0.351 ($\sigma = 0.126$)	200
VCG discrete	0.351 ($\sigma = 0.126$)	0.351 ($\sigma = 0.126$)	0.351 ($\sigma = 0.126$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\subseteq^{**}	\subseteq^{**}	\sim	\sim	\sim
uGSP			\geq^{**}	\geq^{**}	\geq^{**}	\geq^{**}
wGSP				\geq^{**}	\geq^{**}	\geq^{**}
wGFP					$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG						\sim
dVCG						

Table 4.13: Comparing Revenue (V-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.842 ($\sigma = 0.114$)	0.842 ($\sigma = 0.114$)	0.842 ($\sigma = 0.114$)	174
uGSP	0.750 ($\sigma = 0.147$)	0.848 ($\sigma = 0.132$)	0.921 ($\sigma = 0.104$)	200
wGSP	0.882 ($\sigma = 0.104$)	0.938 ($\sigma = 0.082$)	0.979 ($\sigma = 0.046$)	200
wGFP	0.925 ($\sigma = 0.065$)	0.925 ($\sigma = 0.065$)	0.925 ($\sigma = 0.065$)	198
VCG	0.938 ($\sigma = 0.083$)	0.938 ($\sigma = 0.083$)	0.938 ($\sigma = 0.083$)	200
VCG discrete	0.939 ($\sigma = 0.081$)	0.939 ($\sigma = 0.081$)	0.939 ($\sigma = 0.081$)	200

	GFP	uGSP	wGSP	wGFP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\supseteq^{**}	\supseteq^{**}	\supseteq^{**}
wGFP					$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG						\sim
dVCG						

Table 4.14: Comparing Relevance (V-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.148 ($\sigma = 0.149$)	0.148 ($\sigma = 0.149$)	0.148 ($\sigma = 0.150$)	174
uGSP	0.033 ($\sigma = 0.077$)	0.168 ($\sigma = 0.174$)	0.429 ($\sigma = 0.259$)	200
wGSP	0.001 ($\sigma = 0.002$)	0.057 ($\sigma = 0.045$)	0.257 ($\sigma = 0.123$)	200
wGFP	0.033 ($\sigma = 0.015$)	0.033 ($\sigma = 0.015$)	0.033 ($\sigma = 0.015$)	198

	GFP	uGSP	wGSP	wGFP
GFP		\subseteq^{**}	\sim	$\geq \dagger^{**}$
uGSP			\geq^{**}	\sim
wGSP				\supseteq^{**}
wGFP				

Table 4.15: Comparing Envy (V-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.611 ($\sigma = 0.398$)	0.638 ($\sigma = 0.387$)	0.640 ($\sigma = 0.388$)	200
uGSP	0.584 ($\sigma = 0.392$)	0.657 ($\sigma = 0.400$)	0.682 ($\sigma = 0.412$)	200
wGSP	0.648 ($\sigma = 0.427$)	0.671 ($\sigma = 0.440$)	0.675 ($\sigma = 0.443$)	200
VCG discrete	0.916 ($\sigma = 0.151$)	0.916 ($\sigma = 0.151$)	0.916 ($\sigma = 0.151$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\subseteq^{**}	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.16: Comparing Efficiency (BHN-UNI distribution)

Mechanism	Worst	Median	Best	n	
GFP	0.225 ($\sigma = 0.180$)	0.252 ($\sigma = 0.193$)	0.257 ($\sigma = 0.198$)	200	
uGSP	0.196 ($\sigma = 0.170$)	0.258 ($\sigma = 0.197$)	0.295 ($\sigma = 0.220$)	200	
wGSP	0.231 ($\sigma = 0.201$)	0.270 ($\sigma = 0.218$)	0.296 ($\sigma = 0.234$)	200	
VCG	0.269 ($\sigma = 0.160$)	0.269 ($\sigma = 0.160$)	0.269 ($\sigma = 0.160$)	200	
VCG discrete	0.219 ($\sigma = 0.236$)	0.219 ($\sigma = 0.236$)	0.219 ($\sigma = 0.236$)	200	
	GFP	uGSP	wGSP	VCG	dVCG
	GFP	\subseteq^{**}	\sim	\sim	\sim
	uGSP		\sim	\sim	\sim
	wGSP			\sim	\sim
	VCG				$\geq \dagger^{**}$
	dVCG				

Table 4.17: Comparing Revenue (BHN-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.510 ($\sigma = 0.351$)	0.528 ($\sigma = 0.342$)	0.530 ($\sigma = 0.344$)	200
uGSP	0.464 ($\sigma = 0.338$)	0.542 ($\sigma = 0.356$)	0.581 ($\sigma = 0.377$)	200
wGSP	0.604 ($\sigma = 0.413$)	0.626 ($\sigma = 0.423$)	0.638 ($\sigma = 0.430$)	200
VCG	0.903 ($\sigma = 0.113$)	0.903 ($\sigma = 0.113$)	0.903 ($\sigma = 0.113$)	200
VCG discrete	0.788 ($\sigma = 0.256$)	0.788 ($\sigma = 0.256$)	0.788 ($\sigma = 0.256$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\subseteq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.18: Comparing Relevance (BHN-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.152 ($\sigma = 0.173$)	0.174 ($\sigma = 0.190$)	0.176 ($\sigma = 0.191$)	200
uGSP	0.121 ($\sigma = 0.180$)	0.191 ($\sigma = 0.220$)	0.287 ($\sigma = 0.311$)	200
wGSP	0.097 ($\sigma = 0.185$)	0.110 ($\sigma = 0.185$)	0.146 ($\sigma = 0.198$)	200

	GFP	uGSP	wGSP
GFP		\subseteq^{**}	\sim
uGSP			\sim
wGSP			

Table 4.19: Comparing Envy (BHN-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.775 ($\sigma = 0.221$)	0.835 ($\sigma = 0.127$)	0.854 ($\sigma = 0.120$)	189
uGSP	0.799 ($\sigma = 0.208$)	0.902 ($\sigma = 0.118$)	0.918 ($\sigma = 0.109$)	200
VCG discrete	0.999 ($\sigma = 0.003$)	0.999 ($\sigma = 0.003$)	0.999 ($\sigma = 0.003$)	200

	GFP	uGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG				$\geq \dagger^{**}$
dVCG				

Table 4.20: Comparing Efficiency (BSS distribution)

Mechanism	Worst	Median	Best	n
GFP	0.255 ($\sigma = 0.171$)	0.306 ($\sigma = 0.154$)	0.360 ($\sigma = 0.170$)	189
uGSP	0.210 ($\sigma = 0.140$)	0.327 ($\sigma = 0.152$)	0.436 ($\sigma = 0.193$)	200
VCG	0.408 ($\sigma = 0.188$)	0.408 ($\sigma = 0.188$)	0.408 ($\sigma = 0.188$)	200
VCG discrete	0.412 ($\sigma = 0.186$)	0.412 ($\sigma = 0.186$)	0.412 ($\sigma = 0.186$)	200

	GFP	uGSP	VCG	dVCG
GFP		\sim	\sim	\sim
uGSP			\sim	\sim
VCG				$\leq \dagger^*$
dVCG				

Table 4.21: Comparing Revenue (BSS distribution)

Mechanism	Worst	Median	Best	n
GFP	0.015 ($\sigma = 0.020$)	0.017 ($\sigma = 0.021$)	0.022 ($\sigma = 0.021$)	189
uGSP	0.016 ($\sigma = 0.023$)	0.046 ($\sigma = 0.057$)	0.107 ($\sigma = 0.113$)	200

	GFP	uGSP
GFP		\sim
uGSP		

Table 4.22: Comparing Envy (BSS distribution)

Mechanism	Worst	Median	Best	n
GFP	0.837 ($\sigma = 0.150$)	0.837 ($\sigma = 0.150$)	0.837 ($\sigma = 0.150$)	153
uGSP	0.650 ($\sigma = 0.230$)	0.819 ($\sigma = 0.191$)	0.916 ($\sigma = 0.145$)	200
wGSP	0.855 ($\sigma = 0.118$)	0.947 ($\sigma = 0.081$)	0.981 ($\sigma = 0.048$)	200
cwGSP	0.956 ($\sigma = 0.040$)	0.995 ($\sigma = 0.010$)	1.000 ($\sigma = 0.002$)	164
VCG discrete	1.000 ($\sigma = 0.000$)	1.000 ($\sigma = 0.000$)	1.000 ($\sigma = 0.000$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\sim	\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
cwGSP					$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG						$\geq \dagger^{**}$
dVCG						

Table 4.23: Comparing Efficiency (CAS-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.344 ($\sigma = 0.114$)	0.344 ($\sigma = 0.114$)	0.344 ($\sigma = 0.114$)	153
uGSP	0.189 ($\sigma = 0.117$)	0.370 ($\sigma = 0.137$)	0.575 ($\sigma = 0.157$)	200
wGSP	0.189 ($\sigma = 0.108$)	0.337 ($\sigma = 0.127$)	0.498 ($\sigma = 0.157$)	200
cwGSP	0.156 ($\sigma = 0.098$)	0.270 ($\sigma = 0.119$)	0.396 ($\sigma = 0.158$)	164
VCG	0.312 ($\sigma = 0.159$)	0.312 ($\sigma = 0.159$)	0.312 ($\sigma = 0.159$)	200
VCG discrete	0.312 ($\sigma = 0.158$)	0.312 ($\sigma = 0.158$)	0.312 ($\sigma = 0.158$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\subseteq^{**}	\sim	\sim	\sim	\sim
uGSP			\sim	\sim	\supseteq^{**}	\supseteq^{**}
wGSP				\sim	\supseteq^{**}	\supseteq^{**}
cwGSP					\sim	\sim
VCG						\sim
dVCG						

Table 4.24: Comparing Revenue (CAS-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.669 ($\sigma = 0.211$)	0.669 ($\sigma = 0.211$)	0.669 ($\sigma = 0.211$)	153
uGSP	0.516 ($\sigma = 0.224$)	0.645 ($\sigma = 0.224$)	0.757 ($\sigma = 0.214$)	200
wGSP	0.718 ($\sigma = 0.167$)	0.813 ($\sigma = 0.167$)	0.878 ($\sigma = 0.140$)	200
cwGSP	0.876 ($\sigma = 0.120$)	0.918 ($\sigma = 0.115$)	0.945 ($\sigma = 0.097$)	164
VCG	0.909 ($\sigma = 0.117$)	0.909 ($\sigma = 0.117$)	0.909 ($\sigma = 0.117$)	200
VCG discrete	0.906 ($\sigma = 0.120$)	0.906 ($\sigma = 0.120$)	0.906 ($\sigma = 0.120$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\sim	\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
cwGSP					\sim	\sim
VCG						\sim
dVCG						

Table 4.25: Comparing Relevance (CAS-UNI distribution)

Mechanism	Worst	Median		Best	n
GFP	0.905 ($\sigma = 0.096$)	0.905 ($\sigma = 0.096$)		0.905 ($\sigma = 0.096$)	180
uGSP	0.788 ($\sigma = 0.161$)	0.905 ($\sigma = 0.119$)		0.973 ($\sigma = 0.062$)	200
wGSP	0.861 ($\sigma = 0.125$)	0.951 ($\sigma = 0.074$)		0.984 ($\sigma = 0.049$)	200
cwGSP	0.952 ($\sigma = 0.044$)	0.996 ($\sigma = 0.010$)		1.000 ($\sigma = 0.002$)	170
VCG discrete	1.000 ($\sigma = 0.000$)	1.000 ($\sigma = 0.000$)		1.000 ($\sigma = 0.000$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\sim	\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
cwGSP					$\leq \dagger^{**}$	\sim
VCG						$\geq \dagger^{**}$
dVCG						

Table 4.26: Comparing Efficiency (CAS-LN distribution)

Mechanism	Worst	Median		Best	n
GFP	0.344 ($\sigma = 0.112$)	0.344 ($\sigma = 0.112$)		0.344 ($\sigma = 0.112$)	180
uGSP	0.214 ($\sigma = 0.101$)	0.376 ($\sigma = 0.112$)		0.558 ($\sigma = 0.146$)	200
wGSP	0.172 ($\sigma = 0.096$)	0.317 ($\sigma = 0.123$)		0.474 ($\sigma = 0.167$)	200
cwGSP	0.143 ($\sigma = 0.086$)	0.249 ($\sigma = 0.114$)		0.363 ($\sigma = 0.160$)	170
VCG	0.283 ($\sigma = 0.166$)	0.283 ($\sigma = 0.166$)		0.283 ($\sigma = 0.166$)	200
VCG discrete	0.283 ($\sigma = 0.166$)	0.283 ($\sigma = 0.166$)		0.283 ($\sigma = 0.166$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\subseteq^{**}	\subseteq^{**}	\sim	\sim	\sim
uGSP			\supseteq^{**}	\sim	\supseteq^{**}	\supseteq^{**}
wGSP				\sim	\supseteq^{**}	\supseteq^{**}
cwGSP					\sim	\sim
VCG						\sim
dVCG						

Table 4.27: Comparing Revenue (CAS-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.784 ($\sigma = 0.166$)	0.784 ($\sigma = 0.166$)	0.784 ($\sigma = 0.166$)	180
uGSP	0.658 ($\sigma = 0.182$)	0.764 ($\sigma = 0.181$)	0.873 ($\sigma = 0.156$)	200
wGSP	0.755 ($\sigma = 0.168$)	0.842 ($\sigma = 0.150$)	0.912 ($\sigma = 0.126$)	200
cwGSP	0.904 ($\sigma = 0.090$)	0.953 ($\sigma = 0.068$)	0.980 ($\sigma = 0.046$)	170
VCG	0.942 ($\sigma = 0.079$)	0.942 ($\sigma = 0.079$)	0.942 ($\sigma = 0.079$)	200
VCG discrete	0.942 ($\sigma = 0.079$)	0.942 ($\sigma = 0.079$)	0.942 ($\sigma = 0.079$)	200

	GFP	uGSP	wGSP	cwGSP	VCG	dVCG
GFP		\sim	\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
cwGSP					\sim	\sim
VCG						\sim
dVCG						

Table 4.28: Comparing Relevance (CAS-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.821 ($\sigma = 0.160$)	0.821 ($\sigma = 0.160$)	0.821 ($\sigma = 0.160$)	89
uGSP	0.711 ($\sigma = 0.263$)	0.812 ($\sigma = 0.242$)	0.867 ($\sigma = 0.225$)	200
wGSP	0.955 ($\sigma = 0.083$)	0.987 ($\sigma = 0.041$)	0.994 ($\sigma = 0.029$)	200
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.29: Comparing Efficiency (HYB-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.503 ($\sigma = 0.141$)	0.503 ($\sigma = 0.141$)	0.504 ($\sigma = 0.142$)	89
uGSP	0.342 ($\sigma = 0.179$)	0.508 ($\sigma = 0.188$)	0.670 ($\sigma = 0.208$)	200
wGSP	0.321 ($\sigma = 0.158$)	0.469 ($\sigma = 0.152$)	0.615 ($\sigma = 0.175$)	200
VCG	0.534 ($\sigma = 0.185$)	0.534 ($\sigma = 0.185$)	0.534 ($\sigma = 0.185$)	200
VCG discrete	0.534 ($\sigma = 0.185$)	0.534 ($\sigma = 0.185$)	0.534 ($\sigma = 0.185$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	\sim	\sim
uGSP			\sim	\supseteq^{**}	\supseteq^{**}
wGSP				\supseteq^{**}	\supseteq^{**}
VCG					\sim
dVCG					

Table 4.30: Comparing Revenue (HYB-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.638 ($\sigma = 0.214$)	0.638 ($\sigma = 0.214$)	0.638 ($\sigma = 0.214$)	89
uGSP	0.570 ($\sigma = 0.257$)	0.652 ($\sigma = 0.261$)	0.708 ($\sigma = 0.255$)	200
wGSP	0.829 ($\sigma = 0.164$)	0.864 ($\sigma = 0.152$)	0.888 ($\sigma = 0.139$)	200
VCG	0.886 ($\sigma = 0.136$)	0.886 ($\sigma = 0.136$)	0.886 ($\sigma = 0.136$)	200
VCG discrete	0.888 ($\sigma = 0.132$)	0.888 ($\sigma = 0.132$)	0.888 ($\sigma = 0.132$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	$\leq \dagger^*$	$\leq \dagger^*$
uGSP			$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\sim	\sim
VCG					\sim
dVCG					

Table 4.31: Comparing Relevance (HYB-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.775 ($\sigma = 0.210$)	0.775 ($\sigma = 0.210$)	0.776 ($\sigma = 0.210$)	171
uGSP	0.698 ($\sigma = 0.305$)	0.759 ($\sigma = 0.298$)	0.812 ($\sigma = 0.275$)	200
wGSP	0.982 ($\sigma = 0.046$)	0.997 ($\sigma = 0.014$)	1.000 ($\sigma = 0.001$)	200
VCG discrete	1.000 ($\sigma = 0.004$)	1.000 ($\sigma = 0.004$)	1.000 ($\sigma = 0.004$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	\sim
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.32: Comparing Efficiency (HYB-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.365 ($\sigma = 0.133$)	0.366 ($\sigma = 0.133$)	0.366 ($\sigma = 0.133$)	171
uGSP	0.192 ($\sigma = 0.116$)	0.320 ($\sigma = 0.160$)	0.472 ($\sigma = 0.229$)	200
wGSP	0.151 ($\sigma = 0.125$)	0.274 ($\sigma = 0.176$)	0.390 ($\sigma = 0.238$)	200
VCG	0.325 ($\sigma = 0.222$)	0.325 ($\sigma = 0.222$)	0.325 ($\sigma = 0.222$)	200
VCG discrete	0.326 ($\sigma = 0.221$)	0.326 ($\sigma = 0.221$)	0.326 ($\sigma = 0.221$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	\sim	\sim
uGSP			\sim	\geq^{**}	\geq^{**}
wGSP				\geq^{**}	\geq^{**}
VCG					\sim
dVCG					

Table 4.33: Comparing Revenue (HYB-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.613 ($\sigma = 0.268$)	0.613 ($\sigma = 0.268$)	0.613 ($\sigma = 0.268$)	171
uGSP	0.543 ($\sigma = 0.315$)	0.596 ($\sigma = 0.325$)	0.651 ($\sigma = 0.321$)	200
wGSP	0.864 ($\sigma = 0.206$)	0.895 ($\sigma = 0.180$)	0.921 ($\sigma = 0.152$)	200
VCG	0.904 ($\sigma = 0.168$)	0.904 ($\sigma = 0.168$)	0.904 ($\sigma = 0.168$)	200
VCG discrete	0.900 ($\sigma = 0.171$)	0.900 ($\sigma = 0.171$)	0.900 ($\sigma = 0.171$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			$\leq \dagger^{**}$	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				\geq^{**}	\geq^{**}
VCG					\sim
dVCG					

Table 4.34: Comparing Relevance (HYB-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.829 ($\sigma = 0.131$)	0.829 ($\sigma = 0.131$)	0.829 ($\sigma = 0.131$)	130
uGSP	0.684 ($\sigma = 0.214$)	0.828 ($\sigma = 0.189$)	0.905 ($\sigma = 0.135$)	200
wGSP	0.858 ($\sigma = 0.112$)	0.950 ($\sigma = 0.073$)	0.981 ($\sigma = 0.043$)	200
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.35: Comparing Efficiency (GIM-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.424 ($\sigma = 0.120$)	0.424 ($\sigma = 0.120$)	0.424 ($\sigma = 0.120$)	130
uGSP	0.274 ($\sigma = 0.129$)	0.462 ($\sigma = 0.147$)	0.650 ($\sigma = 0.153$)	200
wGSP	0.248 ($\sigma = 0.111$)	0.405 ($\sigma = 0.122$)	0.569 ($\sigma = 0.156$)	200
VCG	0.384 ($\sigma = 0.164$)	0.384 ($\sigma = 0.164$)	0.384 ($\sigma = 0.164$)	200
VCG discrete	0.385 ($\sigma = 0.165$)	0.385 ($\sigma = 0.165$)	0.385 ($\sigma = 0.165$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	\sim	\sim
uGSP			\sim	\supseteq^{**}	\supseteq^{**}
wGSP				\supseteq^{**}	\supseteq^{**}
VCG					\sim
dVCG					

Table 4.36: Comparing Revenue (GIM-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.642 ($\sigma = 0.190$)	0.642 ($\sigma = 0.190$)	0.642 ($\sigma = 0.190$)	130
uGSP	0.536 ($\sigma = 0.208$)	0.654 ($\sigma = 0.222$)	0.736 ($\sigma = 0.203$)	200
wGSP	0.728 ($\sigma = 0.153$)	0.818 ($\sigma = 0.160$)	0.869 ($\sigma = 0.145$)	200
VCG	0.900 ($\sigma = 0.124$)	0.900 ($\sigma = 0.124$)	0.900 ($\sigma = 0.124$)	200
VCG discrete	0.900 ($\sigma = 0.126$)	0.900 ($\sigma = 0.126$)	0.900 ($\sigma = 0.126$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					\sim
dVCG					

Table 4.37: Comparing Relevance (GIM-UNI distribution)

Mechanism	Worst	Median	Best	n
GFP	0.891 ($\sigma = 0.091$)	0.892 ($\sigma = 0.091$)	0.892 ($\sigma = 0.091$)	169
uGSP	0.802 ($\sigma = 0.165$)	0.906 ($\sigma = 0.128$)	0.954 ($\sigma = 0.089$)	200
wGSP	0.881 ($\sigma = 0.104$)	0.956 ($\sigma = 0.064$)	0.981 ($\sigma = 0.039$)	200
VCG discrete	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	1.000 ($\sigma = 0.001$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					$\geq \dagger^{**}$
dVCG					

Table 4.38: Comparing Efficiency (GIM-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.398 ($\sigma = 0.119$)	0.398 ($\sigma = 0.119$)	0.398 ($\sigma = 0.118$)	169
uGSP	0.277 ($\sigma = 0.101$)	0.428 ($\sigma = 0.120$)	0.582 ($\sigma = 0.160$)	200
wGSP	0.225 ($\sigma = 0.118$)	0.354 ($\sigma = 0.148$)	0.493 ($\sigma = 0.191$)	200
VCG	0.339 ($\sigma = 0.185$)	0.339 ($\sigma = 0.185$)	0.339 ($\sigma = 0.185$)	200
VCG discrete	0.339 ($\sigma = 0.186$)	0.339 ($\sigma = 0.186$)	0.339 ($\sigma = 0.186$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\subseteq^{**}	\sim	\sim	\sim
uGSP			\geq^{**}	\supseteq^{**}	\supseteq^{**}
wGSP				\supseteq^{**}	\supseteq^{**}
VCG					\sim
dVCG					

Table 4.39: Comparing Revenue (GIM-LN distribution)

Mechanism	Worst	Median	Best	n
GFP	0.755 ($\sigma = 0.156$)	0.755 ($\sigma = 0.155$)	0.755 ($\sigma = 0.155$)	169
uGSP	0.661 ($\sigma = 0.182$)	0.762 ($\sigma = 0.179$)	0.830 ($\sigma = 0.160$)	200
wGSP	0.765 ($\sigma = 0.137$)	0.844 ($\sigma = 0.132$)	0.897 ($\sigma = 0.113$)	200
VCG	0.933 ($\sigma = 0.093$)	0.933 ($\sigma = 0.093$)	0.933 ($\sigma = 0.093$)	200
VCG discrete	0.934 ($\sigma = 0.092$)	0.934 ($\sigma = 0.092$)	0.934 ($\sigma = 0.092$)	200

	GFP	uGSP	wGSP	VCG	dVCG
GFP		\sim	\sim	$\leq \dagger^{**}$	$\leq \dagger^{**}$
uGSP			\leq^{**}	$\leq \dagger^{**}$	$\leq \dagger^{**}$
wGSP				$\leq \dagger^{**}$	$\leq \dagger^{**}$
VCG					\sim
dVCG					

Table 4.40: Comparing Relevance (GIM-LN distribution)

Chapter 5

Application: Maximizing Internet Advertising Revenue

5.1 Introduction

The previous chapter introduced internet advertising auctions, and showed that computational mechanism analysis could be used to analyze and compare different auction designs. This chapter also focuses on internet advertising as an application domain. However, in this chapter, the focus is on computational mechanism analysis as a tool for mechanism design, specifically with the objective of maximizing search-engine revenue.

As described in the previous chapter, the major search engines generate most of their revenue from advertising auctions and all have converged on the weighted generalized second-price auction (wGSP). Notably, this auction design is not incentive compatible—it is typically not in an advertiser’s best interest to truthfully report her per-click willingness to pay. In contrast, incentive compatible designs have been proposed by the research community [e.g., 1], but not adopted by any search engine.

Revenue optimization is typically framed as a question of finding the revenue-maximizing incentive-compatible auction design. We instead ask “how can we optimize revenue within the basic GSP design?” Specifically, we consider a range of different GSP variants that have been used in practice. Each has adjustable parameters; we consider how these parameters should be set in order to optimize revenue.

Our most striking finding is that a new reserve price scheme used by major search engines (“quality-weighted reserves”: low-quality advertisers must pay more per click) is not a good choice from a revenue perspective; the old scheme (“unweighted reserves”: all advertisers have the same per-click reserve price) is substantially better. Indeed, this finding is not just striking, but also robust: we offer evidence for it across three different

analysis methods and two different sets of assumptions about bidder valuations. We observed that richer GSP variants sometimes outperformed GSP with unweighted reserves, but these variants tended to incorporate (approximately) unweighted reserves. Three of this chapter’s other findings also deserve mention here. First, we identify a new GSP variant that is provably revenue optimal in some restricted settings, and that achieves very good revenue in settings where it is not optimal. Second, we perform the first systematic investigation of the interaction between reserve prices and another revenue-optimization technique called “squashing”. Interestingly, we find that squashing can substantially improve the performance of quality-weighted reserve prices, but this effect arises because squashing undoes the quality weighting. Third, we perform the first systematic investigation of how equilibrium selection interacts with revenue optimization techniques like reserve prices. Again, unweighted reserve prices prove to be superior, especially from the perspective of optimizing the revenue of worst-case equilibria.

This chapter proceeds as follows. Section 5.2 describes the GSP variants and how they are used in practice, the model of advertiser preferences, and finally a review of related work. In Section 5.3 we perform a theoretical analysis of the problem of selling only a single position. The simplicity of this setting allows us to identify properties of the optimal auction, and to contrast them with the GSP variants. Section 5.4 covers our first computational analysis of auctions involving multiple ad positions. Here, we used efficient game representations to enumerate the entire set of pure-strategy equilibria. Section 5.5 describes a further computational analysis, where we performed very large-scale experiments, focusing on a commonly studied restricted class of equilibria. Section 5.6 discusses some implications of our results and directions for future work.

5.2 Background

Recall the weighted generalized second-price auction (wGSP) from the previous chapter. (Hereafter, we refer to it as the “vanilla GSP” to distinguish it from the other wGSP-based auction designs that we discuss later. Each advertiser i is assigned a quality score $q_i \in (0, 1]$ by the search engine. We assume that this quality score is equal to the probability that the advertiser’s ad would be clicked on if it were shown in the top position. Each advertiser submits a bid b_i that specifies the maximum per-click price that she would be willing to pay. Ads are shown ordered by $b_i q_i$ from highest to lowest. When a user clicks on an ad, the advertiser pays the minimum amount she could have bid while maintaining her position in the ordering.

Three simple variants of vanilla GSP have been proposed—and used in practice—to improve revenue.

1. **Squashing** changes vanilla GSP by changing how bidders are ranked, according to a real-valued parameter s : bidders are ranked by $b_i q_i^s$. When $s = 1$, squashing GSP is identical to regular GSP. When $s = 0$, squashing GSP throws away all quality information and ranks advertisers by their bids, promoting lower-quality advertisers and forcing higher-quality advertisers to pay more to retain the

same position. Intermediate values of $s \in (0, 1)$ smoothly interpolate between these two extremes [60].

2. An *unweighted reserve price* (**UWR**) specifies a minimum per-click price r that every advertiser must bid (and pay) in order for her ad to be shown.¹ We call this per-click reserve price “unweighted” because it is constant across all advertisers, regardless of their qualities.
3. A *quality-weighted reserve price* (**QWR**) also specifies a minimum price that advertisers must pay, but now this price is increased for higher-quality bidders. Specifically, each advertiser i must pay at least r/q_i per click.

Squashing and both kinds of reserve prices have been used in practice (and in combination). Google initially used a UWR of \$0.05 across all keywords [103], but switched to QWR in 2005 [45]. Yahoo! also had a UWR (\$0.10), but in a well-documented field experiment switched to QWR while both tailoring reserve prices to specific keywords and dramatically increasing them overall [81]. Yahoo! researchers have publicly confirmed that their auctions use squashing [71]. However, since search engines withhold many auction details (e.g., the methods used to calculate quality scores and minimum bids), it is impossible to be certain of current practice.

Finally, we consider three richer GSP variants. The first two, **UWR+Sq** and **QWR+Sq**, combine squashing with the two reserve price variants. The third—which to our knowledge had not previously been discussed in the literature, and which we dub **anchoring**²—imposes unweighted reserve prices, but ranks advertisers based only on the portion of bids that exceeds the reserve price, multiplied by the quality score: $(b_i - r)q_i$. Anchoring is interesting because (as we show in Section 3) it is the optimal auction for some very simple settings; we also found that it performed well in settings where it was not provably optimal.

5.2.1 Model

For this chapter, we focus on Varian’s model (V) [103], which is one of the most widely studied models, particularly among researchers studying revenue optimization.³ In this model, a setting is specified by a 4-tuple $\langle N, v, q, \alpha \rangle$: N is the set of agents (numbered $1, \dots, n$); v_i specifies agent i ’s value for a single click; q_i specifies agent i ’s quality score, which is equal to the probability that i ’s ad would receive a click if shown in the top position ($q_i > 0$); and α_j is the probability that an agent with quality of 1.0 will receive a click in position j . Observe that $\alpha_1 = 1.0$; we further assume that α is decreasing, meaning that higher positions get

¹Note: we assume that advertisers who do not bid above their minimum bid do not affect the outcome in any way. Particularly, this means that an advertiser who bids below her minimum bid does not affect the prices that other agents must pay.

²Roberts *et al.*, studying much the same problem as this chapter, independently invented the same auction design [88].

³Among the models considered in the previous chapter, V is the most important for this analysis. EOS [30] is unsuitable, because it lacks quality scores. This lack means that we cannot compare many GSP variants that differ in how they handle quality scores. The other models include multi-parameter agents (BHN and BSS) or externalities (cascade, hybrid and GIM). Although these features do not prevent computational analysis using our methods (as in Section 5.4), they do rule out more conventional methods (such as those in Section 5.5). Perhaps because these models are harder to analyze, there is also essentially no literature on revenue optimization for them. Revenue optimization in richer models remains an important open problem where CMA can make a substantial contribution.

more clicks. Thus, if agent i 's ad is shown in position j with a price of p per click, then her expected payoff is $\alpha_j q_i (v_i - p)$.

As in Chapter 4 and in most of the literature on GSP [e.g., 30, 103], we assume that the setting is common knowledge to all the advertisers. This assumption is motivated by the fact that in practice, advertisers can learn a great deal about each other through repeated interactions. We assume that the search engine is able to directly observe and condition on the number of bidders and their quality scores. As is common in the literature on revenue optimization (going back to [75] and continuing in work related to GSP [81, 97]), we further assume that details of the setting (v, q, α) are drawn from a distribution which is known to the auctioneer. Thus while the search engine is unable to choose a reserve price conditional on the advertisers' valuations, it can base this decision on the distribution from which these values are drawn. We further assume that the auctioneer can observe and condition on the number of agents, and on each agent's quality score. To understand this last assumption, note that the search engine directly observes every click on an ad. Thus, even if q_i begins as private information, it is impossible for an advertiser to misreport this value. This assumption is a key distinction between our work and previous work on revenue optimization (most notably, [97]).

5.2.2 Related Work

Revenue has been a major consideration since the earliest equilibrium analysis of vanilla GSP. Foundational research in the area [30, 103] analyzed a specific equilibrium refinement, under which they found that GSP was at least as good at generating revenue as VCG. Subsequent research has shown that VCG (and therefore GSP) achieves revenue close to that of the optimal auction [25]—though in practice, this could mean that search engines leave billions of dollars on the table. Other work has looked into general Nash equilibria (without the refinements mentioned earlier) and found that GSP has many equilibria, ranging from some much worse than VCG [98] to others that are significantly better [64, 98].

Lahaie and Pennock [60] first introduced the concept of squashing. While squashing behaves similarly to the virtual values of Myerson, in that it tends to promote weak bidders, they proved that no squashing scheme (or indeed any other manipulation of quality scores) can yield an optimal auction. (Our Figure 5.1 gives some visual intuition as to why this is true.) They also performed substantial simulation experiments, demonstrating the effectiveness of squashing as a means of sacrificing efficiency for revenue. Recently—and departing from the model we consider in our own work—it has been shown that squashing can improve efficiency when quality scores are noisy [59].

Reserve prices have been used in GSP auctions for years, but our theoretical understanding of their effects is still relatively limited. Since weighted GSP has the same outcome as VCG in many analyses (e.g., [1] and the Bayesian analyses of [103] and [41]), one could infer that weighted GSP with a weighted reserve would have the same outcome as VCG with an unweighted reserve, which often corresponds to the optimal auction of [75]. [81] showed that GSP with weighted reserves is not quite equivalent to the optimal auction in the case

of asymmetric bidders. Recently, [97] showed that GSP with weighted reserves is optimal when the quality score is part of the advertisers' private information. However, the question of how to optimize revenue in the arguably more realistic case where quality scores are known to the auctioneer remains open. Despite the fact that squashing and reserve prices have been used together in practice, we are aware of no studies of how they interact. Further, little is known about how equilibrium selection affects the revenue of GSP with reserves or squashing.

5.3 First Analysis: Single-Slot Auctions with Known Quality Scores

For our first analysis of the problem of revenue optimization in GSP, we consider an extremely restricted case—selling a single slot to advertisers with independent, identically distributed per-click valuations, but known quality scores. We restrict ourselves to a single slot because it allows us to rely directly upon Myerson's characterization to identify the optimal auction. Observe that our use of Varian's model in this setting is less restrictive than it might appear: richer models such as cascade [e.g., 36, 56] and position preferences [e.g., 9, 11] all collapse to Varian's model in the single-slot case.

First, we consider the problem of which kind of reserve prices are optimal.

Proposition 12. *The optimal auction uses the same per-click reserve price for all bidders in any one-position setting for which all agents' per-click valuations (v) are independently drawn from a common, regular distribution g .*

Proof. First, observe that although per-click valuations are identically distributed in this setting, agents' per-impression valuations (denoted \mathcal{V}) are not. If an agent i 's per-click valuation is v_i , then her per-impression valuation (given q_i) is $q_i v_i = \mathcal{V}_i$. Because the auctioneer is effectively selling impressions, it is the latter value that matters. Let f_i and F_i denote the probability density function and cumulative distribution function of \mathcal{V}_i .

As was shown by Myerson [75], when f is regular the optimal auction allocates by virtual values ψ :

$$\psi_i(\mathcal{V}_i) = \mathcal{V}_i - \frac{1 - F_i(\mathcal{V}_i)}{f_i(\mathcal{V}_i)}. \quad (5.1)$$

For any per-click valuation distribution g , we can identify the per-impression valuation distribution for an agent with quality q_i : $f(q_i v_i) = g(v_i)/q_i$ and $F(q_i v_i) = G(v_i)$. Substituting these into (1) gives

$$\psi_i(q_i v_i) = q_i v_i - \frac{1 - G_i(v_i)}{g_i(v_i)/q_i} = q_i \left(v_i - \frac{1 - G_i(v_i)}{g_i(v_i)} \right). \quad (5.2)$$

The value v_i that makes this expression equal to zero is independent of q_i , and so the optimal per-click reserve is independent of q_i . \square

Auction	Revenue	Parameter(s)
VCG/GSP	0.208	—
Squashing	0.255	$s = 0.19$
QWR	0.279	$r = 0.375$
UWR	0.316	$r = 0.549$
QWR+Sq	0.321	$r = 0.472, s = 0.24$
UWR+Sq	0.322	$r = 0.505, s = 0.32$
Anchoring	0.323	$r = 0.5$

Table 5.1: Comparing GSP variants for two bidders with $q_1 = 1$, $q_2 = 1/2$, and $v_1, v_2 \sim U(0, 1)$.

Next, we consider a simple value distribution: the uniform one. Because this case has such a simple functional form, it is easy to identify the optimal auction for such bidders. In fact, the optimal auction for the uniform distribution is precisely the anchoring rule described earlier.

Proposition 13. *The anchoring GSP auction is optimal in any one-position setting for which (1) all the agents’ per-click valuations (v) are independently drawn from a uniform distribution on $[0, \bar{v}]$ (hereafter $U(0, \bar{v})$), and (2) each agent i ’s quality score q_i is known to the auctioneer.*

Proof. For valuations from $U(0, x)$, $f(v) = 1/x$ and $F(v) = v/x$. Note that for every agent i , x corresponds to i ’s maximum possible per-impression valuation $q_i \bar{v}$. Substituting these into Equation (1) gives

$$\psi_i(q_i v_i) = q_i v_i - \frac{1 - q_i v_i / (q_i \bar{v})}{1 / (q_i \bar{v})} = q_i (2v_i - \bar{v}). \quad (5.3)$$

Thus, the optimal per-click reserve price r_i^* occurs at $\psi_i(q_i r_i) = 0$, $r_i = \bar{v}/2$ in this case. In the optimal auction, advertisers are ranked by $\psi(q_i v_i) = q_i (2v_i - \bar{v}) \propto q_i (v_i - r_i^*)$, and so the anchoring auction is optimal. \square

However, not all value distributions give rise to optimal auctions with simple (e.g., linear) forms. Consider the log-normal distribution, which some researchers have argued is a good model of real-world bidder valuations [60, 81]. The optimal auction for log-normal distributions uses unweighted reserve prices, and behaves similarly to anchoring when bids are close to the reserve. However, far from the reserves, the optimal auction’s allocation more closely resembles vanilla GSP. (See Figure 5.1 for a visualization of the optimal auction for uniform valuations and Figure 5.4 for a visualization of the optimal auction for log-normal valuations.)

The uniform distribution also makes it easy to calculate the optimal auction’s expected revenue. We considered the case of two bidders, one with high quality ($q_1 = 1$) and one with lower quality ($q_2 = 1/2$), and calculated the optimal parameter settings for each of the GSP variants defined above (see Table 5.1).⁴ While anchoring (of course) generated the most revenue, other mechanisms could also be configured to achieve very nearly

⁴Because of the simplicity of the model, we could calculate the expected revenues numerically (to 10^{-5} accuracy). To find the optimal parameter settings, we used grid search with increments of 0.001 for r and increments of 0.01 for s .

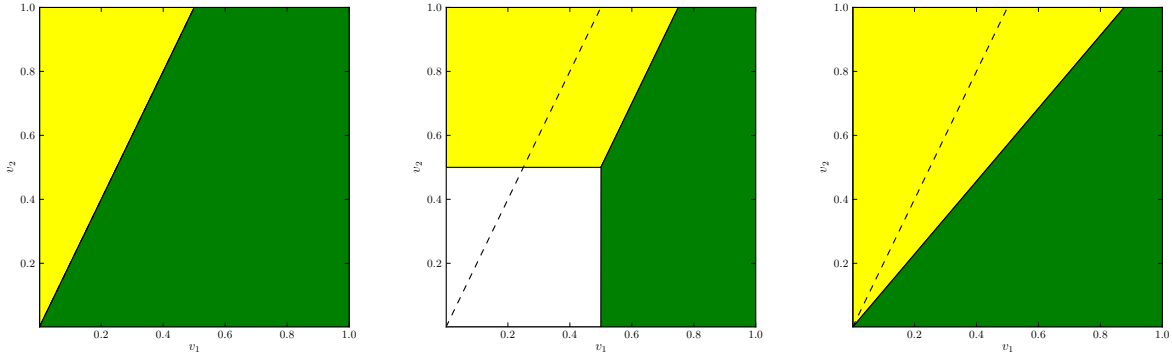


Figure 5.1: Allocation functions visualized (regions in which agent 1 wins are dark green; regions where agent 2 wins are yellow.) Left: the efficient (VCG) auction; middle: the revenue-optimal auction; right: the optimal squashing auction. Note that squashing changes the slope of the dividing line, while the optimal auction transposes it without changing the slope. (Specifically, the optimal auction “anchors” the dividing line to the point where every agent bids his reserve price.) This provides intuition for the result by [60] that no combination of squashing and reserve prices can implement the optimal auction.

optimal revenues: reserves and squashing used together were $\sim 99\%$ optimal, and UWR was $\sim 98\%$ optimal. Squashing and QWR were far behind (at $\sim 79\%$ and $\sim 83\%$ respectively).

We can gain insight into the GSP variants’ similarities and differences by visualizing their allocation functions, for the same setting with two bidders with uniformly distributed valuations and different quality scores (see Figures 5.1–5.3). In each case, the x and y axes correspond to the per-click valuations of agents 1 and 2 respectively. The green region, yellow region and white region respectively indicate joint values for which agent 1 wins, agent 2 wins, and neither agent wins. The dashed line indicates the dividing line of the efficient allocation, with agent 1 winning below the line and agent 2 winning above it.

In summary, this section considered single-slot auctions with i.i.d. per-click values, and obtained the following main findings:

1. the optimal auction uses unweighted reserve prices;
2. when values are uniform, anchoring GSP is optimal;
3. in a very restricted uniform setting, the richer mechanisms (anchoring, QWR+sq, and UWR+sq) achieve approximately equal revenue when optimized, and are slightly better than UWR which is better than QWR and squashing.

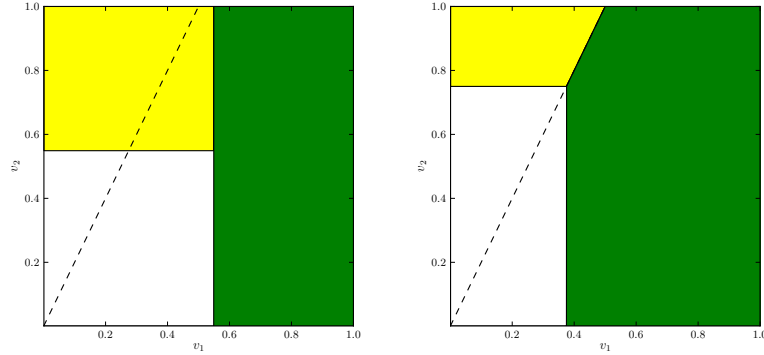


Figure 5.2: Left: the optimal UWR auction; right: the optimal QWR auction. Note that the optimal unweighted reserve is higher than the reserve used by the optimal auction (0.588 rather than 0.5), and that agent 2 only wins when agent 1 does not meet his reserve, because 1's quality is much higher. Also, note the compromise involved in quality-weighted reserve prices: because the reserve prices must correspond to a point on the efficient dividing line, obtaining a reasonable reserve for agent 1 (relative to the optimal auction) results in a much-too-high reserve for agent 2. Because of this compromise, QWR generates $\sim 13\%$ less revenue than UWR. Note that whenever agent 1 exceeds his reserve price in UWR, he wins regardless of agent 2's bid. For multi-slot UWR auctions, this can have an unexpected side effect: it can be impossible for a high-quality advertiser to win the second position.

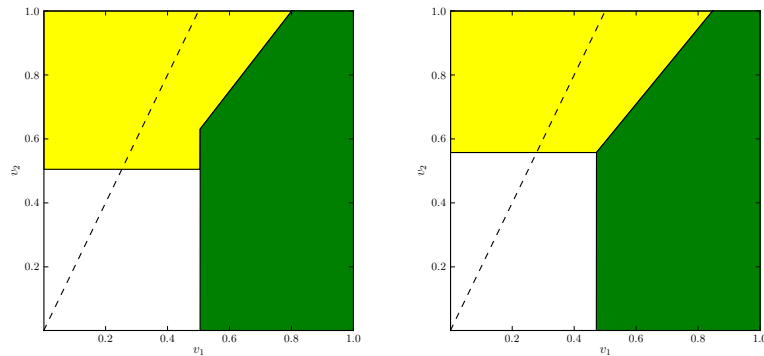


Figure 5.3: Left: the UWR+Sq auction; right: the QWR+Sq auction. Both have reserves that are much closer to the reserves of the optimal auction (and both are within $\sim 1\%$ of revenue-optimal), but both use substantial squashing (0.2 and 0.3 respectively, where squashing of 0 indicates completely disregarding quality scores).

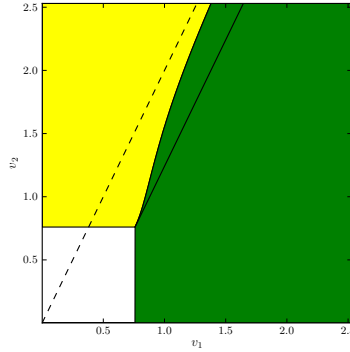


Figure 5.4: The optimal auction for log-normal valuations ($\mu = 0$ and $\sigma = 0.25$) and plotting valuations up to the 99.9th quantile. This auction resembles anchoring (shown with the solid black line) when values are below or near the reserve price. As values get further from the reserve, the allocation tends towards the efficient auction.

5.4 Second Analysis: Multiple Slots, All Pure Nash Equilibria

As mentioned earlier, it is widely known that GSP has multiple Nash equilibria that can yield substantially different revenue and social welfare. For our second analysis, we investigated how equilibrium selection affects GSP and the six GSP variants by directly calculating their pure strategy Nash equilibria. We use the wGSP encoding algorithm from Chapter 4 to represent these auction games as AGGs, and AGG-SEM (from Chapter 3) to enumerate all⁵ the PSNEs of each game.

For this set of experiments we used a uniform distribution⁶ over settings: drawing each agent’s valuation from $U(0, 25)$, each agent’s quality score from $U(0, 1)$, and α_{k+1} from $U(0, \alpha_k)$. We generated 100 5-bidder, 5-slot settings. (Observe that our use of reserves implies that not all ads will be shown for every realization of bidder values.) Every auction has a minimum bid of 1 bid increment per click, with ties broken randomly and prices rounded up to the next whole increment. We used grid search to explore the space of possible parameter settings. We varied reserve prices between 0.1 and 1 in steps of 0.1 (this range only affects low quality advertisers and only in QWR), and between 1 and 25 in steps of 2. We varied squashing power between 0 and 1 in steps of 0.2.

As with our previous two-bidder, one-slot analysis, we used grid search to explore the space of possible reserve prices and squashing factors. Specifically, we computed all pure-strategy Nash equilibria of every

⁵As is common in worst-case analysis of equilibria of auctions, [e.g., 12, 92] we assume that bidders are conservative, i.e., no bidder ever follows the weakly dominated strategy of bidding more than his valuation. Without this assumption, many implausible equilibria are possible, e.g., even single-good Vickrey auctions have equilibria that are unboundedly far from efficient, and unboundedly far from the revenue of truthful bidding.

⁶We could not get meaningful results for log-normal distributions. In a log-normal distribution, a large fraction of the expected revenue is contributed by a small fraction of instances involving bidders with exceptionally high quality scores and valuations. Thus, accurate expected-revenue estimates require many more samples than we could practically generate.

Auction	Revenue	Parameter(s)	Auction	Revenue	Parameter(s)
Vanilla GSP	3.814	—	Vanilla GSP	9.911	—
Squashing	4.247	$s = 0.4$	QWR	10.820	$r = 5.0$
QWR	9.369	$r = 9.0$	Squashing	11.534	$s = 0.2$
Anchoring	10.212	$r = 13.0$	UWR	11.686	$r = 11.0$
QWR+Sq	10.217	$r = 15.0, s = 0.2$	Anchoring	12.464	$r = 11.0$
UWR	11.024	$r = 15.0$	QWR+Sq	12.627	$r = 7.0, s = 0.2$
UWR+Sq	11.032	$r = 15.0, s = 0.6$	UWR+Sq	12.745	$r = 9.0, s = 0.2$

Table 5.2: Comparing the various auction variants given their optimal parameter settings (Left: Worst-case equilibrium; Right: Best-case equilibrium) Bold indicates variants that are significantly better than all other variants, but not significantly different from each other (based on $p \leq 0.05$ with Bonferroni correction).

one of our 100 perfect-information auction settings, and every discretized setting of each GSP variant’s parameter(s). For each variant we identified the parameter settings that maximized: (i) the revenue of the worst-case equilibrium, averaged across settings; and (ii) the revenue of the best-case equilibrium, again averaged across settings.

Broadly, we found that every reserve price scheme dramatically improved worst-case revenue, though UWR was particularly effective. Squashing did not help appreciably with the revenue of worst-case equilibria (see Figure 5.5). Comparing the mechanisms (Table 5.2), we found that UWR and UWR+Sq were among the best and were dramatically better than any other mechanism in terms of worst-case equilibria. Also, we noticed that optimizing for worst-case equilibria consistently yielded higher reserve prices than optimizing for best-case equilibria.

In summary, for this experiment we enumerated the equilibria of perfect-information, 5-slot, 5-bidder ad auction settings with independent, uniform valuations, and found that:

1. There is a huge gap between best- and worst-case equilibria (over $2.5\times$ for vanilla GSP). Squashing does not help to close this gap, but reserve prices do.
2. The optimal reserve price is much higher (for any GSP variant) when optimizing worst-case revenue than when optimizing best-case revenue.
3. When considering best-case equilibria, the revenue ranking remains roughly the same as in our simple 2-bidder, 1-slot analysis (Anchoring \simeq QWR+sq \simeq UWR+sq $>$ UWR $>$ Squashing $>$ QWR). When considering the worst-case equilibria, the revenue ranking changes slightly (UWR \simeq UWR+sq $>$ Anchoring \simeq QWR+sq $>$ QWR $>$ Squashing).

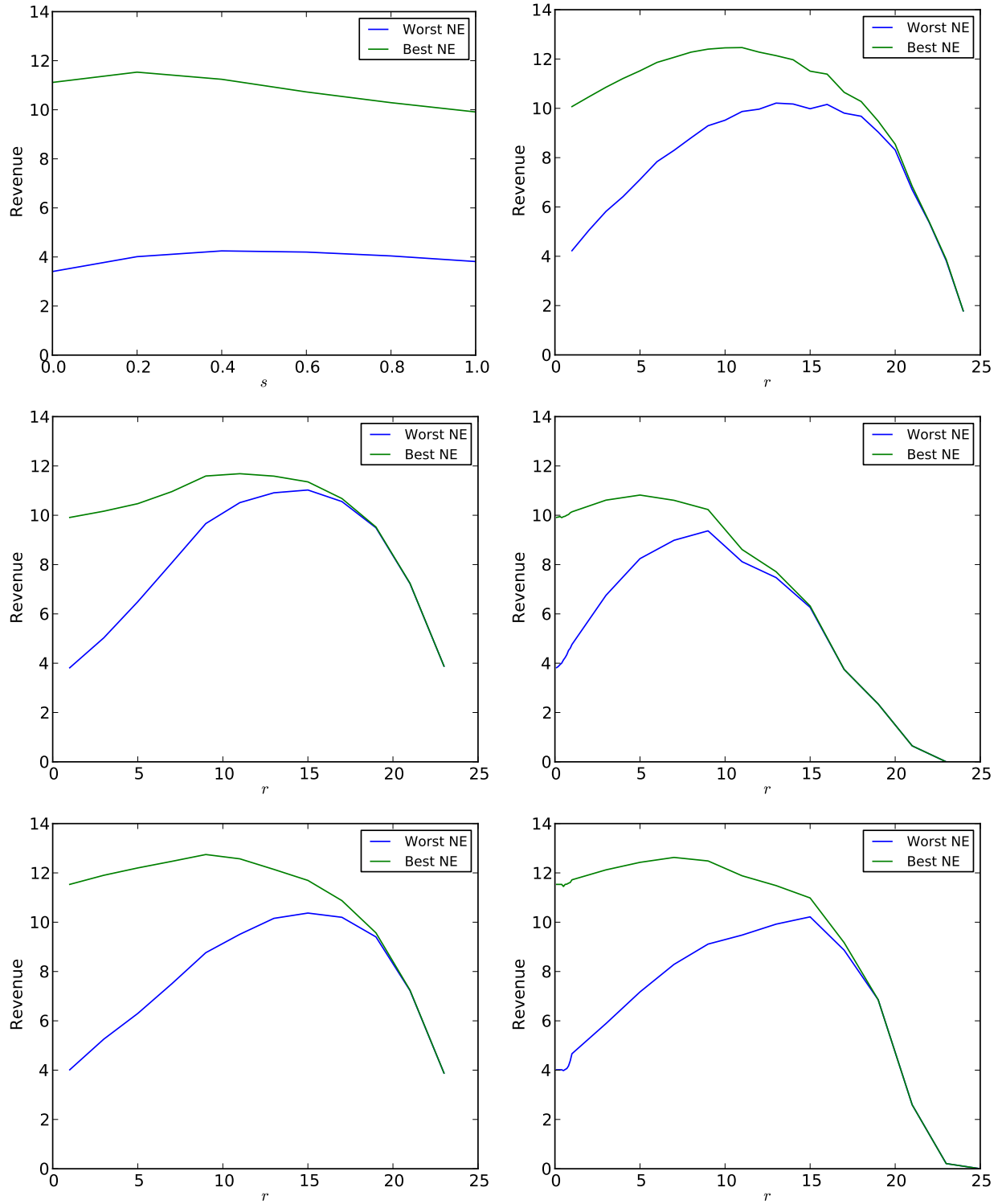


Figure 5.5: Depending on parameter settings, revenue can vary dramatically between worst- and best-case equilibria. Squashing has almost no effect on worst-case equilibrium, while any reserve price scheme can substantially improve it. (Top left: squashing; top right: anchoring; middle left: UWR; middle right: QWR; bottom left: UWR+sq ($s = 0.2$); bottom right: QWR+sq ($s = 0.2$))

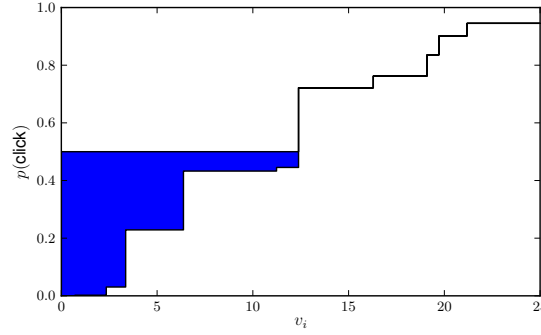


Figure 5.6: It is easy to compute incentive compatible pricing for a position auction. For every marginal increase in click probability that an agent gains by moving up a position, she must pay that probability times the minimum she must bid to be shown that position. Thus, her payment must be equal to the shaded area.

5.5 Third Analysis: Multiple Slots, Equilibrium Refinement

For our third analysis, we again considered multiple-slot settings. In this case we solved the equilibrium selection problem by considering the perfect-information Nash equilibrium in which each agent's expected payment is equal to what she would pay in a dominant-strategy truthful mechanism with the same allocation function as the corresponding GSP variant. This refinement has been used extensively in the analysis of vanilla GSP, where it is the unique Nash equilibrium equivalent to VCG's truthful equilibrium (i.e., one that chooses the same outcome and charges the same expected payments). When applied to vanilla GSP, this equilibrium has a number of desirable properties:

- It is guaranteed to exist (provided that bids are continuous) and is computable in polynomial time [1].
- The outcome is a competitive, symmetric and envy-free equilibrium [30, 103]
- The equilibrium is impersonation-proof [53].
- It does not violate the non-contradiction criterion (i.e., we should not be interested in equilibria of the perfect-information game that generate more expected revenue than the optimal auction) [29].

This equilibrium refinement can also be applied to other GSP variants ([60] used it to analyze squashing, and [29] used it to analyze reserve prices). This equilibrium is guaranteed to exist for all of our GSP variants because they are all monotonic (i.e., increasing an agent's bid weakly increases his position and therefore his expected number of clicks).

Although we focus on this equilibrium for mainly economic reasons, this choice also has computational advantages. Specifically, it is possible to compute the payments very quickly using the algorithm of Aggarwal *et al* [1]. (See Figure 5.6.)

For the experiments presented in this section, we considered two distributions:

- **Uniform**, where each agent’s valuation is drawn from $U(0, 25)$, each agent’s quality score is drawn from $U(0, 1)$, and α_{k+1} is drawn from $U(0, \alpha_k)$ (as in Section 5.4);
- **Log-normal**, where each agent’s valuation and quality are drawn from log-normal distributions, and valuation and quality are positively correlated using a Gaussian copula [77]. The exact parameters were provided to us by Sébastien Lahaie of Yahoo! Research, who derived them from confidential bid data. (To maintain confidentiality, the bids were scaled by arbitrary positive constants. Thus, the bids, prices and revenue in our results are not in any standard unit of currency.) Although we cannot disclose them, we can say that the distribution is a refinement of the distribution studied in [60].

From the uniform distribution, we sampled 1000 5-bidder, 5-slot settings; for the log-normal distribution we sampled 10000 5-bidder, 5-slot settings.⁷ To explore auction parameters, we used a simple grid search. We varied reserve prices between 0 and 30 in steps of 2, between 30 and 100 in steps of 10, between 100 to 1000 in steps of 100, and from 1000 to 10000 in steps of 1000. We varied squashing power between 0 and 1 in steps of 0.25. Our objective was expected revenue averaged across all samples.

We began by investigating the optimal parameter settings for each mechanism. For squashing, the optimal squashing power was 0 for log-normal distributions (as was also observed in previous work [60]), but greater than zero for uniform distributions, where squashing was also somewhat less effective (see Figure 5.7). UWR and anchoring had similar optimal reserve prices, which were dramatically different from QWR’s optimal reserve (see Figure 5.8). Adding squashing to UWR produced some improvements and had little effect on the optimal reserve price (see Figure 5.9). QWR greatly benefited from squashing, but the optimal reserve price was extremely sensitive to the squashing parameter (see Figure 5.10).

We then compared GSP variants (summarized in Table 5.3). We found that among simple variants UWR was clearly superior to both squashing and QWR. The richer variants—anchoring, QWR+Sq and UWR+Sq—all performed comparably well (within $\sim 2\%$ of each other), though QWR+Sq was consistently the worst.

Interestingly, QWR+Sq was only competitive with the other top mechanisms when squashing power was set close to zero. Observe that squashing has two effects when added to QWR: it changes the ranking among bidders who exceed their reserve prices, but it also changes the reserve prices. As squashing power gets closer to zero, reserve prices tend towards UWR. We hypothesized that QWR+Sq only performed as well as it did because of this second effect. To tease apart these two properties, we tested them in isolation. Specifically, we tested (1) a GSP variant in which reserve prices were weighted by squashed quality but ranking among reserve-exceeding bidders was performed according to their true quality scores, and (2) a GSP variant in which reserve prices were weighted by true quality scores but rankings were done using squashed quality

⁷A substantial fraction of the expected revenue in log-normal settings comes from rare, high valuation bidders. Thus, the expected revenue had much higher variance than in the uniform case; we thus needed many more samples to reduce noise and obtain statistically significant results.

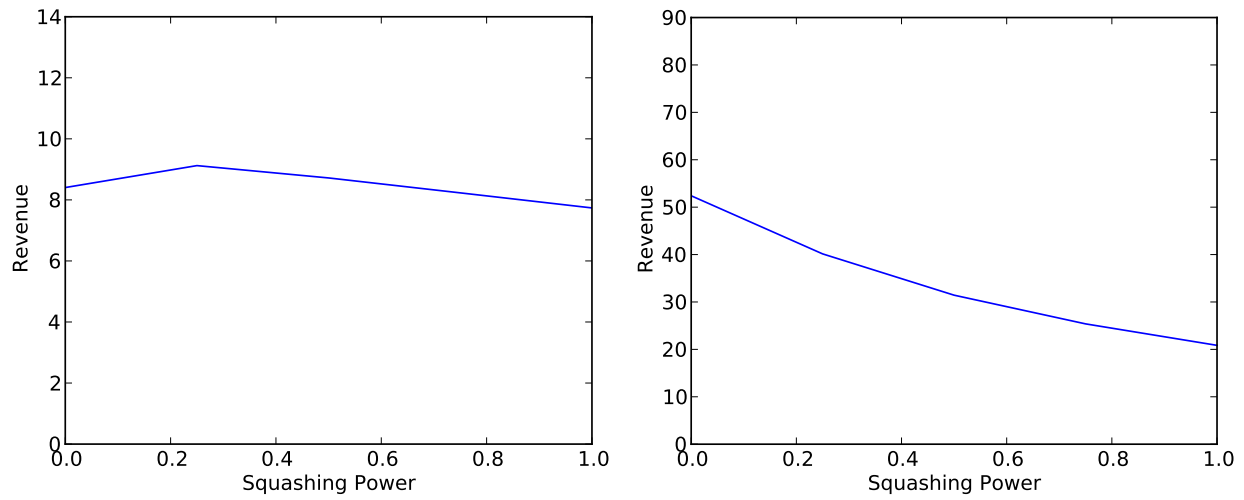


Figure 5.7: The optimal squashing power is zero or close to it. Squashing offers modest revenue gains given a uniform value distribution (left) and substantial gains given a log-normal value distribution (right).

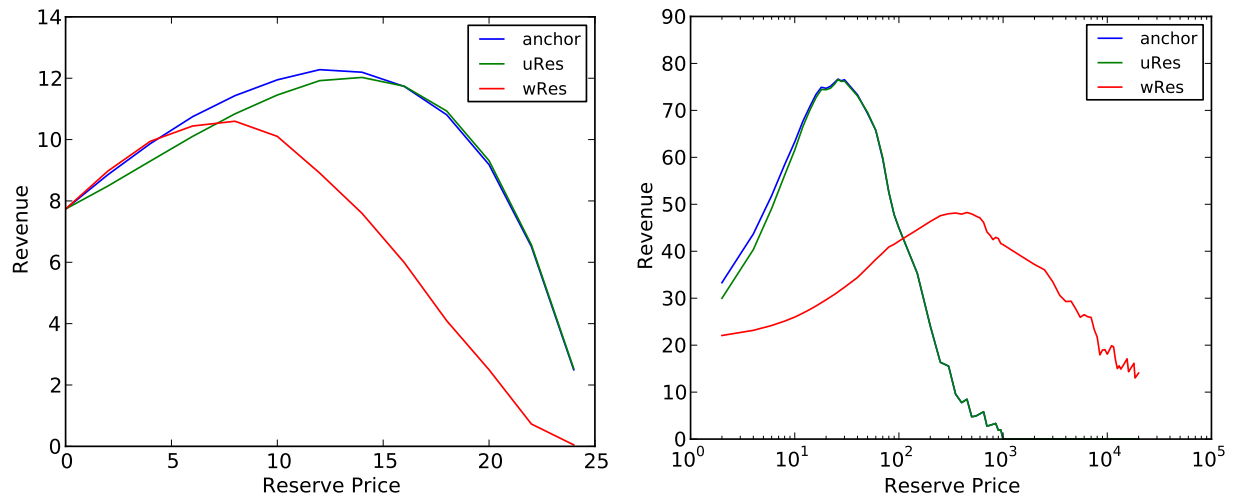


Figure 5.8: All three reserve-based variants (anchoring, QRW and UWR) provide substantial revenue gains. Anchoring is slightly better than UWR, and both are substantially better than QWR. (left: uniform; right: log-normal)

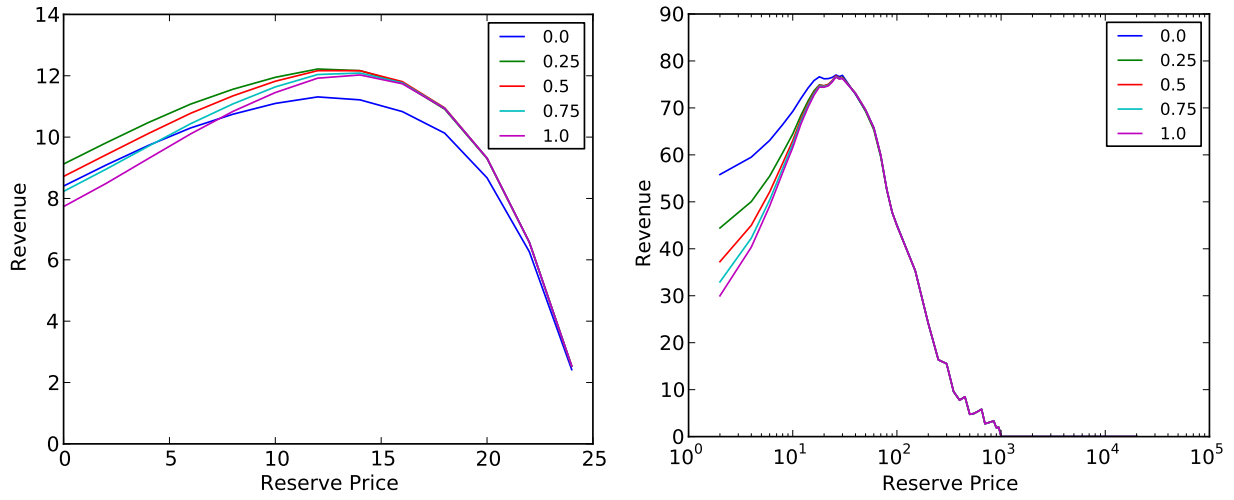


Figure 5.9: Adding squashing to UWR provides modest marginal improvements (compared to the optimal unweighted reserve price with no squashing) and does not substantially affect the optimal reserve price. (left: uniform; right: log-normal)

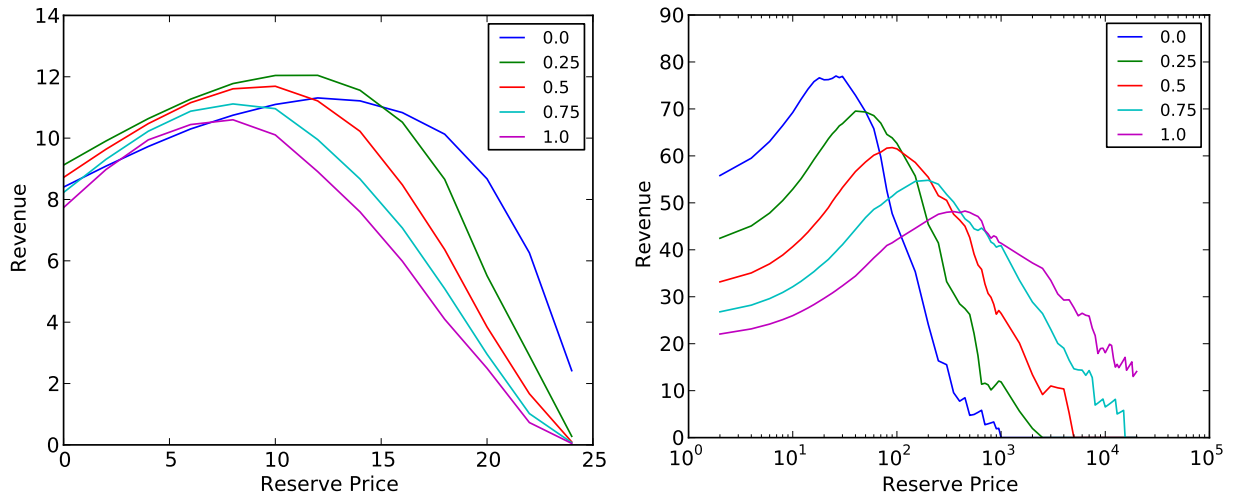


Figure 5.10: Adding squashing to QWR provides dramatic improvements. However, the higher the squashing power, the less the reserve prices are actually weighted by quality. In the case of a log-normal value distribution, the optimal parameter setting ($s = 0.0$) removes quality scores entirely and is thus equivalent to UWR. Note that different values of squashing power lead to dramatically different optimal reserve prices. (left: uniform; right: log-normal)

Auction	Revenue	Parameter(s)
Vanilla GSP	7.737	—
Squashing	9.123	$s = 0.25$
QWR	10.598	$r = 8.0$
UWR	12.026	$r = 14.0$
QWR+Sq	12.046	$r = 12.0, s = 0.25$
UWR+Sq	12.220	$r = 12.0, s = 0.25$
Anchoring	12.279	$r = 12.0$

Auction	Revenue	Parameter(s)
Vanilla GSP	20.454	—
QWR	48.071	$r = 400.0$
Squashing	53.349	$s = 0.0$
QWR+Sq	79.208	$r = 20.0, s = 0.0$
UWR	80.050	$r = 20.0$
Anchoring	80.156	$r = 20.0$
UWR+Sq	81.098	$r = 20.0, s = 0.5$

Table 5.3: Comparing the various auction variants given their optimal parameter settings. (Left: uniform distribution; right: log-normal distribution.) Note that in this analysis, vanilla GSP is equivalent to VCG. Bold indicates variants that are significantly better than all other variants, but not significantly different from each other. ($p \leq 0.05$ with Bonferroni correction.)

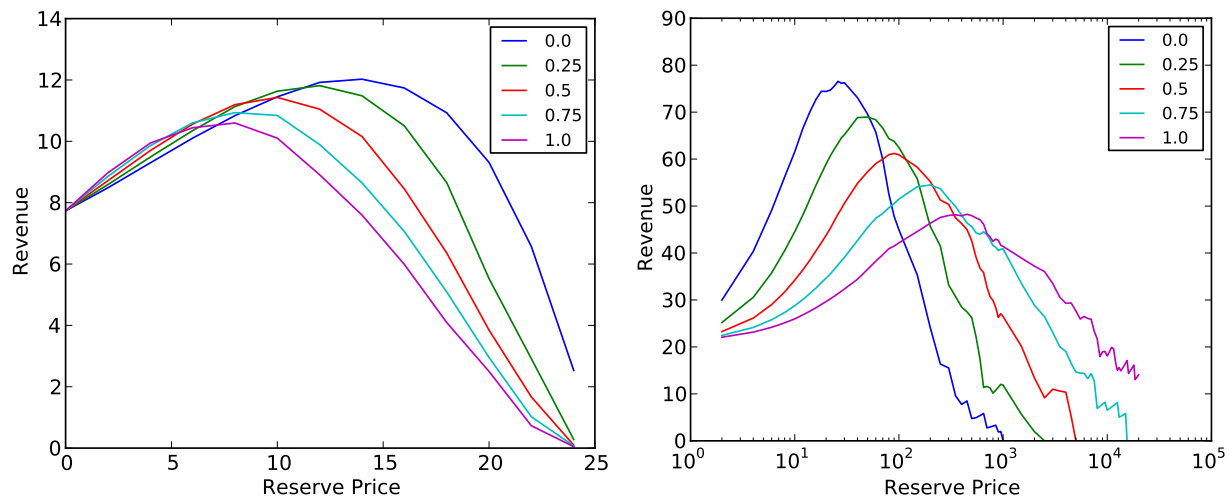


Figure 5.11: When squashing is only applied to reserve prices, it can dramatically increase QWR’s revenue. However, there has to be a lot of squashing (i.e., s close to 0), and the optimal reserve price is very dependent on the squashing power. In fact, for both distributions, the optimal parameters set $s = 0$, in which case the mechanism is identical to UWR.

scores. These experiments confirmed our hypothesis: the first variant (Figure 5.11)—which uses squashing to make the mechanism behave more like UWR—was much more effective at increasing revenue than the second (Figure 5.12).

Next, we investigated the effect of varying the number of bidders. Broadly, we found that the ranking among auctions remained consistent (see Figure 5.13). The optimal reserve price tended to increase with the number of bidders, particularly in the case of UWR. We found this especially interesting because a heuristic often described in the literature is to take Myerson’s optimal reserve prices and add them to GSP [e.g., 81]. Our results show that Myerson’s finding that the optimal reserve price does not vary with the number of bidders is only a property of the optimal auction, not of other auctions such as these GSP variants.

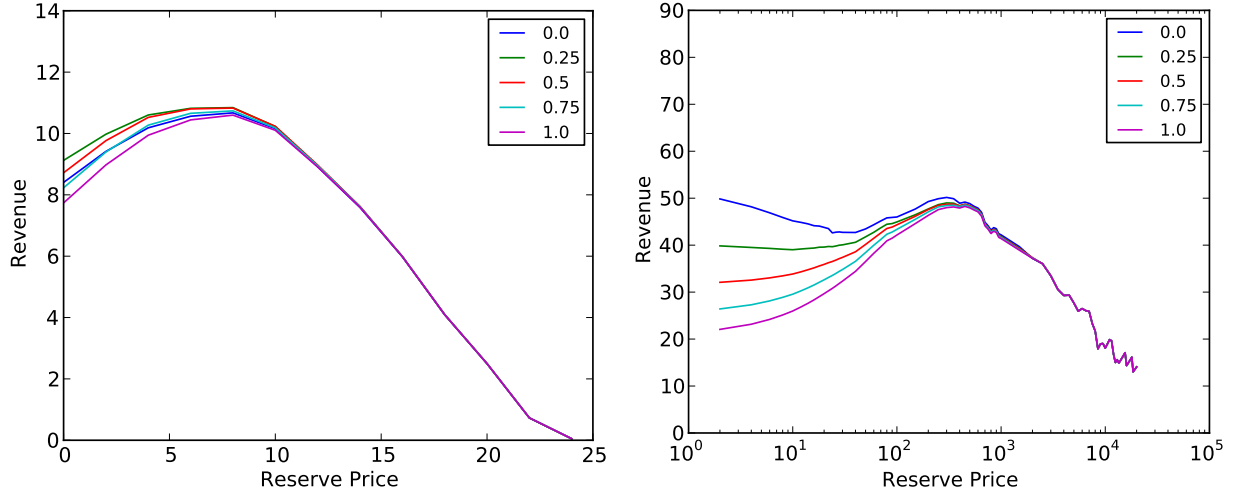


Figure 5.12: When squashing is only applied to ranking, but not to the reserve prices, the marginal gains from squashing over QWR (with the optimal reserve) are very small.

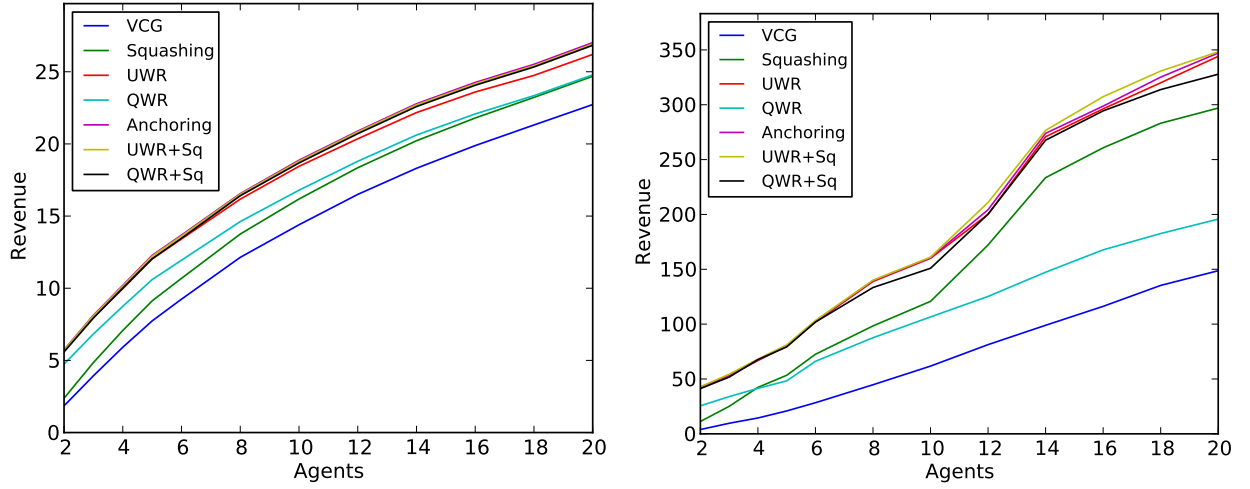


Figure 5.13: As the number of agents varies, the relative ranking of GSP variants remains mostly unchanged. The one notable exception to this is squashing and QWR, their ranking can be reversed depending on the distribution and number of bidders.

In summary, this section considered the equilibria of multi-slot ad auctions which are revenue equivalent to the truthful equilibria of corresponding dominant-strategy mechanisms with the same allocation rules, and considered both uniform and log-normal valuations. Our main conclusions were that:

1. As in the previous analyses, the mechanisms best able to optimize revenue were Anchoring, QWR+sq, UWR+sq and UWR; either squashing or QWR took fifth place with the other in sixth. Within these groups, the exact ranking depended on the distribution and the number of bidders.
2. QWR+sq only performed as well as it did when squashing was configured to make its reserve prices behave like UWR. When squashing is applied to the allocation in QWR, but not to the reserve prices, very little revenue improvement was possible.

5.6 Conclusions and Future Work

This chapter demonstrates how computational mechanism analysis can enable computational mechanism design. Despite the fact that we only explored a small space of mechanisms, we were able to discover new and important things about revenue-optimizing position auctions. First, we observed that the highest revenues were produced by mechanisms that (directly or effectively) use unweighted per-click reserve prices, an observation that we were able to replicate with two more-conventional methods of analysis. Second, we found that any kind of reserve price dramatically improved worst-case-equilibrium revenue, while vanilla GSP and squashing were very sensitive to equilibrium selection. However, we found that squashing could provide extra benefits when used in conjunction with UWR or QWR. This finding could not be made with existing techniques; CMA was necessary to explore the full space of equilibria. Third, our experimental findings also inspired new theoretical insights, including a characterization of the optimal mechanism for some settings (a novel auction called anchoring that is also nearly-optimal in settings where it is not optimal) and a characterization of the optimal reserve prices for a wide range of settings.

Our rather robust conclusion that UWR achieves higher revenues than QWR raises the question of why Google and Yahoo! both made the transition from unweighted to quality-weighted reserves. One likely explanation is that search engines do not aim to optimize their short-term revenue, but instead optimize long-term revenue via other short term objectives such as efficiency, user satisfaction, revenue under a constraint that ads are costly to show, etc. Other possibilities are that search engines are motivated by other business considerations entirely,⁸ that they have simply acted in error, or that our findings expose a flaw in the standard model of position auctions. Finally, it is possible that the premise of our question is wrong: perhaps search engines do not in fact use QWR, but instead use some other (secret) approach to setting reserves.

We believe that the most pressing open problem stemming from our work is to attempt to resolve these

⁸[81] report that they were explicitly instructed to use weighted reserve prices in their experiments because these are more consistent with an otherwise weighted auction and are perceived to be more fair by bidders.

questions by examining richer models that allow short-term revenue to be contrasted with longer-term revenue. Considering short-term revenue, we conjecture that in the field experiment of [81], where Yahoo! increased revenue by increasing reserve prices and simultaneously switching to QWR, the revenue increases would have been even greater if Yahoo! had retained optimized reserve prices but maintained UWR. To analyze longer-term revenue, a richer model could include quality and click probability that are determined by the advertiser's choice of ad text, rather than being exogenous. In equilibrium, this choice must be a best response to the rules of the auction and the choices of the other agents. For example, consider the problem of an advertiser who has two choices of ad text. One choice will yield 1000 clicks per hour, leading to 11 sales per hour. The other choice yields 10 clicks per hour, but every click produces a sale. With weighted reserve prices (and no squashing) the advertiser will always choose the first text, since it produces more sales per hour for the same price. With appropriate quality-weighted reserve prices (or squashing), the advertiser would choose the second, which generates nearly as many sales, and requires him to pay the reserve price far less often. It is not immediately clear which text the search engine should prefer: the first satisfies more users, but also wastes the time of many users who click through but do not buy.

Chapter 6

Application: Strategic Voting in Elections

6.1 Introduction

This chapter shows a dramatically different application for computational mechanism analysis: the study of strategic voting in elections. As with the auction applications we have discussed so far, the kinds of voting mechanisms that are used in practice tend to be very simple and structured in ways that make them amenable to compact representations (e.g., anonymity is a common feature). Unlike auctions, incentive compatible mechanisms are nearly non-existent (and those that do exist are so undesirable as to almost never be used in practice¹), so researchers tend to focus on mechanisms that are not incentive compatible.

Since voters may have an incentive vote strategically to influence an election's results, according to their knowledge or perceptions of others' preferences, much research has considered ways of limiting manipulation. This can be done by exploiting the computability limits of manipulations (e.g., finding voting mechanisms for which computing a beneficial manipulation is NP-hard [7, 8, 110]), by limiting the range of preferences (e.g., if preferences are single peaked, there exist non-manipulable mechanisms [31]), randomization [38, 85], etc.

When studying the problem of vote manipulation, nearly all research falls into two categories: coalitional manipulation and equilibrium analysis. Much research into coalitional manipulation considers models in which a group of truthful voters faces a group of manipulators who share a common goal. Less attention has been given to Nash equilibrium analysis which models the (arguably more realistic) situation where all voters are potential manipulators. One reason is that it is difficult to make crisp statements about this problem: strategic voting scenarios give rise to a multitude of Nash equilibria, many of which involve implausible outcomes. For example, even a candidate who is ranked last by all voters can be unanimously elected in a

¹A simple majority vote is incentive compatible, provided there are only two candidates. For settings with more than two candidates, the Gibbard-Satterthwaite theorem [37, 94] shows that every incentive compatible mechanism either 1) is a dictatorship where only a single agent's vote matters, or 2) is not onto, in the sense that it disqualifies all but at most two candidates *a priori*.

Nash equilibrium—observe that when facing this strategy profile, no voter gains from changing his vote. Another problem is that finding even a single Nash equilibrium of a game can be computationally expensive, and plurality votes can have exponentially many equilibria (in the number of voters).

Given the tools of CMA, this chapter naturally focuses on the second approach, analysis using the Nash (and subsequently, Bayes-Nash) equilibria of voting games. We focus on plurality, as it is by far the most common voting mechanism used in practice. We refine the set of equilibria by adding a small additional assumption: that agents realize a very small gain in utility from voting truthfully; we call this restriction a *truthfulness incentive*. We ensure that this incentive is small enough that it is always overwhelmed by the opportunity to be pivotal between any two candidates: that is, a voter always has a greater preference for swinging an election in the direction of his preference than for voting truthfully. All the same, this restriction is powerful enough to rule out the bad equilibrium described above, as well as being, in our view, a good model of reality, as voters might reasonably have a preference for voting truthfully.

The computational approach has not previously been used in the literature on strategic voting, because the resulting normal-form games are enormous. For example, representing some of our games (e.g., with 20 players and 3 candidates) in the normal form would require billions of payoff-table entries ($20 \times 3^{20} \simeq 6.97 \times 10^{10}$).

Our first contribution is an equilibrium analysis of full-information models of plurality elections. We analyze the number of Nash equilibria that exist when truthfulness incentives are present. We also examine the winners, asking questions like how often they also win the election in which all voters vote truthfully, or how often they are also Condorcet winners. We also investigate the social welfare of equilibria; for example, we find that it is very uncommon for the worst-case result to occur in equilibrium. Our approach can be generalized to richer mechanisms where agents vote for multiple candidates (i.e., approval, k -approval, and veto).

Our second contribution involves the arguably more realistic scenario in which the information available to voters is incomplete. We assume that voters know only a probability distribution over the preference orders of others, and hence identify Bayes-Nash equilibria. We found that although the truthfulness incentive eliminates the most implausible equilibria (i.e., where the vote is unanimous and completely independent of the voters preferences), many other equilibria remain. Similarly to Duverger’s law (which claims that plurality election systems favor a two-party result [28], but does not directly apply to our setting), we found that a close race between almost any pair of candidates was possible in equilibrium. Equilibria supporting three or more candidates were possible, but less common.

6.1.1 Related Work

Analyzing equilibria in voting scenarios has been the subject of much work, with many researchers proposing various frameworks with limits and presumptions to deal with both the sheer number of equilibria, and to deal with more realistic situations, where there is limited information. Early work in this area, by McKelvey and Wendell [67], allowed for abstention, and defined an equilibrium as one with a Condorcet winner. As this is a very strong requirement, such an equilibrium does not always exist, but they established some criteria for this equilibrium that depend on voters' utilities.

Myerson and Weber [76] wrote an influential article dealing with the Nash equilibria of voting games. Their model assumes that players only know the probability of a tie occurring between each pair of players, and that players may abstain (for which they have a slight preference). They show that multiple equilibria exist, and note problems with Nash equilibrium as a solution concept in this setting. The model was further studied and expanded in subsequent research [21, 54]. Assuming a slightly different model, Messner and Polborn [70], dealing with perturbations (i.e., the possibility that the recorded vote will be different than intended), showed that equilibria only includes two candidates (Duverger's law). Our results, using a different model of partial information (Bayes-Nash), show that with the truthfulness incentive, there is a certain tendency towards such equilibria, but it is far from universal.

Looking at iterative processes makes handling the complexity of considering all players as manipulators simpler. Dhillon and Lockwood [23] dealt with the large number of equilibria by using an iterative process that eliminates weakly dominated strategies (a requirement also in Feddersen and Pesendorfer's definition of equilibrium [32]), and showed criteria for an election to result in a single winner via this process. Using a different process, Meir et al. [69] and Lev and Rosenschein [62] used an iterative process to reach a Nash equilibrium, allowing players to change their strategies after an initial vote with the aim of myopically maximizing utility at each stage.

Dealing more specifically with the case of abstentions, Desmedt and Elkind [22] examined both a Nash equilibrium (with complete information of others' preferences) and an iterative voting protocol, in which every voter is aware of the behavior of previous voters (a model somewhat similar to that considered by Xia and Contizer [109]). Their model assumes that voting has a positive cost, which encourages voters to abstain; this is similar in spirit to our model's incentive for voting truthfully, although in this case voters are driven to withdraw from the mechanism rather than to participate. However, their results in the simultaneous vote are sensitive to their specific model's properties.

Rewarding truthfulness with a small utility has been used in some research, though not in our settings. Laslier and Weibull [61] encouraged truthfulness by inserting a small amount of randomness to jury-type games, resulting in a unique truthful equilibrium. A more general result has been shown in Dutta and Sen [27], where they included a subset of participants which, as in our model, would vote truthfully if it would not change the

result. They show that in such cases, many social choice functions (those that satisfy the No Veto Power) are Nash-implementable, i.e., there exists a mechanism in which Nash equilibria correspond to the voting rule. However, as they acknowledge, the mechanism is highly synthetic, and, in general, implementability does not help us understand voting and elections, as these involve a predetermined mechanism. The work of Dutta and Laslier [26] is more similar to our approach. They use a model where voters have a lexicographic preference for truthfulness, and study more realistic mechanisms. They demonstrated that in plurality elections with odd numbers of voters, this preference for truthfulness can eliminate all pure-strategy Nash equilibria. They also studied a mechanism strategically equivalent to approval voting (though they used an unusual naming convention), and found that when a Condorcet winner exists, there is always a pure-strategy Nash equilibrium where the Condorcet winner is elected.

6.2 Definitions

Elections are made up of candidates, voters, and a mechanism to decide upon a winner.

Definition 1. *Let C be a set of m candidates, and let A be the set of all possible preference orders over C . Let V be a set of n voters. Every voter $v_i \in V$ has some element in A which is his true, “real” value (which we shall mark as a_i), and some element of A that he announces as his value, which we shall denote as \tilde{a}_i . The voting mechanism is a function $f : A^n \rightarrow C$.*

Note that our definition of a voter incorporates the possibility of him announcing a value different than his true value (strategic voting).

In this work, we restrict our attention to scoring rules, i.e., voting rules in which each voter assigns a certain number of points to each candidate, and the candidate with the most points wins. Specifically, we study scoring rules in which each candidate can get at most 1 point from each voter. We focus on four mechanisms:

- **Plurality:** A single point is given to one candidate.
- **Veto:** A point is given to everyone except one candidate.
- **k-approval:** A point is given to exactly k candidates.
- **Approval:** A point is given to as many candidates as each voter chooses.

Another important concept is that of a Condorcet winner.

Definition 2. *A Condorcet winner is a candidate $c \in C$ such that for every other candidate $d \in C$ ($d \neq c$) the number of voters that rank c over d is at least $\lfloor \frac{n}{2} \rfloor + 1$.*

Condorcet winners do not exist in every voting scenario, and many voting rules—including plurality—are not Condorcet-consistent (i.e., even when there is a Condorcet winner, that candidate may lose).

To reason about the equilibria of voting systems, we need to formally describe them as games, and hence to map agents' preference relations to utility functions. More formally, each agent i must have a utility function $u_i : A^n \mapsto \mathbb{R}$, where $u_i(a_V) > u_i(a'_V)$ indicates that i prefers the case when all the agents have voted a_V over the case when the agents vote a'_V . Representing preferences as utilities rather than explicit rankings allows for the case where i is uncertain about what outcome will occur. This can arise either because he is uncertain about the outcome given everyone's actions (because of random tie-breaking rules), or because he is uncertain about the actions the other agents will take (e.g., agents behaving randomly; agents playing strategies that condition on information that i does not observe). Here we assume that an agent's utility only depends on the candidate that gets elected and on his own actions (e.g., an agent can get some utility for voting truthfully). Thus, we obtain simpler utility functions $u_i : C \times A \mapsto \mathbb{R}$, with an agent i 's preference for outcome a_V denoted $u_i(f(a_V), \tilde{a}_i)$.

In this work, we consider two models of games, full-information games and symmetric Bayesian games. In both models, each agent must choose an action \tilde{a}_i without conditioning on any information revealed by the voting method or by the other agents. In a full-information game, each agent has a fixed utility function which is common knowledge to all the others. In a symmetric Bayesian game, each agent's utility function (or "type") is an independent, identically distributed draw from a commonly known distribution of the space of possible utility functions, and each agent must choose an action without knowing the types of the other agents, while seeking to maximize his expected utility.

We consider a plurality voting setting with voters' preferences chosen randomly. We show detailed results for the case of 10 voters and 5 candidates (numbers chosen to give a setting both computable and with a range of candidates), but we also show that changing these numbers results in qualitatively similar equilibria.

Suppose voter i has a preference order of $a^5 \succ a^4 \succ \dots \succ a^1$, and the winner when voters voted a_V is a^j . We then define i 's utility function as

$$u_i(f(a_V), \tilde{a}_i) = u_i(a^j, \tilde{a}_i) = \begin{cases} j & a_i \neq \tilde{a}_i \\ j + \varepsilon & a_i = \tilde{a}_i, \end{cases}$$

with $\varepsilon = 10^{-6}$.

Note that we use utilities because we need, when computing an agent's best response, to be able to compare nearly arbitrary distributions over outcomes (e.g., for mixed strategies or Bayesian games). This is not meant to imply that utilities are transferable in this setting. Most of our equilibria would be unchanged if we moved to a different utility model, provided that the preferences were still strict, and the utility differences between outcomes were large relative to ε . The one key distinction is that agents are more likely to be indifferent to lotteries (e.g., an agent that prefers $A \succ B \succ C$ is indifferent between $\{A, B, C\}$ and $\{B\}$) than under some other utility models.

As with perfect information games, we consider Bayesian games with a fixed number of candidates (m) and voters (n). The key difference is that the agents' preferences are not *ex ante* common knowledge. Instead, each agent's preferences are drawn from a distribution $p_i : A \mapsto \mathbb{R}$. Here we consider the case of symmetric Bayesian games, where every agent's preferences are drawn independently from the same distribution, p . Due to computational limits, we cannot study games where p has full support; each agent would have $5^{51} \simeq 7.5 \times 10^{83}$ pure strategies. Instead, we consider distributions where only a small subset of preference orders are in the support of p . We generate distributions by choosing six preference orderings, uniformly at random (this gives a more reasonable $5^6 = 15625$ pure strategies). For each of these orderings a , we draw $p(a)$ from a uniform $[0, 1]$ distribution. These probabilities are then normalized to sum to one. This restricted support only affects what preference orders the agents can have; we do not restrict agents' action sets in any way.

Note that formally the ε truthfulness incentive represents a change to the game, rather than a change in the solution concept. However, there is an equivalence between the two approaches: for any sufficiently small ε , the set of pure-strategy Nash equilibria in the game with ε truthfulness incentives is identical to the set of pure-strategy Nash equilibria (of the game without truthfulness incentives) that also satisfy the property that only the pivotal agents (i.e., agents who, were their vote to change, the outcome would change) deviate from truthfulness. The meaning of sufficiently small depends on the agents' utility functions, and on the tie-breaking rule. If \underline{u} is the difference in utility between two outcomes, and \underline{t} is the minimum joint probability of any type profile (in a Bayesian game), then ε must be less than $\underline{ct}/|C|$ (the $1/|C|$ factor comes from the fact that uniform tie-breaking can select some candidate with that probability).

6.3 Method

Encoding plurality games as action-graph games is relatively straightforward. For each set of voters with identical preferences, we create one action node for each possible way of voting. For each candidate, we create a sum node that counts how many votes the candidate receives. Directed edges encode which vote actions contribute to a candidate's score, and that every action's payoff can depend on the scores of all the candidates (see Figure 6.1). The same approach generalizes to approval-based mechanisms; if an action involves approving more than one candidate, then there must be an edge from that action node to each approved candidate's sum node. Similarly, positional scoring rules (e.g., Borda) can be encoded by using weighted sum nodes.

A variety of Nash-equilibrium-finding algorithms exist for action-graph games [19, 51]. In this work, we used the support enumeration method (see [84, 100] and Chapter 3) exclusively, because it allows Nash equilibrium enumeration. This algorithm works by iterating over possible supports, testing each for the existence of a Nash equilibrium. In the worst case, this requires exponential time, but in practice SEM's heuristics (exploiting symmetry and conditional dominance) enable it to find all the pure-strategy Nash

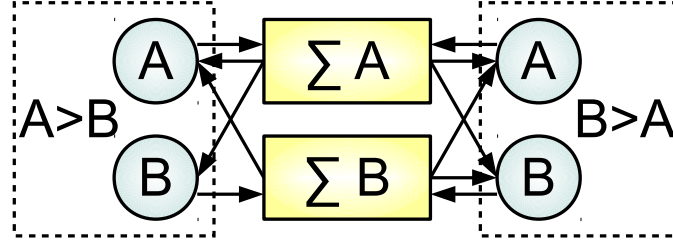


Figure 6.1: An action graph game encoding of a simple two-candidate plurality vote. Each round node represents an action a voter can choose. Dashed-line boxes define which actions are open to a voter given his preferences; in a Bayesian AGG, an agent’s type determines the box from which he is allowed to choose his actions. Each square is a sum node, tallying the number of votes a candidate received.

equilibria (PSNEs) of a game quickly.

We represented our symmetric Bayesian games using a Bayesian game extension to action-graph games [47]. Because we were concerned only with *symmetric* pure Bayes-Nash equilibria, it remained feasible to search for every equilibrium with SEM. (No pruning of supports is possible because dominance checking of BAGGs is NP-hard. Thus, SEM in this context essentially amounts to pure brute-force search.)

6.4 Pure-Strategy Nash Equilibrium Results

To examine pure strategies, we ran 1000 voting experiments using plurality with 10 voters and 5 candidates.

6.4.1 Selectiveness of the Truthfulness Incentive

The logical first question to ask about our truthfulness incentive is whether or not it is effective as a way of reducing the set of Nash equilibria to manageable sizes. As a baseline, when we did not use any equilibrium selection method, each plurality game had over a million PSNEs. A stronger baseline is the number of PSNEs that survive removal of weakly dominated strategies (RDS). RDS reduces the set by an order of magnitude, but still allows over 100,000 PSNEs per game to survive. In contrast, the truthfulness incentive reduced the number of PSNEs down to 30 or fewer, with the median game having only 3. Interestingly, a handful of games (1.1%) had no PSNEs. Laslier and Dutta [26] had shown that PSNEs were not guaranteed to exist, but only when the number of voters is odd (and at least 5). Our results show that the same phenomenon can occur, albeit infrequently, when the number of voters is even.

One of the problems with unrestricted Nash equilibria is that there are so many of them; the other problem is that they are compatible with any outcome. Given that the TI is so effective at reducing the set of PSNEs, one could wonder whether or not the TI is helpful for the second problem. Unfortunately, the TI had only limited effectiveness in ruling some outcomes as impossible: only 4.4% of games support exactly one outcome in

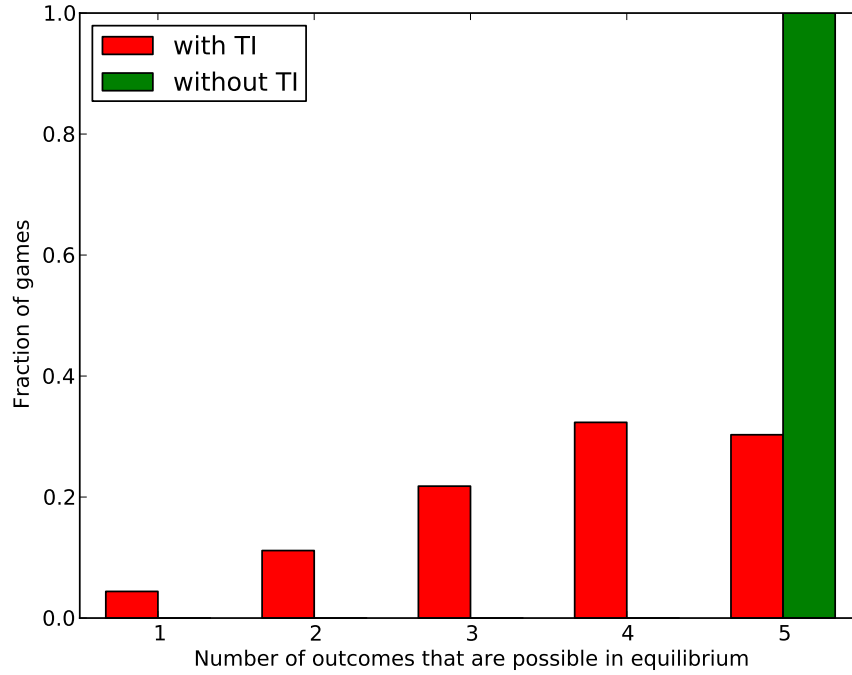


Figure 6.2: Even with the truthfulness incentive, many different outcomes were still possible in equilibrium.

equilibrium. However, nearly always ($> 99\%$ of the time) some outcomes occur more frequently than others. See Figures 6.2 and 6.3.

6.4.2 Equilibrium Outcomes

With a workable equilibrium selection method, we can now consider the question of what kinds of outcomes occur in plurality. We shall examine two aspects of the results: the preponderance of equilibria with victors being the voting method’s winners, and Condorcet winners. Then, moving to the wider concept of social welfare of the equilibria (which we can consider since we work with utility functions), we examine both the social welfare of the truthful voting rule vs. best and worse possible Nash equilibria and the average rank of the winners in the various equilibria.

The first issues to consider are to what extent truthful voting is an equilibrium, and to what extent the agents cancel out each other’s manipulations (i.e., when there are non-truthful Nash equilibria that lead to the same outcome as truthful voting). We call a candidate the “truthful winner” iff that candidate wins when voters vote truthfully. For 63% of the games, the truthful preferences were a Nash equilibrium, but more interestingly, many of the Nash equilibria reached the same result as the truthful preferences: 80% of the games had at least one equilibrium where the truthful winner wins, and looking at the multitude of equilibria, 42% elected the truthful winner (out of games with a truthful result as an equilibrium, the share was 52%). Without the

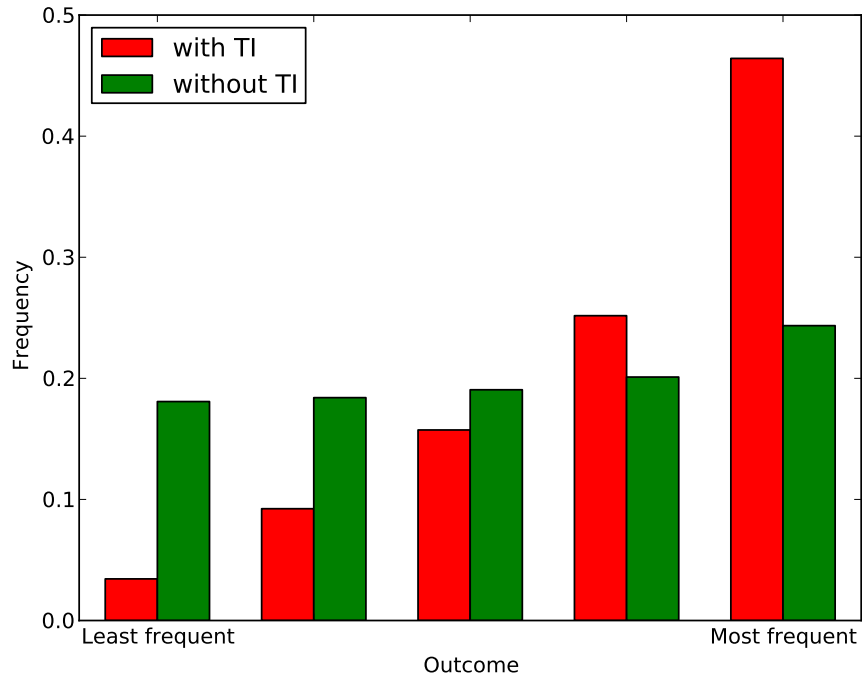


Figure 6.3: With the truthfulness incentive, some outcomes occur much more frequently than others.

truthfulness incentive, the truthful winner only won 22% of equilibria.

Next, we turn to the question of whether or not strategic voting leads to the election of good candidates, starting with Condorcet winners. 55% of games had Condorcet winners, which would be elected by truthful voting in 49% of the games (not a surprising result; plurality is known not to be Condorcet consistent). However, the combination of truthful voting both being an equilibrium and electing a Condorcet winner only occurred in 42% of games. In contrast, a Condorcet winner could win in some Nash equilibrium in 51% of games (though only 36% of games would elect a Condorcet winner in every equilibrium).

Turning to look at the social welfare of equilibria, once again, the existence of the truthfulness incentive enabled us to reach “better” equilibria. In 93% of the cases, the worst-case outcome was not possible at all (recall that without the truthfulness incentive, every result is possible in some Nash equilibrium), while only in 30% of cases, the best outcome was not possible. While truthful voting led to the best possible outcome in 59% of cases, truthful voting was still stochastically dominated by the best-case Nash equilibrium (see Figure 6.4).

When looking at the distribution of welfare throughout the multitudes of equilibria, one can see that the concentration of the equilibria is around high-ranking candidates, as the average share of equilibria by candidates with an average ranking (across all voters in the election) of less than 1 was 56%. (See Figure 6.5.) Fully 72%, on average, of the winners in every experiment had above (or equal) the median rank, and in more than half the experiments (52%) all equilibria winners had a larger score than the median. As a comparison,

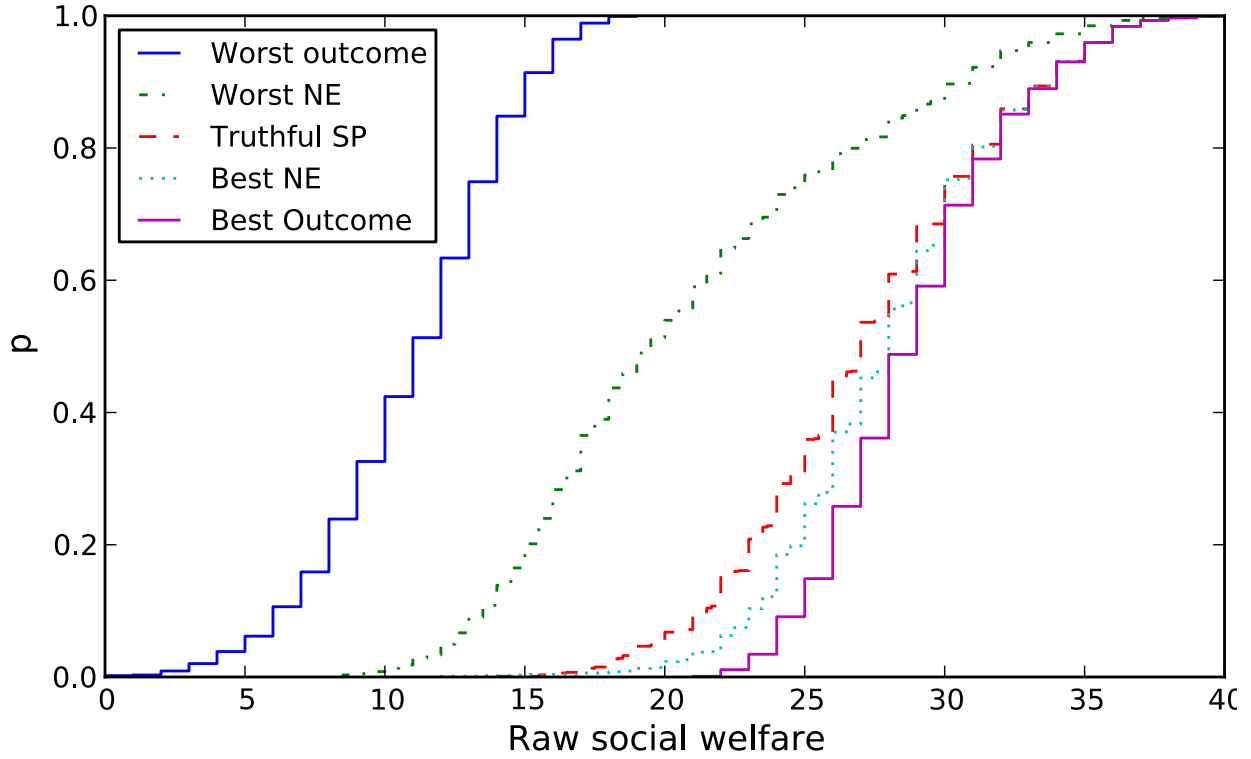


Figure 6.4: CDF of social welfare.

the numbers from experiments without the truthfulness incentive, are quite different: candidates—whatever their average rank—won, with minor fluctuations, about the same number of equilibria (57% of winners, were, on average, above or equal to the median rank).

6.4.3 Scaling Behavior and Stability

We next varied the number of voters and candidates. Our main finding was that when voter number was odd, the probability that a game had no equilibria at all increased dramatically, as the truthfulness incentive caused many such situations to be unstable (see Figure 6.6). Less surprisingly, as the number of candidates increased PSNEs were less likely to exist.

Nevertheless, these equilibria equilibria retained the properties we have seen—a concentration of equilibria around “quality” candidates, such as truthful and Condorcet winners, as can be seen in Figure 6.7a, for truthful winners (never below 40%) and in Figure 6.7b for Condorcet winners. These effects were even more pronounced with an odd number of voters, as the number of equilibria was so small.

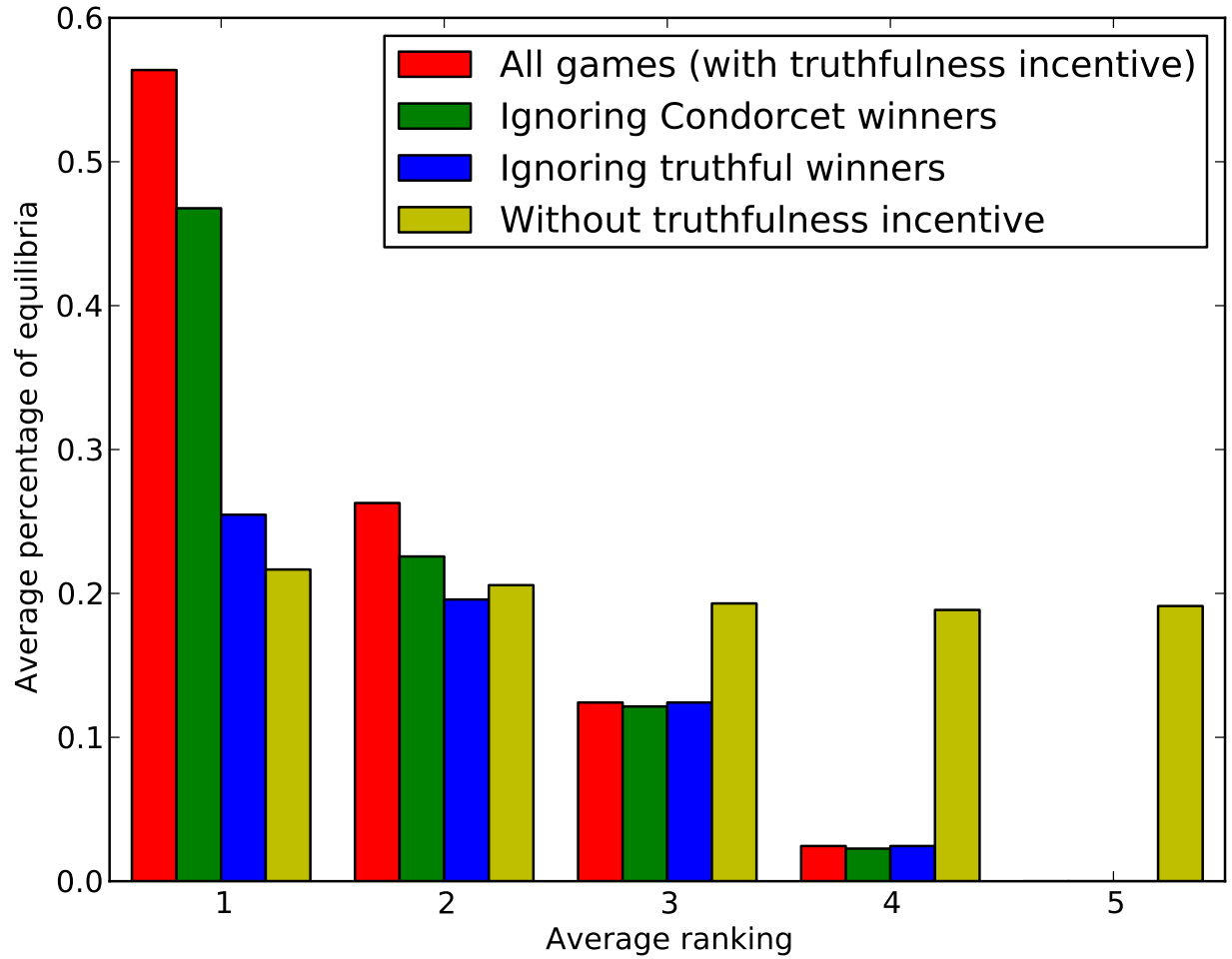


Figure 6.5: The average proportion of equilibria won by candidates with average rank of 0–1, 1–2, etc.

6.4.4 Richer Mechanisms

Approval (and variants such as k -approval and veto) are straightforward extensions of plurality, so it is natural to consider whether our approach will work similarly well for these mechanisms. Also, Laslier and Dutta [26] were able to resolve the existence problem for plurality and approval, but noted that the existence of PSNEs for k -approval was an open problem.

Thus, we started by investigating k -approval and veto. As was the case with plurality, the truthfulness incentive kept the number of equilibria manageable. As can be seen from Figure 6.8, in all cases more than 75% of games had 35 equilibria or fewer, and it seems that the number of equilibria roughly increases with the number of candidates. Our data also allowed us to resolve the open problem of equilibrium existence: for every value of k and m we observed at least one instance without any PSNE.

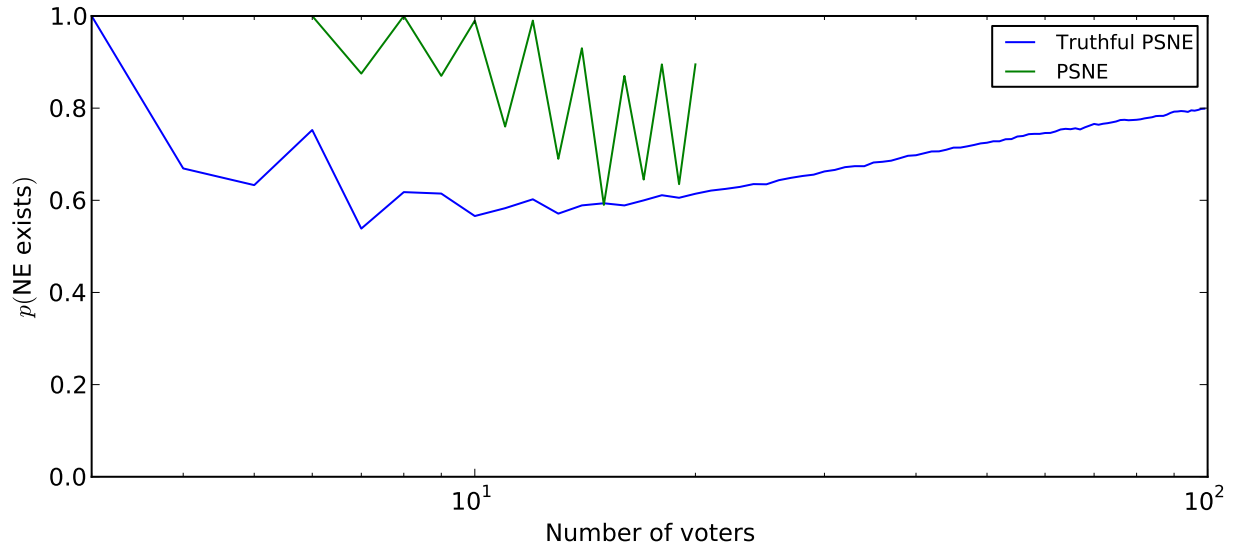
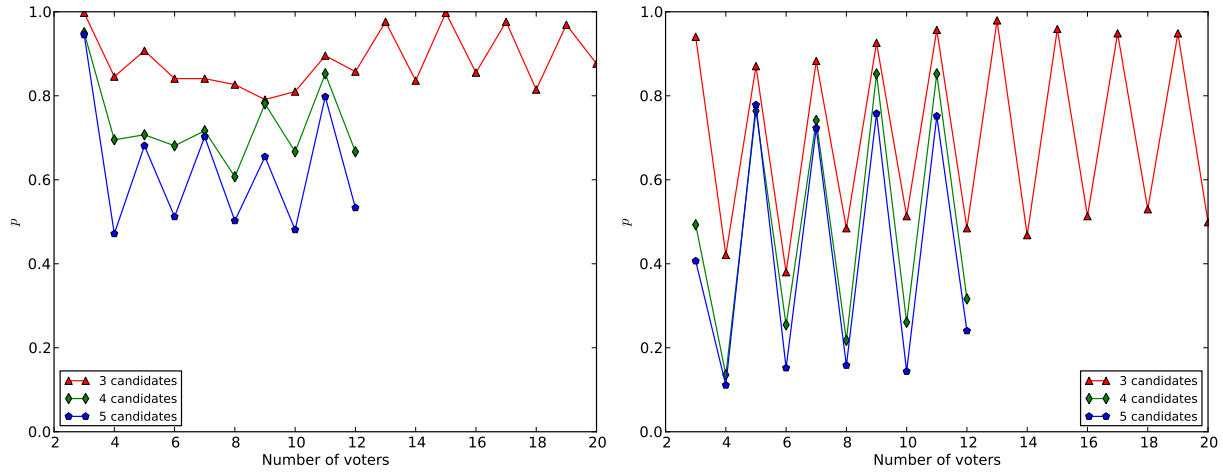


Figure 6.6: Percentage of games with a Nash equilibrium of a given type with 3 candidates and varying number of voters.



(a) Percentage of equilibria that elect the truthful winner.

(b) Percentage of equilibria that elect the Condorcet winner.

Figure 6.7: Varying the number of voters and candidates.

We also considered approval voting. Laslier and Dutta have already shown that a Condorcet consistent equilibrium is always guaranteed to exist. However, this raises an interesting question: are there other Nash equilibria? In our experiments, we found that approval voting had an extremely large number of equilibria (over 200,000 per game), so it seems that the addition of another dimension (allowing each voter to decide how many candidates to vote for), reduces the effectiveness of the truthfulness incentive.

Looking at the equilibrium outcomes, we found that they maintained some of the qualities that were present

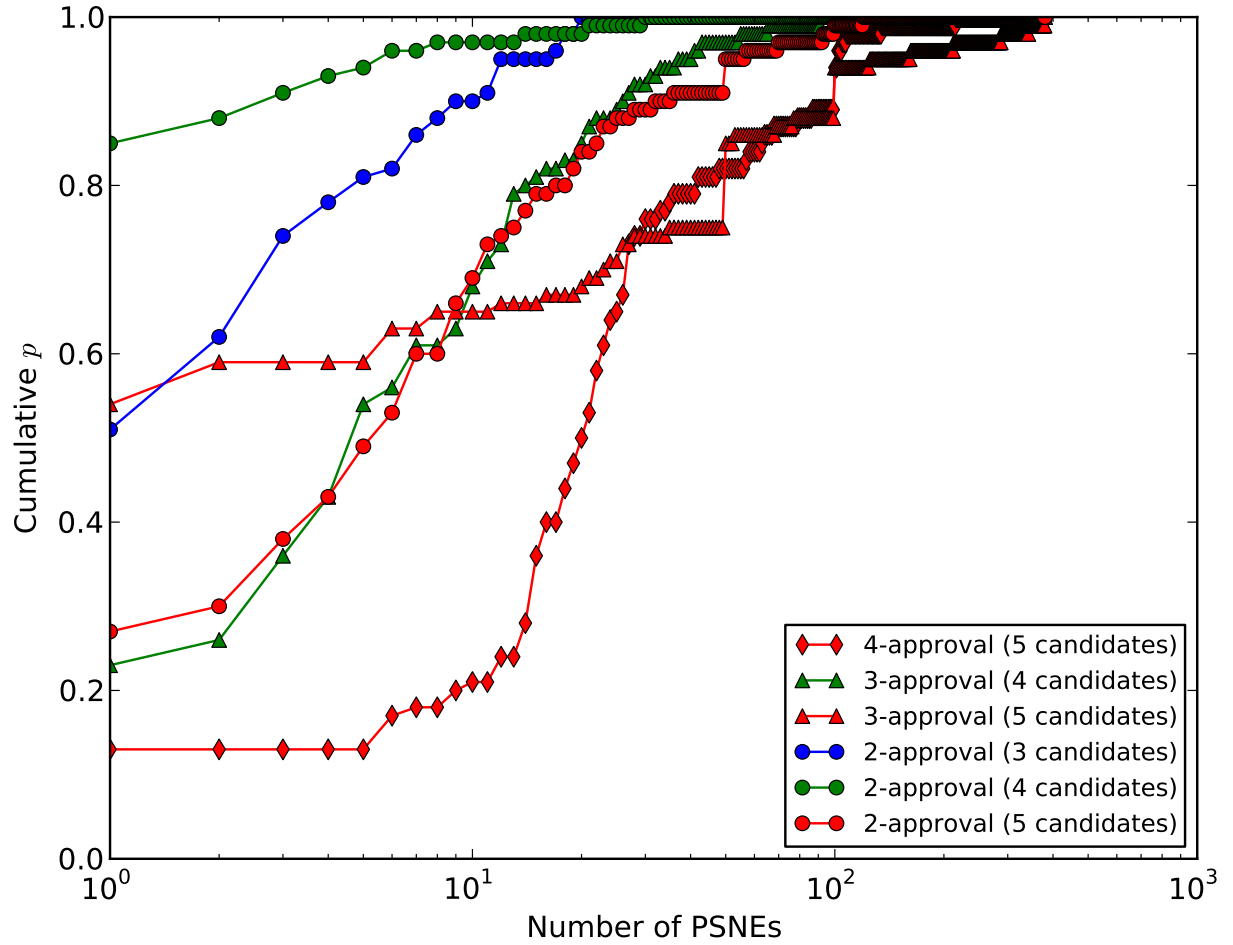


Figure 6.8: Empirical CDF of counts of equilibria.

in plurality equilibrium outcomes: truthful winners won, on average, in over 30% of the equilibria in every setting, and sometimes more (e.g., 2-approval with 4 candidates resulted in almost 50% of equilibria, on average, electing truthful winners). However, Condorcet winners were not similarly common in equilibrium, and it seems that as the number of candidates grows, and as the number of candidates to which voters allot points to increases, the percent of equilibria with a Condorcet winner drops (so in 2-approval such equilibria are common, in 3-approval somewhat less so, and in 4-approval even less).

6.5 Bayes-Nash Equilibria Results

Moving beyond the full-information assumption, we considered plurality votes where agents have incomplete information about each other's preferences. In particular, we assumed that agents have independent, identically distributed (but not necessarily uniformly distributed) preferences, and that each agent knows only his own

preferences and the (commonly-known) prior distribution. Again, we considered the case of 10 voters and 5 candidates, but now also introduced 6 possible types for each voter. For each of 50 games, we computed the set of all symmetric pure-strategy Bayes-Nash equilibria, both with and without the ε -truthfulness incentive.²

Our first concern was studying how many equilibria each game had and how the truthfulness incentive affected the number of equilibria. The set of equilibria was small (< 28 in every game) when the truthfulness incentive was present. Surprisingly though, removing the truthfulness incentive added only a few equilibria. In fact, in the majority of games (76%), there were exactly five new equilibria: one for each strategy profile in which all types vote for a single candidate. Looking into the structure of these equilibria, we found two interesting, and seemingly contradictory, properties. First, most equilibria (95%) only ever involved two or three candidates (i.e., voters only voted for a limited set of candidates). Second, every candidate was involved in some equilibrium. Thus, we can identify an equilibrium by the number of candidates it involves (see Figure 6.9). Notably, most equilibria involved only two candidates, with each type voting for their most preferred candidate of the pair. Further, most games had 10 such equilibria, one for every possible pair. There were two reasons why some pairs of candidates did not have corresponding equilibria in some games. First, sometimes one candidate Pareto-dominated the other (i.e., was preferred by every type). Second, sometimes the types that liked one candidate were so unlikely to be sampled that close races occurred with extremely low probability (relative to ε); in such cases, agents preferred to be deterministically truthful than pivotal with very small probability. This observation allowed us to derive the following theoretical result about when a 2-candidate equilibrium will exist.

Theorem 14. *In any symmetric Bayesian plurality election game (with $n \geq 2$), for any pair of candidates c_1, c_2 , one of the following conditions is true:*

- *with some positive ε truthfulness incentive, there exists a Bayes-Nash equilibrium where each voter votes for his most preferred of c_1, c_2 ; or*
- *one of the candidates Pareto-dominates the other ex ante (i.e., the probability that a voter prefers the second candidate to the first is zero).*

Sketch. If c_1 Pareto dominates c_2 , then every agent would vote for c_1 and no agent could influence the outcome. For any non-zero ε , voters would deviate to honest voting instead of c_1 .

So as long as there is non-zero probability of some agent preferring c_2 to c_1 , every agent has a non-zero probability of being pivotal between those two outcomes (and a zero probability of being pivotal between any two other outcomes). For a sufficiently small ε , the value of influencing the outcome overwhelms the value of truthful voting. □

²We omitted two games from our results. The omitted games each have a type with very low probability. For some profiles, the probability of agents with these types being pivotal was less than machine- ε . This led to SEM finding “Bayes-Nash equilibria” that were actually only ε -Bayes-Nash.

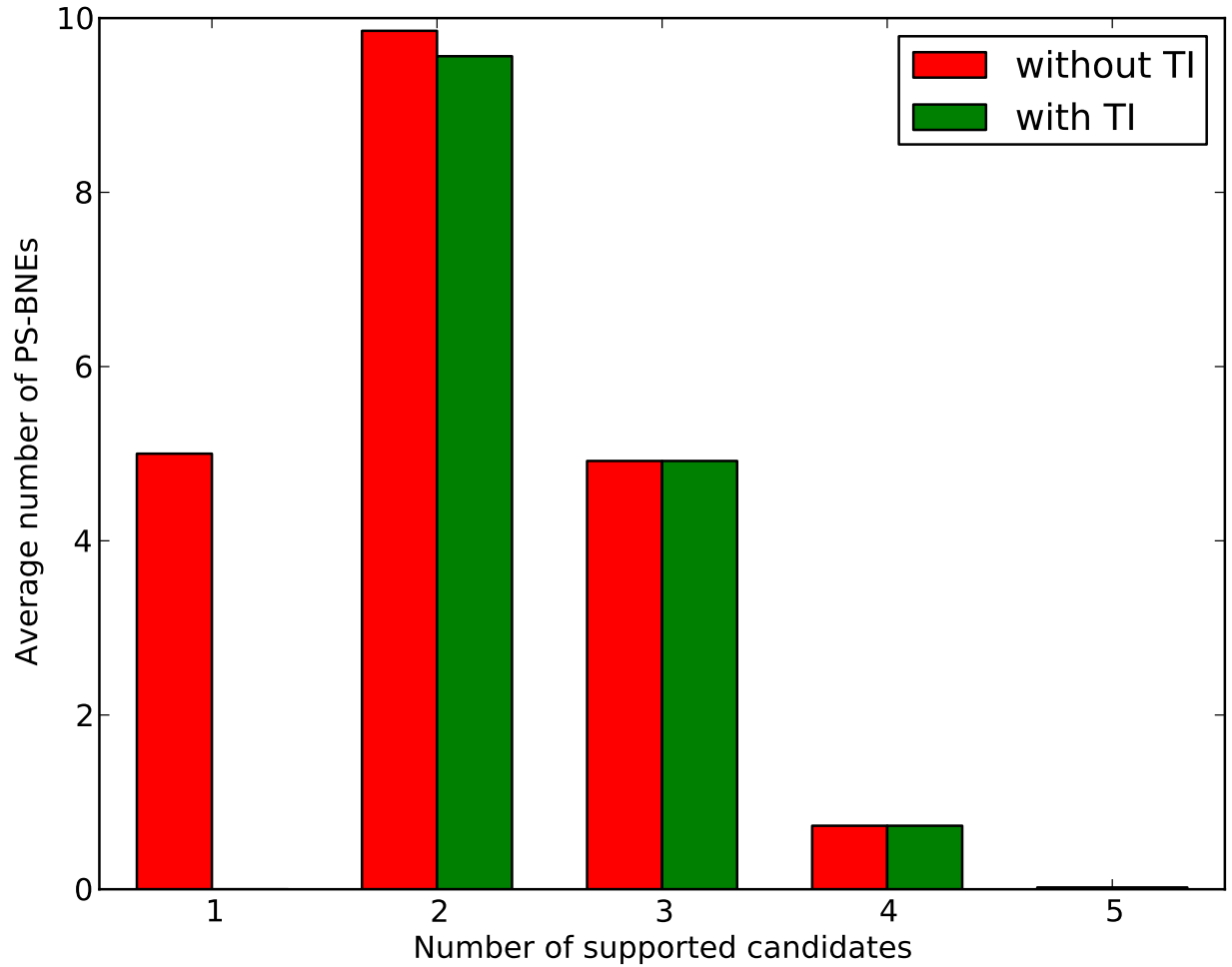


Figure 6.9: Every instance had many equilibria, most of which only involved a few candidates.

These two-candidate equilibria have some interesting properties. Because they can include any two candidates where one does not Pareto-dominate the other, they can exist even when a third candidate Pareto-dominates both. Thus, it is possible for two-candidate equilibria to fail to elect a Condorcet winner. However, because every two-candidate equilibrium is effectively a pairwise runoff, it is impossible for a two-candidate equilibrium to elect a Condorcet loser.

Equilibria supporting three or more candidates are less straightforward. Which 3-candidate combinations are possible in equilibrium (even without ε -truthful incentives) can depend on the specific type distribution and the agents' particular utilities. Also, in these equilibria, agents do not always vote for their most preferred of the three alternatives (again, depending on relative probabilities and utilities). Finally, 3-candidate equilibria can elect a Condorcet loser with non-zero probability.

6.6 Discussion and Future Work

Our work shows that computational mechanism analysis can be applicable to mechanisms other than position auctions. Using the AGG representation and the AGG-SEM algorithm, we were able to explore the entire space of Nash equilibria for games where conventional representations and algorithms would be far too costly to use. As was the case with position auctions, AGG-SEM was particularly valuable because of its ability to enumerate all PSNEs of a game. The issue of multiple equilibria arose even with the added assumption of the truthfulness incentive; further, it is only by enumeration that we were able to measure exactly how many equilibria a given game had with and without the truthfulness incentive.

We saw several interesting results, beyond a reduction in the number of equilibria due to our truthfulness incentive. One of the most significant was the “clustering” of many equilibria around a subset of candidates, that reflect the voters aggregated preferences (e.g., Condorcet winners). A very large share of each game’s equilibria resulted in winners that were either truthful winners (according to plurality) or Condorcet winners. Truthful winners were selected in a larger fraction of equilibria when the total number of equilibria was fairly small (as was the case in a large majority of our experiments), and their share decreased as the number of equilibria increased (where we saw, in cases where there were Condorcet winners, that those equilibria took a fairly large share of the total). Furthermore, these results held up even when varying the number of candidates and voters, and many of them appear to also hold with other voting systems, such as veto and k -approval.

Looking at social welfare enabled us to compare equilibrium outcomes to all other possible outcomes. We observed that plurality achieved nearly the best social welfare possible (a result that did not rely on our truthfulness incentive). While another metric showed the same “clustering” we noted above, most equilibrium results concentrated around candidates that were ranked, on average, very highly (on average, more than 50% of winners in every experiment had a rank less than 1). This suggests that one should question whether it is even important to minimize the amount of manipulation, as we found that manipulation by all voters very often results in socially beneficial results.

In the Bayes-Nash results, we saw that lack of information often pushed equilibria to be a “battle” between a subset of the candidates—usually two candidates (as Duverger’s law would indicate), but occasionally more.

There is much more work to be done in the vein we introduced in this chapter. This includes examining the effects of changing utility functions, as well as looking at more voting rules and determining properties of their equilibria. Voting rules can be ranked according to their level of clustering, how good, socially, their truthful results are, and similar criteria. Furthermore, it would be worthwhile to examine other distributions of preferences and preference rules, such as single-peaked preferences.

Chapter 7

The Positronic Economist

7.1 Introduction and Motivation

The work of the previous chapters clearly demonstrates that computational mechanism analysis is technologically feasible, and can produce novel, scientifically interesting results. The major thread that connects this body of work, and differentiates it from other work that uses computational methods to analyze mechanisms, is that the equilibrium computation is done by state-of-the-art general-purpose equilibrium-finding algorithms.

This chapter focuses on laying out a way forward for our approach to computational mechanism analysis, codifying some of the desiderata that motivated our past design decisions, and ultimately providing a general, easy-to-use system that other researchers can build on to answer their own mechanism analysis questions. We call this system “the Positronic Economist.”

Our desiderata for a computational-mechanism-analysis system are spelled out in our “Three Laws of Positronic Economics,” inspired by Asimov’s [1942] “Three Laws of Robotics.” (See Figure 7.1.)

In order for a CMA system to address real open questions, fidelity is key: any imprecision in either the game representation or the equilibrium could compromise the relevance of the system’s findings. While approximate equilibria are a major subject of research in algorithms and complexity, researchers in mechanism design and auction theory overwhelmingly favor exact equilibria.

Our second desideratum is speed, with a strong emphasis on empirical performance over worst-case asymptotic guarantees. While polynomial-time guarantees are strongly associated with good empirical performance, they do not always provide a complete picture, e.g., see the discussion of different IRSDS variants in Chapter 3. For our game representation of choice, Bayesian action-graph games (BAGGs), Nash equilibrium finding is an NP- or PPAD-hard problem, depending on the equilibrium type, but good heuristics can often

1. **Precision:** Games are represented exactly. Equilibria are either exact, or are ε -equilibria where ε is on the order of machine- ε .^a
2. **Speed:** Algorithms must be fast enough to be used in practice, except when this is in conflict with the first law. Algorithms must provably require only polynomial time and space, except when this
 - is in conflict with the first law,
 - cannot be done without answering a major open complexity-theory problem, such as $P == NP$.
3. **Autonomy:** Human effort should be minimized, except when this is in conflict with the first two laws.

^aFor all the analyses in preceding chapters, $\varepsilon = 10^{-10}$. Since the games typically involve payoffs that can be 1 or more, this represents a multiplicative error of one part per at least 10^{10} . In contrast, IEEE 32-bit floating point numbers have 22 significand bits, thus have a multiplicative error of up to one part per $2^{22} \simeq 4.2 \times 10^7$. Rational numbers represented as a ratio of signed 32-bit integers have a multiplicative error $2^{31} \simeq 2.2 \times 10^9$.

Figure 7.1: The Three Laws of Positronic Economics

find exact equilibria quickly (see Chapter 3 and Jiang et al. [51]).

Our last desideratum for a CMA system is that it be autonomous, requiring minimal human effort and ingenuity. While the previous two desiderata are substantially addressed by previous work, this one is not, and thus receives the most attention in this chapter. In our pipeline (Figure 1.1), analysis is broken down into a two-stage process: encoding followed by solving. Due to our use of BAGGs, solving is straightforward—we rely on off-the-shelf Nash equilibrium finding algorithms. On the other hand, until now, encoding has required ingenuity and effort, first for devising a suitable graphical-model representation of a game and then for working with lower-level APIs or file specifications to produce a BAGG encoding for use with an existing solver.

The main aim of the Positronic Economist system is to produce compact BAGG representations of economically relevant settings while preserving the main advantages of such BAGG representations: fidelity and speed.

This chapter makes two major contributions. The first is the Positronic Economist API (PosEc), which is a high-level python-based declarative language for describing games in a form that resembles their natural mathematical representation as closely as possible. The second is a pair of complementary algorithms that automatically infer the structure of a game specified in PosEc and produce a compact BAGG.

The structure of this chapter is as follows. First, we survey related work. Next, we describe the PosEc representation language. We go on to describe projection, an extension of our mathematical model that can be used to describe some common forms of game structure. We then describe our structure inference algorithms and prove some properties about their performance. We move on to experimental results, demonstrating that

our algorithms can work in practice to quickly produce compact games. Finally, we discuss some directions for future work.

7.1.1 Related work

There is a small body of work on the use of equilibrium computation for analysis of mechanisms, but none of it interfaces with the compact games literature in the same way as our work. Nevertheless, we now reconsider the three other general-purpose CMA systems of which we are aware, assessing how well they satisfy our desiderata. In brief: each of the systems we discuss can represent games that cannot be represented by the other systems, including ours. However, our approach is best able to leverage high-performance equilibrium-finding algorithms.

The first system, due to Rabinovich et al. [87], does not explicitly specify how games are to be represented. Instead, a user has to provide code for computing expected utility given a strategy profile. This could require substantial human effort; observe that one of the main benefits of action-graph games is that they provide such an efficient computational procedure. This also gives rise to some risk of lost fidelity; for example, in their application to simultaneous auctions, Rabinovich et al. were forced to approximate the tie-breaking rule. Further, their system only supports the fictitious play algorithm (FP), which is a relatively weak Nash equilibrium computation algorithm, being prone to getting stuck in cycles. Thus, FP can consume extreme amounts of time without ever finding a high-fidelity (i.e., small ϵ) equilibrium.

Second, the system of Vorobeychik et al. [108] represents mechanisms and settings as piecewise linear equations. Given that many single-parameter mechanisms and settings are described algebraically in the literature, this representation requires very little human effort and has great fidelity. However, the only equilibrium-finding algorithm available for this representation is iterative best response, which is a degenerate case of fictitious play and converges even less often.

The third system, called empirical algorithmic game theory (EAGT), developed by Wellman and others [52, 66, etc.], involves writing a software simulator of a given game. At the simulator stage, perfect fidelity is possible, and human effort is mitigated since the user can build the simulator in whatever language is most convenient. However, these simulator-based games are then reduced to normal-form games by sampling from the simulator based on a representative set of strategies. Devising these strategies requires substantial user effort; it also risks loss of fidelity. The method does offer the advantage that high-fidelity equilibrium-finding algorithms can be used once a normal-form game representation is obtained.

7.2 Representing Games in PosEc

The Positronic Economist system consists of two key parts: a declarative language for specifying utility functions and mechanisms, and a structure inference system for automatically constructing compact Bayesian Action Graph Games (BAGGs)¹ for settings described in this language. We describe both of these elements in turn, beginning with the language. Our goal is to follow the natural mathematical formalization of a mechanism as closely as possible.

7.2.1 Mechanisms and Settings

Recall the definitions of mechanisms and settings, from Chapter 2.

An *epistemic-type Bayesian game* is specified by $\langle N, A, \Theta, p, \mathcal{U} \rangle$, where

- N is a set of agents, numbered 1 to n ,
- $A = A_1 \times A_2 \times \cdots \times A_n$ and A_i is a set of actions that agent i can perform,
- Θ is the a set of private types that an agent can have,
- p is the joint type distribution, $p \in \Delta\Theta^n$, and
- \mathcal{U} is a profile of n utility functions where $\mathcal{U}_i : A \times \Theta^n \rightarrow \mathbb{R}$.

This paper considers “mechanism-based games,” and so splits games into two parts, a mechanism and a setting. A *mechanism* is given by $\langle A, M \rangle$ where

- $A = A_1 \times A_2 \times \cdots \times A_n$ and A_i is a set of actions that agent i can perform, and
- M is the choice function, $M : A \rightarrow \Delta O$.

A *Bayesian setting* is given by $\langle N, O, \Theta, p, u \rangle$ where

- N is a set of agents, numbered 1 to n ,
- O is a set of outcomes,
- Θ is the a set of private types that an agent can have,
- p is the joint type distribution, $p \in \Delta\Theta^n$,² and

¹

²We will assume that p can be factored into independent p_i 's where $p(\theta) = \prod_1^n p_i(\theta_i)$. This assumption is not without loss of generality, but is present in the current reference implementation of BAGGs. Thus, any setting that is not consistent with this assumption cannot be represented by the Positronic Economist.

- u is a utility function $u : N \times \Theta^n \times O \rightarrow \mathbb{R}$.³

Perfect information settings are a special case of Bayesian settings where each agent's type distribution is a point mass. Any mechanism and setting that both use the same n and O can be combined to form a game where $\mathcal{U}_i(\theta_N, a_N) = u(i, \theta_N, M(a_N))$ and where $a_N \in A$ denotes an action profile.

7.2.2 The PosEc Modeling Language

PosEc is a language that aims to make it easy for users to describe mechanisms and settings. This section provides examples and discussion of some of the key decisions that went into its design. The appendices go much further, giving an introduction to the API as well as exhaustive documentation of the API's specifications.

Our modeling language is based on python. Thus, the tuples, sets and utility functions of the mathematical representation become tuples, set and functions in python. To specify a mechanism-based game, a user must define a choice function `M` and a `setting`. For example, here is the choice function for a plurality vote with deterministic tie-breaking:

```
def M(setting, a_N):
    for c1 in setting.O:
        c1Wins = True
        for c2 in setting.O:
            if a_N.count(c2) > a_N.count(c1):
                c1Wins = False
        if c1Wins: return c1
```

A corresponding setting follows, where the elements of `Theta` represent descending preference orderings.

```
n=10
O = ("c1", "c2", "c3")
Theta = [("c1", "c2", "c3"),
          ("c2", "c3", "c1"),
          ("c3", "c1", "c2"),
          ("c1", "c3", "c2")]
P = [UniformDistribution(Theta)]*n
def u(i, theta, o, a_i):
    return theta[i].index(o)
setting = Setting(n, O, Theta, P, u)
```

Once the user has defined the mechanism and setting in this way, the Positronic Economist system creates

³Note that it is equivalent to say either

1. u is a profile of n utility functions $u_i : \Theta \times O \rightarrow \mathbb{R}$; or
2. u is a utility function that takes an agent number as one of its parameters $u : N \times \Theta \times O \rightarrow \mathbb{R}$.

While form (1) is more common, form (2) is dramatically easier for a novice Python user to produce correctly. Thus, we prefer form (2).

a BAGG of that game (described in the next section), allowing it to be analyzed using a variety of high-performance algorithms.

7.2.3 Special Functions in PosEc

One of our goals was to let the user implement their utility and choice functions however they liked. Indeed, PosEc will convert any valid Python functions into a valid BAGG.

However, the way that the functions are implemented can affect the size of the BAGG and the amount of time required to produce it. In this section, we outline the key PosEc constructs that can be used to signal game structure.

First, consider an anonymous mechanism with a constant number of actions c , such as our voting game. The corresponding normal-form game requires $O(nc^n)$ space, while the corresponding AGG is $O(n^c)$. We can signal that the mechanism is anonymous by using the `a.N.count()` function, as in the plurality-vote code above.

Second, many important settings involve exponentially large outcome spaces (consider e.g., stable matching settings such as kidney exchange), which require exponential space when represented using standard Python classes. The situation is even worse if the natural specification of a setting involves an infinite outcome space (e.g., in much of the auction literature valuations and payments can be arbitrary real values) as there are no standard Python classes for infinite sets.⁴ Both problems have the same solution: introducing special-purpose set-like classes. The PosEc system includes several such set-like classes, allowing for outcome spaces that include permutation, for use in position auctions, real values, and partition matroids, as well as their Cartesian products.

Third, randomized mechanisms also need to be handled carefully, in order to avoid violating the first law. In particular, if a choice function performed its own randomizing, e.g., via the Python `random` module, PosEc would need to sample the choice function in order even to approximate the distribution over outcomes. Instead, mechanisms should be implemented with the choice function returning a *distribution* over outcomes, so that PosEc can compute the expected utilities as accurately as possible.

The overriding goal for the PosEc API is to allow the user to precisely specify games as easily as possible. Thus, our design decisions emphasize an extremely simple general language for the user to build with, rather than a palette of options for the user to choose from. For example, there is no special keyword to specify that an agent has quasilinear utility with linear value for money. Instead, we make it easy for the user to specify such a utility function directly:

⁴Of course, PosEc only generates finite-action games. Nevertheless, PosEc encourages the specification of infinite outcome spaces when it is natural to do so; this allows the user to vary the discretization imposed by a mechanism while holding the setting constant.

```

def u(i, theta, o, a_i):
    alloc, payments = o
    if alloc==i:
        return theta[i]-payments[i]
    return -payments[i]

```

Naturally, we hope that users will produce a library of reusable mechanisms and settings. We provide code for position auctions and voting.

7.2.4 Projection

In many single-good auction settings (including settings with the utility function shown in the previous section), each agent’s payoff depends only on whether he is allocated the good and on his own payment. Such structure is computationally useful: an agent’s utility may be computed without deriving a distribution over the entire outcome space. We can generalize this concept via the idea of *projection*.

Formally, a *projected setting* is $\langle N, O, \Theta, \Psi, u, \pi \rangle$ where N, O and Θ are defined as before. Ψ is the space of projected outcomes, u is overloaded where $u : N \times \Theta^n \times \Psi \rightarrow \mathbb{R}$, and π is a projection function $\pi : N \times O \rightarrow \Psi$ (where for any $o, o' \in O$, $\pi(i, o) = \pi(i, o')$ implies $u(i, \theta_N, o) = u(i, \theta_N, o')$). Projected settings do not necessarily lead to computational savings, but are useful because they make it possible to define mechanisms that only compute projected outcomes for a single agent at a time. Formally, a *projected mechanism* is specified by $\langle A, M' \rangle$ where A is again a set of action profiles, and M is a choice function over (distributions over) projected outcomes; $M' : N \times A \rightarrow \Delta\Psi$ (where $M'(i, a_N) = \pi(i, M(a_N))$). Any projected setting and projected mechanism that both use the same n and Ψ can be combined to form a Bayesian game where $\mathcal{U}_i(\theta_N, a_N) = u(i, \theta_N, M'(a_N))$ for each action profile $a_N \in A$. Such Bayesian games can be much more compact than Bayesian games based on the equivalent (unprojected) settings and mechanisms. PosEc’s black-box structure inference algorithm can discover these more compact BAGGs by itself; however, this can take significant time: e.g., $\Omega(2^n)$ computation for digital good settings and $\Omega(n!)$ for position auction settings. Thus, while expressing projection structure can be more work for the user, doing so can yield exponentially faster computation.

7.3 Structure Inference in PosEc

The second main component of PosEc is structure inference: automatically generating compact BAGGs given setting and mechanism descriptions in PosEc’s own modeling language. We provide two approaches for doing this. First, “white-box structure inference” uses structure made explicit in the PosEc representation—e.g., via use of the count operator, randomization via distributions, projection, etc.—to generate a BAGG. (In the degenerate case, no such structure is explicitly given, and we obtain an exponential-size BAGG.) Second,

Input: Bayesian game, utility represented as a function

Output: Bayesian action-graph game

Create a BAGG with actions agents N , actions A , types Θ , type distribution p , and no edges or function nodes

foreach Action-node a **do**

 create payoff table for action node a

repeat

$f \leftarrow 0$

foreach projected configuration c on neighbors of a **do**

 try to compute payoff given c

if success **then** add c and payoff to table

else

$v \leftarrow$ a missing variable needed to compute payoff

if no function node computes v **then**

 add function node and edges to compute v

 add edge from function node to a

 erase contents of payoff table

$f \leftarrow 1$

break

until $f=0$;

Figure 7.2: White-Box Structure Inference

“black-box structure inference” takes the BAGG generated in the first step and probes it to find additional structure and hence to obtain a more compact BAGG. This procedure is completely automatic, but (1) depends on the size of the initial representation, and (2) relies upon heuristics to avoid getting stuck in local minima. Thus, black-box structure inference should be seen as a procedure for further refining the compact BAGGs obtained via white-box structure inference, not as a replacement for white-box structure inference.

7.3.1 White-Box Structure Inference

We aim to obtain what we call the “straightforward BAGG”: a BAGG that contains only those function nodes and edges that are necessary to compute features used by the input game. Let ℓ_s denote the representation length of this game. Our goal is to compute the straightforward BAGG using only $\text{poly}(\ell_s)$ calls to the utility function of the input game. We do so via a relatively direct algorithm. Essentially, it works by beginning with a totally disconnected action graph, and progressively adding function nodes and edges whenever their absence means that the utility function cannot compute a payoff. (See Algorithm 7.2.)

Theorem 15. *For any game parameterized by the number of agents n , the number of actions per agent m , and the number of types t , where the choice and utility functions each can make a constant number of different calls to accessor functions, and where any weighted-sum calls involve weights bounded by $\text{poly}(nmt)$, the straightforward BAGG will require only $\text{poly}(nmt)$ space.*

Proof sketch. WBSI introduces at most one function node per accessor call. A weighted max node can have at most $O(nmt)$ different projected configurations. A weighted sum node with maximum w can only have at most $O(wn)$ different projected configurations. For each action node, there are a constant number of neighbors, all of which are function nodes. Thus the possible projected configurations on the neighborhood of any action node is at most the Cartesian product of the possible projected configurations of each neighbor. Since these spaces are all $\text{poly}(nmt)$ and there are boundedly many of them, the total configuration space in the neighborhood of every action is at most $\text{poly}(nmt)$. \square

Small outputs are important because the BAGGs produced by WBSI are typically used as inputs to game-solving algorithms, which often require worst-case exponential time. Nevertheless, these algorithms often have good empirical performance. (See e.g., Chapter 3 and Jiang et al. [51]). Thus, it is important that WBSI also be fast, so as not to become the main bottleneck.

Theorem 16. *The white-box structure-inference algorithm (Algorithm 7.2) runs in $O(c(\ell_s)^2)$ time, where ℓ_s denotes the size of its output, the straightforward BAGG, and c denotes the amount of time that the input code requires to compute a single agent’s payoff for a single type-action-profile.*

Proof sketch. The runtime is dominated by the computation of payoff tables. The outer for loop and repeat loop jointly take $O(\ell_s)$ time: the for loop runs once per action node, and each iteration of the repeat loop after the first one involves creating a new edge, and both actions and edges take up space in the BAGG representation. The inner for loop iterates over projected configurations, where one payoff per projected configuration is also part of the BAGG representation. Because this loop only deals with BAGGs that contain weakly fewer edges than the straightforward BAGG, it can only iterate over projected-configuration spaces that are weakly smaller than the projected configuration space of the straightforward AGG. Thus, this inner loop also takes $O(\ell_s)$ time. \square

Provided the outcome and utility functions passed to WBSI are polynomial time, and make bounded numbers of calls to accessor functions, WBSI will run in polynomial time and produce a polynomial-sized BAGG as output.

7.3.2 Black-Box Structure Inference

The goal of black-box structure inference is to take a BAGG obtained from white-box structure inference—in the degenerate case, a completely unstructured BAGG—and return another BAGG that more efficiently represents the same game. In other words, BBSI is a constrained optimization problem where the feasible region is the set of all BAGGs that are equivalent to the input game and the objective is to find the smallest BAGG, measured by input length, in the feasible region. We can also define a decision version of BBSI: BAGGREducIBILITY takes a BAGG G and an integer k , and asks whether there exists a BAGG G' that is strategically equivalent to G and that has representation size no larger than k . We conjecture that BAGGREducIBILITY is NP-hard.

Our focus in this paper is thus on heuristics for performing BBSI. There are some natural local operations that can produce a smaller, strategically equivalent BAGG. A simple, polynomial-time example is to try cutting an edge to an action node, reducing by one the dimension of that action node’s payoff table. (E.g., in a GFP auction, an agent’s payoff is unaffected by each bid less than his own.) Another example is aggregating two or more actions, replacing their inputs with a single sum node. (E.g., in a no-externality GFP auction, an agent’s payoff only depends on the number of higher bids, not these bids’ values.) Unfortunately, such local operations can get stuck in local optima. Consider, for example, a five-candidate vote using the two-approval mechanism where voters have the option to abstain. This setting can be encoded as summing the number of approvals each candidate gets using five sum nodes, for a $O(n^5)$ representation. However, a naive anonymous encoding could count how many agents approved each pair of candidates, for a $O(n^{10})$ representation. It is not possible to get from the naive encoding to the efficient encoding using the two operations defined above; none of the counts can be cut and no pair of counts can be replaced with their sum.

Given that locally improving moves can get stuck in local optima, we opted to implement BBSI using an iterative local-search (ILS) algorithm that sometimes makes non-improving moves. Our ILS algorithm optimizes the size of the payoff table for each action node independently. For each action node, the algorithm starts from the initial neighborhood of that node (in order to preserve information already encoded in the BAGG by WBSI), and alternates between non-improving perturbations (introducing function nodes that aggregate existing inputs, even when those inputs cannot subsequently be cut) and locally improving moves (cutting edges to an action node when doing so maintains strategic equivalence).

Our BBSI-ILS algorithm is a heuristic search with several free parameters, which were manually configured to exploit the kinds of structure that are commonly present in mechanism-based games. Initially, the focus is on finding and exploiting anonymity structure, so the mechanism introduces SUM nodes to compute action counts and tries to cut arcs that provide asymmetric information. Once this phase is complete, the algorithm searches more broadly. The perturbation phase introduces weighted SUMs constructed by adding together existing weighted SUM nodes, biased in favor of not introducing weights greater than 1. The improvement phase cuts any arc, not just those that provide asymmetric information.

7.4 Experimental Results

In this section, we describe experimental evidence that our structure inference algorithms can produce compact BAGGs in a practical amount of time.

We began by recreating games from Chapter 4 and Chapter 6—specifically, GFP and wGSP position auctions, and two-approval votes—which were originally produced using hand-tuned encoders. To test WBSI, we generated straightforward specifications of these settings in PosEc. In all cases, WBSI produced exactly the same output as the hand-tuned encoders, which are dramatically smaller than the corresponding normal-form

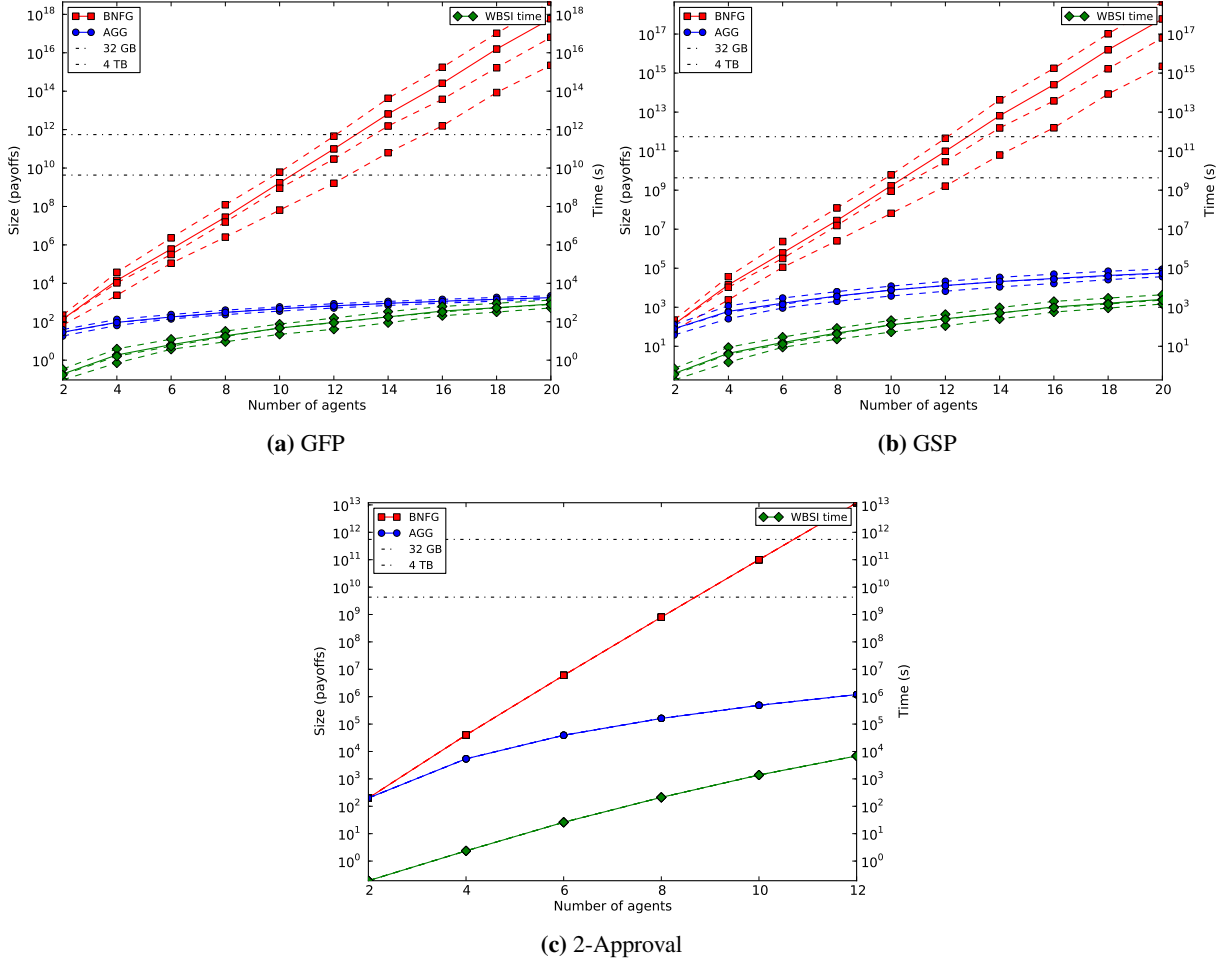


Figure 7.3: Measuring the performance of WBSI on efficiently-represented games. Solid lines correspond to mean size and runtime while the dashed lines correspond to the 5th percentile, median and 95th percentile of 50 runs.

games. Runtime performance was also good: even extremely large games involving 20 players were produced in less than an hour, except in the case of two-approval. (The largest two-approval games we created had 12 players. Even so, these games took roughly three hours each to produce. See Figure 7.3.)

To validate black-box structure-inference, we tested the same games, both with and without well-implemented inputs. When the inputs were well implemented (i.e., using appropriate accessor functions), BBSI did not make substantial, further reductions to representation size, suggesting that the solution obtained by WBSI was already nearly optimal. In contrast, BBSI was able to make dramatic improvements to the size of poorly implemented inputs, shrinking games from their exponential normal-form size to within less than twice the size of hand-tuned encodings. However, the runtime of BBSI grew with the size of the normal form: exponentially with n . Thus, BBSI could take as much as an hour even for small games ($n = 4$). (See

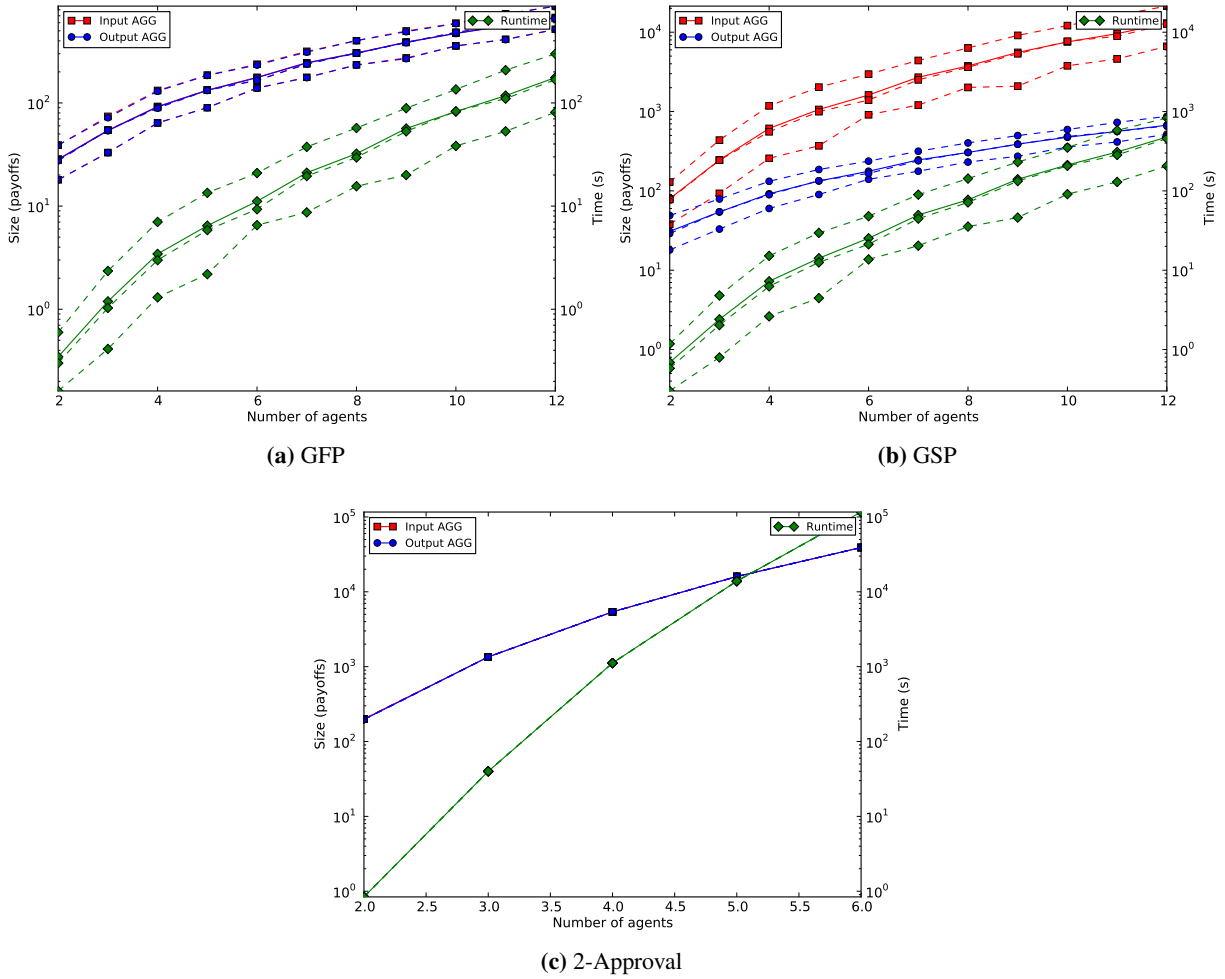


Figure 7.4: Measuring the performance of the BBSI-ILS algorithm on tuned inputs. Because these inputs were already somewhat efficient, the compression is mostly small, except in the case of GSP. Runtime was manageable for position auction games, but prohibitively expensive for larger voting games.

Figures 7.4 and 7.5.)

We have so far motivated BBSI as a refinement of WBSI for computational reasons. However, it has another use: as a sort of automated tutorial to help a user make better use of the PosEc system. Although BBSI cannot practically be applied to large, unstructured games, it can indirectly help in such cases by identifying more efficient representations of smaller, related games. A user can then examine this representation to learn what sort of structure is important, and write a PosEc description of the setting that makes this structure explicit.

This tutorial mode works by taking a PosEc-produced BAGG and explaining, for each action node, which accessor function calls are used to produce the payoff table. By design, our BBSI implementation only makes

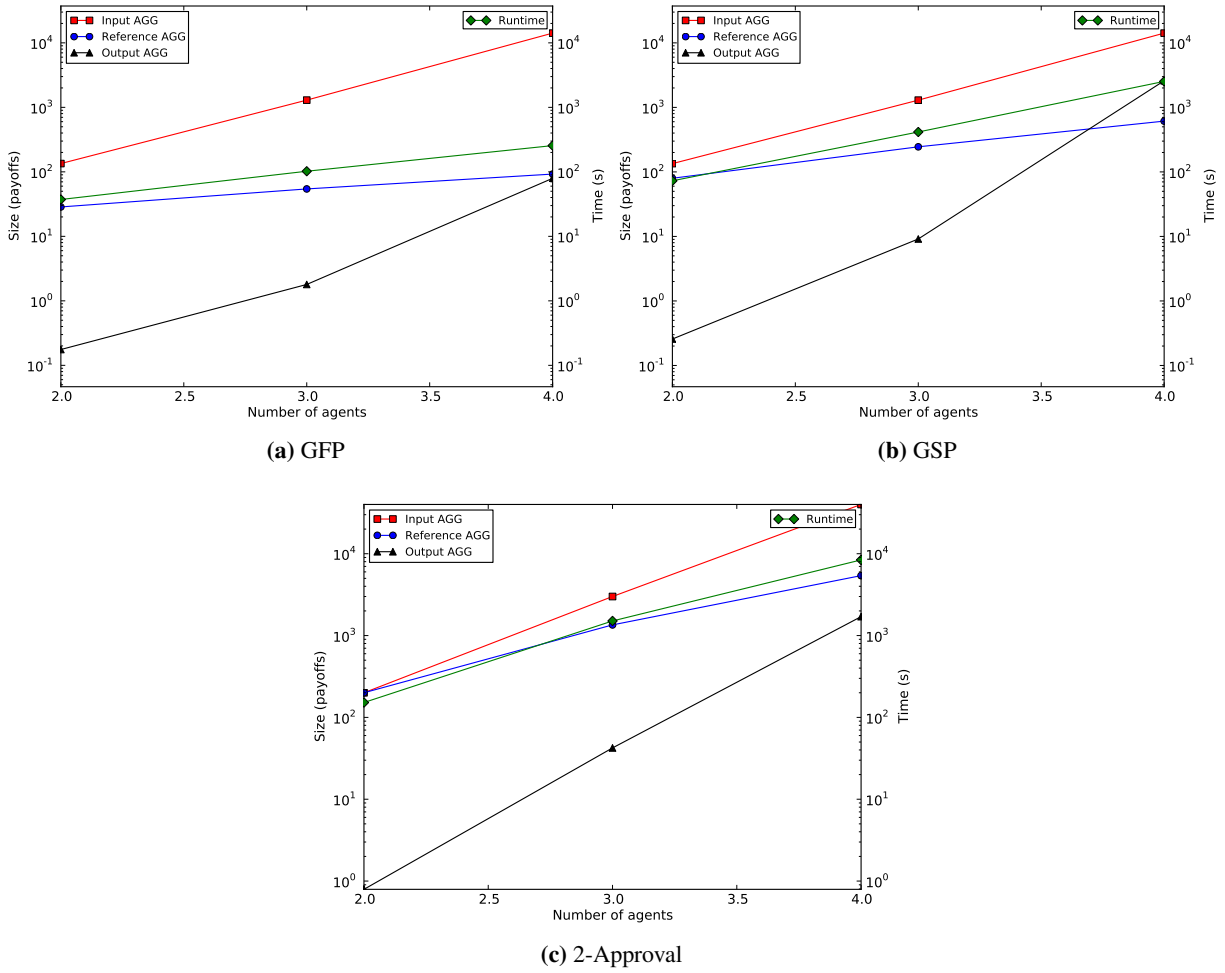


Figure 7.5: Measuring the performance of the BBSI-ILS algorithm on untuned inputs. BBSI was able to produce a dramatic improvement in size for every input, though it was seldom as good as the reference games produced by WBSI with a tuned input.

moves that are consistent with PosEc’s accessors. Specifically, every cut reduces the number of accessor calls and every perturbation introduces a function node that corresponds to a single accessor call. Thus, the output of BBSI is always explainable terms of PosEc accessors. Because our aim here is to help a user represent a class of games in a way that scales well in the number of agents, the appropriate measure is not the size of the output BAGG, but rather the asymptotic size of the family of BAGGs with the same structure. To measure this, we calculate an upper bound on the size of each payoff table, as a function of n , based on the accessor functions. For example, a call to `count()` will return an integer between zero and $n - 1$ —or 1 and n if counting how many agents played the action corresponding to that payoff table entry, while a call to `any()` has two possible values. Thus, an action node that is explained by these two calls has a payoff table with at most $2n$ entries. From this expression, it is possible to compute a bound on the size of the BAGG, as a function of n .

For each of the games from the previous subsection, we computed these bounds, as a measure of the efficiency of the structure discovered by BAGGs. We compared this measure against two benchmarks: the size of the normal form and the size of the efficient representation produced by carefully hand-tuned inputs to WBSI. In our experiments, extrapolating from the structure that BBSI identified in games with three agents, we found that extrapolated games were dramatically larger than hand-tuned games produced by WBSI, but far smaller than normal-form games. (See Figure 7.6.) Thus, we found that BBSI was capable of producing output that is not only more compact than unstructured games, but also informative to human users.

7.5 Conclusion and Future Work

This chapter makes two major technical contributions: the Positronic Economist declarative language for describing mechanism-based Bayesian games, and the two structure-inference algorithms that make it possible to compactly represent such games as Bayesian action-graph games. These contributions dramatically reduce the human effort necessary to perform computational mechanism analysis, without leading to a substantial loss of accuracy or speed. There are many potential applications in which PosEc could shed light on hard-to-analyze economic situations. Even within the limited and well-studied sphere of single-good auctions, PosEc could be used to study non-linear utility for money (e.g., budgets; risk attitudes), asymmetric valuation distributions, other-regarding preferences (both altruism and spite), and conditional type dependence (including common and affiliated values).

One limitation of the current PosEc system is that it can only describe simultaneous-move games. In contrast, many real-world mechanisms proceed in multiple stages (e.g., sequential auctions; clock-proxy auctions). In both cases, decisions made in one stage can affect which outcomes are possible or desirable in the next stage. BAGGs are inherently single-stage games, but could be used to analyze multi-stage mechanisms by representing the different stages as individual BAGGs and solving the complete system by a process of backward induction. Such a process would likely resemble the special-purpose algorithm that Paes Leme et al. [82] proposed for computing the equilibria of sequences of single-good auctions. Alternatively, CMA could be performed using a compact game representation that explicitly supports multi-stage games, such as temporal AGGs [50] or MAIDs [57]. Unfortunately, the algorithms for reasoning about such representations currently offer much poorer performance than algorithms for reasoning about BAGGs.

Another limitation of the PosEc system is the cost of explicitly representing types and actions; no BAGG can be asymptotically smaller than its number of types or actions. Thus, games with large type or action spaces—such as combinatorial auctions—cannot be succinctly represented. There is very little work on representing games with implicitly specified action spaces—Koller and Milch [57] and Ryan et al. [93] are the two exceptions, unfortunately both without good implementations—but as this literature develops there may be opportunities for extending our encode-and-solve CMA approach to new game families.

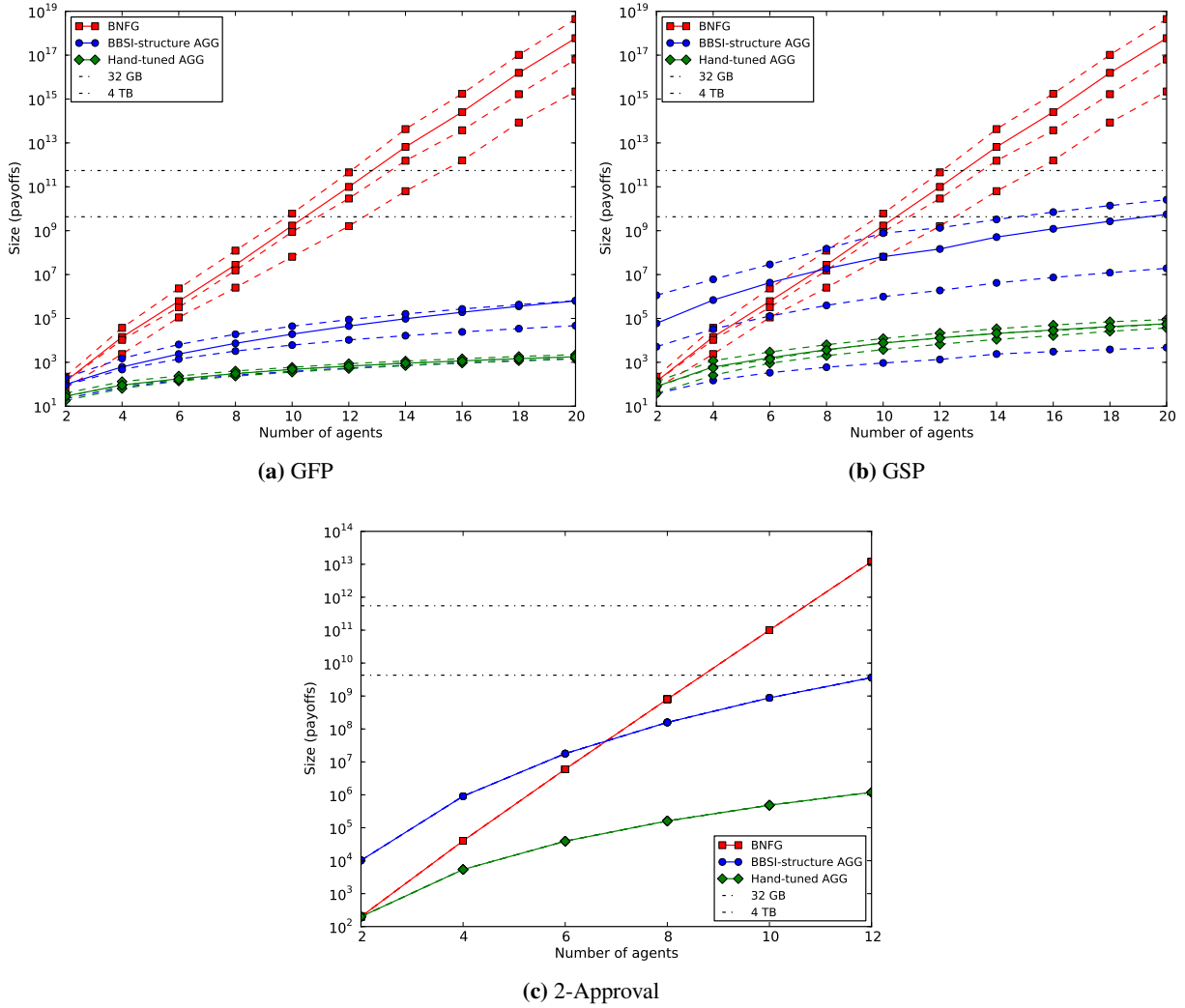


Figure 7.6: Measuring the scales of BAGGs using structure learned by BBSI. Note that the extrapolated BBSI output is a loose upper bound; for example, no (B)AGG representation can ever include more payoff table entries than the strategically (B)NFG. Solid lines correspond to mean size and runtime while the dashed lines correspond to the 5th percentile, median and 95th percentile of 50 runs. In position-auction games, we only included actions up to each bidder’s valuation. Thus, the size (in terms of number of actions) could vary from game to game. This is the cause of the much higher variance on position-auction games, compared to voting games.

Bibliography

- [1] G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. In *EC*, 2006. → pages 94, 97, 105
- [2] G. Aggarwal, J. Feldman, S. Muthukrishnan, and M. Pal. Sponsored search auctions with Markovian users. In *Workshop on Ad Auctions*, 2008. → pages 37
- [3] I. Asimov. Runaround. In *Astounding Science Fiction*, 1942. → pages 129
- [4] S. Athey and G. Ellison. Position auctions with consumer search. *Quarterly Journal of Economics*, 126:1213–1270, 2011. → pages 34
- [5] S. Athey and D. Nekipelov. A structural model of sponsored search advertising auctions. Working paper. → pages 9
- [6] L. Babai. Monte-carlo algorithms in graph isomorphism testing. Technical report, Universite de Montreal, 1979. → pages 9
- [7] J. J. Bartholdi III and J. B. Orlin. Single transferable vote resists strategic voting. *Social Choice & Welfare*, 8:341–354, 1991. → pages 113
- [8] J. J. Bartholdi III, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice & Welfare*, 6(3):227–241, 1989. → pages 113
- [9] M. Benisch, N. Sadeh, and T. Sandholm. The cost of inexpressiveness in advertisement auctions. In *Workshop on Ad Auctions*, 2008. → pages 36, 56, 74, 98
- [10] K. Bhawalkar and T. Roughgarden. Welfare guarantees for combinatorial auctions with item bidding. 2011. → pages 2
- [11] L. Blumrosen, J. D. Hartline, and S. Nong. Position auctions and non-uniform conversion rates. In *Workshop on Ad Auctions*, 2008. → pages 36, 53, 98
- [12] I. Caragiannis, C. Kaklamanis, P. Kanellopoulos, M. Kyropoulou, B. Lucier, R. Paes Leme, and E. Tardos. On the efficiency of equilibria in generalized second price auctions. Invited to the Journal of Economic Theory (JET). → pages 35, 102
- [13] S. Chawla and J. D. Hartline. Auctions with unique equilibria. In *EC*, 2013. → pages 51
- [14] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS*, 2006. → pages 8

- [15] M. R. Chernick. *Bootstrap Methods, A practitioner's guide*. Wiley, 1999. → pages 22, 45
- [16] V. Conitzer and T. Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *ACM-EC*, 2004. → pages 1
- [17] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990. → pages 3
- [18] P. Cramton. *Simultaneous ascending auctions*. MIT Press, 2006. → pages 2
- [19] C. Daskalakis, G. Schoenebeck, G. Valiant, and P. Valiant. On the complexity of Nash equilibria of action-graph games. In *SODA*, 2009. → pages 8, 18, 118
- [20] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39:195–259, 2009. → pages 8
- [21] F. De Sinopoli. On the generic finiteness of equilibrium outcomes in plurality games. *Games and Economic Behavior*, 34(2):270–286, February 2001. → pages 115
- [22] Y. Desmedt and E. Elkind. Equilibria of plurality voting with abstentions. In *Proceedings of the 11th ACM conference on Electronic commerce (EC '10)*, pages 347–356, Cambridge, Massachusetts, June 2010. → pages 115
- [23] A. Dhillon and B. Lockwood. When are plurality rule voting games dominance-solvable? *Games and Economic Behavior*, 46(1):55–75, 2004. → pages 115
- [24] J. Dickhaut and T. Kaplan. A program for finding Nash equilibria. *Mathematica J.*, 1:87–93, 1991. → pages 12
- [25] S. Dughmi, T. Roughgarden, and M. Sundararajan. Revenue submodularity. In *ACM-EC*, 2009. → pages 97
- [26] B. Dutta and J.-F. Laslier. Costless honesty in voting. in 10th International Meeting of the Society for Social Choice and Welfare, Moscow, 2010. → pages 116, 119, 123
- [27] B. Dutta and A. Sen. Nash implementation with partially honest individuals. *Games and Economic Behavior*, 74(1):154–169, 2012. → pages 115
- [28] M. Duverger. *Political Parties: Their Organization and Activity in the Modern State*. Methuen Publishing, 1959. → pages 114
- [29] B. Edelman and M. Schwarz. Optimal auction design and equilibrium selection in sponsored search auctions. HBS Working Paper, 2010. → pages 105
- [30] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, March 2007. → pages 4, 5, 34, 35, 68, 96, 97, 105
- [31] B. Escoffier, J. Lang, and M. Öztürk. Single-peaked consistency and its complexity. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 366–370, Amsterdam, The Netherlands, 2008. → pages 113

- [32] T. Feddersen and W. Pesendorfer. Voting behavior and information aggregation in elections with private information. *Econometrica*, 65(5):1029–1058, September 1997. → pages 115
- [33] W. Gaertner. *A Primer in Social Choice Theory*. Oxford, 2006. → pages 1, 2
- [34] D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematics Monthly*, 69:9–14, 1962. → pages 1
- [35] E. Gerding, Z. Rabinovich, A. Bye, E. Elkind, and N. R. Jennings. Approximating mixed Nash equilibria using smooth fictitious play in simultaneous auctions. In *AAMAS*, 2008. → pages 10
- [36] A. Ghosh and M. Mahdian. Externalities in online-advertising. In *WWW: International World Wide Web Conference*, 2008. → pages 37, 98
- [37] A. Gibbard. Manipulation of voting schemes. *Econometrica*, 41(4):587–602, July 1973. → pages 113
- [38] A. Gibbard. Manipulation of schemes that mix voting with chance. *Econometrica*, 45(3):665–681, April 1977. → pages 113
- [39] I. Giotis and A. Karlin. On the equilibria and efficiency of the GSP mechanism in keyword auctions with externalities. In *WINE*, 2008. → pages 37, 58
- [40] R. Gomes, N. Immorlica, and E. Markakis. Externalities in keyword auctions: an empirical and theoretical assessment. In *WINE*, 2009. → pages 37
- [41] R. D. Gomes and K. Sweeney. Bayes-Nash equilibria of the generalized second-price auction. Working paper, 2011. → pages 97
- [42] G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: Hard and easy games. *JAIR*, 24: 357–406, 2005. → pages 8, 18
- [43] S. Govindan and R. Wilson. A global Newton method to compute Nash equilibria. *J. Economic Theory*, 110:65–86, 2003. → pages 9, 22, 44
- [44] J. D. Hartline and Q. Yan. Envy, truth, and profit. In *EC*, 2011. → pages 34
- [45] Inside AdWords. A common AdWords misconception explained..., Jan. 2006. <http://adwords.blogspot.ca/2006/01/common-adwords-misconception-explained.html>. → pages 96
- [46] A. X. Jiang and K. Leyton-Brown. Computing pure Nash equilibria in symmetric action-graph games. 2007. → pages 8
- [47] A. X. Jiang and K. Leyton-Brown. Bayesian action-graph games. In *NIPS*, 2010. → pages 2, 3, 8, 119
- [48] A. X. Jiang and K. Leyton-Brown. A general framework for computing optimal correlated equilibria in compact games. In *WINE*, 2011. → pages 4
- [49] A. X. Jiang and K. Leyton-Brown. Polynomial-time computation of exact correlated equilibrium in compact games. *Games and Economic Behavior*, (0):–, 2013. ISSN 0899-8256. doi:<http://dx.doi.org/10.1016/j.geb.2013.02.002>. → pages 3
- [50] A. X. Jiang, K. Leyton-Brown, and A. Pfeffer. Temporal action-graph games: A new representation for dynamic games. In *UAI*, 2009. → pages 142

- [51] A. X. Jiang, K. Leyton-Brown, and N. A. R. Bhat. Action-graph games. *GEB*, 71:141–173, 2011. → pages 2, 3, 6, 7, 8, 9, 11, 15, 17, 22, 44, 118, 130, 137
- [52] P. R. Jordan, Y. Vorobeychik, and M. P. Wellman. Searching for approximate equilibria in empirical games. In *AAMAS*, 2008. → pages 131
- [53] I. A. Kash and D. C. Parkes. Impersonation strategies in auctions. In *WINE*, 2010. → pages 105
- [54] K. Kawai and Y. Watanabe. Inferring strategic voting. Technical report, Northwestern, 2010. → pages 115
- [55] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI*, 2001. → pages 2, 6
- [56] D. Kempe and M. Mahdian. A cascade model for externalities in sponsored search. In *WINE*, 2008. → pages 37, 61, 70, 98
- [57] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *GEB*, 45: 181–221, 2003. → pages 2, 142
- [58] V. Krishna. *Auction Theory*. Academic Press, 2002. → pages 1, 2
- [59] S. Lahaie and P. McAfee. Efficient ranking in sponsored search. In *WINE*, 2011. → pages 97
- [60] S. Lahaie and D. M. Pennock. Revenue analysis of a family of ranking rules for keyword auctions. In *ACM-EC*, 2007. → pages 34, 36, 39, 42, 43, 49, 96, 97, 99, 100, 105, 106
- [61] J.-F. Laslier and J. W. Weibull. A strategy-proof condorcet jury theorem. forthcoming, *Scandinavian Journal of Economics*, 2012. → pages 115
- [62] O. Lev and J. S. Rosenschein. Convergence of iterative voting. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 2, pages 611–618, Valencia, Spain, June 2012. → pages 115
- [63] R. Lipton and E. Markakis. Nash equilibria via polynomial equations. In *LATIN*, volume 2976 of *LNCS*, pages 413–422, 2004. → pages 12
- [64] B. Lucier, R. Paes Leme, and E. Tardos. On revenue in the generalized second price auction. In *WWW*, 2012. → pages 97
- [65] O. Mangasarian. Equilibrium points of bimatrix games. *J. Society for Industrial and Applied Mathematics*, 12:778–780, 1964. → pages 12
- [66] B. A. Mayer, E. Sodomka, A. Greenwald, and M. P. Wellman. Accounting for price dependencies in simultaneous sealed-bid auctions. In *EC*, 2013. → pages 131
- [67] R. D. McKelvey and R. E. Wendell. Voting equilibria in multidimensional choice spaces. *Mathematics of Operations Research*, 1(2):144–158, May 1976. → pages 115
- [68] R. D. McKelvey, A. M. McLennan, and T. L. Turocy. Gambit: Software tools for game theory, 2006. <http://econweb.tamu.edu/gambit>. → pages 9, 22

- [69] R. Meir, M. Polukarov, J. S. Rosenschein, and N. R. Jennings. Convergence to equilibria of plurality voting. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, pages 823–828, Atlanta, July 2010. → pages 115
- [70] M. Messner and M. K. Polborn. Miscounts, Duverger’s law and Duverger’s hypothesis. Technical Report 380, IGIER (Innocenzo Gasparini Institute for Economic Research), Bocconi University, 2011. → pages 115
- [71] C. Metz. Yahoo! “handicaps” its search ad auctions. does google “squash” too?, 2010. URL http://www.theregister.co.uk/2010/09/16/yahoo_does_squashing/. → pages 96
- [72] P. S. Michael P. Wellman, Amy Greenwald. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition*. MIT Press, 2007. → pages 1
- [73] R. G. J. Miller. *Simultaneous Statistical Inference*. Springer, 1981. → pages 45
- [74] B. Murtagh and M. Saunders. MINOS, 2010. <http://www.sbsi-sol-optimize.com>. → pages 22
- [75] R. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1), 1981. → pages 2, 4, 97, 98
- [76] R. B. Myerson and R. J. Weber. A theory of voting equilibria. *The American Political Science Review*, 87(1):102–114, March 1993. → pages 115
- [77] R. B. Nelsen. *An Introduction to Copulas*. Springer, 2006. → pages 106
- [78] N. Nisan and A. Ronen. Algorithmic mechanism design. In *STOC*, 1999. → pages 1
- [79] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, Cambridge, UK, 2007. → pages 5, 8
- [80] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS*, pages 880–887, 2004. → pages 2, 12, 22
- [81] M. Ostrovsky and M. Schwarz. Reserve prices in internet advertising auctions: A field experiment. Working Paper, 2009. → pages 96, 97, 99, 109, 111, 112
- [82] R. Paes Leme, V. Syrgkanis, and E. Tardos. Sequential auctions and externalities. In *SODA*, 2011. → pages 142
- [83] D. L. Poole and A. K. Mackworth. *Artificial Intelligence*. Cambridge University Press, 2011. → pages 15
- [84] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. *GEB*, 63:642–662, 2008. → pages xi, 2, 9, 11, 12, 14, 24, 44, 118
- [85] A. D. Procaccia. Can approximation circumvent Gibbard-Satterthwaite? In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, pages 836–841, Atlanta, Georgia, July 2010. → pages 113

- [86] Z. Rabinovich, E. Gerding, M. Polukarov, and N. R. Jennings. Generalised fictitious play for a continuum of anonymous players. In *IJCAI*, 2009. → pages 10
- [87] Z. Rabinovich, V. Naroditskiy, E. H. Gerding, and N. R. Jennings. Computing pure bayesian nash equilibria in games with finite actions and continuous types. *Artificial Intelligence*, To appear:–, 2012. → pages 131
- [88] B. Roberts, D. Gunawardena, I. A. Kash, and P. Key. Ranking and tradeoffs in sponsored search auctions. In *EC*, 2013. URL <http://research.microsoft.com/pubs/193878/1304.7642v1.pdf>. → pages 96
- [89] R. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Int. J. Game Theory*, 2: 65–67, 1973. → pages 6
- [90] T. Roughgarden. *Selfish Routing*. PhD thesis, Cornell University, 2002. → pages 1
- [91] T. Roughgarden and C. Papadimitriou. Computing correlated equilibria in multi-player games. *JACM*, 37:49–56, 2008. → pages 2, 3
- [92] T. Roughgarden and E. Tardos. Do externalities degrade GSP’s efficiency? In *Workshop on Advertising Auctions*, 2012. → pages 37, 102
- [93] C. T. Ryan, A. X. Jiang, and K. Leyton-Brown. Computing pure strategy Nash equilibria in compact symmetric games. In *EC*, 2010. → pages 142
- [94] M. A. Satterthwaite. Strategy-proofness and Arrow’s conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2): 187–217, April 1975. → pages 113
- [95] B. Shi, E. Gerding, P. Vytelingum, and N. Jennings. An equilibrium analysis of competing double auction marketplaces using fictitious play. In *ECAI*, 2010. → pages 10
- [96] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, 2010. → pages 6, 8
- [97] Y. Sun, Y. Zhou, and X. Deng. Optimal reserve prices in weighted gsp auctions: Theory and experimental methodology. In *Workshop on Ad Auctions*, 2011. → pages 97, 98
- [98] D. R. M. Thompson and K. Leyton-Brown. Computational analysis of perfect-information position auctions. In *ACM-EC*, pages 51–60, 2009. → pages iii, 22, 24, 33, 49, 97
- [99] D. R. M. Thompson and K. Leyton-Brown. Revenue optimization in the generalized second-price auction. In *EC*, 2013. → pages iii
- [100] D. R. M. Thompson, S. Leung, and K. Leyton-Brown. Computing Nash equilibria of action-graph games via support enumeration. In *WINE*, 2011. → pages iii, 44, 118
- [101] D. R. M. Thompson, O. Lev, K. Leyton-Brown, and J. Rosenschein. Empirical analysis of plurality election equilibria. In *The 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Saint Paul, Minnesota, USA, 2013. → pages iii

- [102] G. van der Laan, A. J. J. Talman, and L. van Der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of Operations Research*, 12:377–397, 1987. → pages 9, 22, 44
- [103] H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 25:1163–1178, 2007. → pages 2, 4, 5, 9, 34, 35, 96, 97, 105
- [104] R. V. Vohra. *Advanced Mathematical Economics*. Routledge, 2005. → pages 16
- [105] Y. Vorobeychik. Simulation-based game theoretic analysis of keyword auctions with low-dimensional bidding strategies. In *UAI*, 2009. → pages 10
- [106] Y. Vorobeychik. Probabilistic analysis of simulation-based games. *ACM Transactions on Modeling and Computer Simulation*, 20, 2010. → pages
- [107] Y. Vorobeychik. A game theoretic bidding agent for the ad auction game. In *International Conference on Agents and Artificial Intelligence*, 2011. → pages 10
- [108] Y. Vorobeychik, D. M. Reeves, and M. P. Wellman. Constrained automated mechanism design for infinite games of incomplete information. *JAAMAS*, 25:313–351, 2012. → pages 5, 9, 131
- [109] L. Xia and V. Conitzer. Stackelberg voting games: Computational aspects and paradoxes. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, pages 805–810, Atlanta, Georgia, USA, 2010. → pages 115
- [110] L. Xia, M. Zuckerman, A. D. Procaccia, V. Conitzer, and J. S. Rosenschein. Complexity of unweighted coalitional manipulation under some common voting rules. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 348–353, Pasadena, California, USA, July 2009. → pages 113
- [111] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. 32:565–606, 2008. → pages 24
- [112] M. Yokoo, Y. Sakurai, and S. Matsubara. The effect of false-name bids in combinatorial auctions: New fraud in internet auctions. *GEB*, 46:174–188, 2004. → pages 2

Appendix A

Using the PosEc API

Recall that PosEc's main function is to take mechanisms and settings and convert them into (B)AGGs. Thus, the most essential call is

`agg = makeAGG(setting, mechanism)`. Then, the AGG object can be stored to file in a format that Gambit can read:

`agg.saveToFile(filename)`. The challenging part of using PosEc is specifying settings and mechanisms.

A.1 Representing Settings

In PosEc, settings are represented by Python objects. Their constructors are designed to mimic the mathematical representation in the body of the paper. Their constructor have the following forms:

- `Setting(n, O, Theta, u),`
- `BayesianSetting(n, O, Theta, P, u),`
- `ProjectedSetting(n, O, Theta, u, Psi, pi), and`
-

`ProjectedBayesianSetting(n, O, Theta, P, u, Psi, pi),`

where

`n` is an integer specifying the number of agents,

`O` is a “set-like object”¹ containing the set of possible outcomes,

`Theta` is the (finite) collection of types which must be hashable (in non-Bayesian games, this is assumed to be an `n`-length list/tuple where

`Theta[i]` is the type of agent *i*),

`P` is an *n*-length vector of type distributions represented as

`Distribution` objects (essentially mappings from

`Theta` to real numbers),

`u` is a utility function (described below),

`Psi` is a set-like object containing the set of projected outcomes, and

`pi` is a projection function (described below).

Every utility function has the signature

`u(i, theta, o, a_i)` where

i is an integer indicating the agent number,

`theta` is a

`TypeProfile` object (essentially an *n*-length vector of types, but with accessor methods described below),

o is an outcome (from set

`O`), and

`a_i` is agent *i*’s action. Every utility function returns a real value.

¹We define set-like object to mean any object that has

`__contains__()` and `__eq__()` methods. This includes list, tuple and set, as well as several special purpose classes provided by PosEc.

Every projection function has the signature

$\pi(i, o)$ where

i is the agent number and

o is an outcome. Every projection function returns a projected outcome (i.e., an element of the set

Ψ).

A.2 Set-Like Classes for Outcome Spaces

PosEc includes several set-like classes to deal with the fact that often mechanism outcome spaces are too large to represent with conventional Python collections such as

list or

set.

`RealSpace(k)` is a class for k -dimensional real vectors.

`CartesianProduct(*factors)` is a class for Cartesian products of set-like objects. For example, one could create a the outcome space for a single-good auction with n -bidders (i.e., where the outcome space consists of an allocation between 0 and $n - 1$, and an n -length vector of payments) as follows:

```
O = CartesianProduct(range(n), RealSpace(n)).
```

`Permutations(S)` is a class for every permutation of the elements of collection

S .

`PowerSet(S)` is a class containing every subset of

S , while

`PartitionMatroid(S, k)` is a class containing every subset of

S of length at most k .

A.3 Representing Mechanisms

PosEc provides two abstracted mechanism classes: mechanisms with and without projections. These classes have nearly identical constructors:

`Mechanism(A, M)` and

`ProjectedMechanism(A, M)`.

`A` is an action-set function, which has the signature

`A(setting, i, theta_i)` where

`setting` is any valid setting,

`i` is the agent number, and

`theta_i` is that agent's type. The output of an action-set function is always a container (e.g.,

list,

tuple) of actions.

`M` is an outcome function (described below). The output of any outcome function is either an outcome (from `O`) or a

Distribution over outcomes.

For a

`Mechanism`, the outcome function has the signature

`M(`

`setting, a_N)` where

`setting` is a valid setting and

`a_N` is an

`ActionProfile` object (essentially an n -length vector of actions, but with accessor methods described below).

For a

`ProjectedMechanism`, the outcome function has the signature

`M(setting, i, theta_i, a_N)` where

`setting` is a valid setting,

`i` is an agent number,

`theta_i` is agent i 's type and

`a_N` is an

`ActionProfile` object.

`PosEc` includes two probability distribution classes:

`UniformDistribution(events)` and

`Distribution(events, probabilities)`. Both accept finite collections of hash-able events.

A.4 Accessor Methods

Both

`ActionProfile` and

`TypeProfile` objects have many accessor methods. At the most basic level, each is an n -length vector where

`v[i]` returns the i^{th} element.

The accessor methods for an

`ActionProfile` are:

- `a_N.action(j)` – returns the action played by agent `j`
- `a_N.any(A, agents = None)` – returns

True iff any agent plays an action in list

A. If the agents parameter is not

None, it can be a collection of agents to consider.

-

`a.N.argmax(A, fn = lambda x:x, default = None)` – returns some action

a where (1)

a is in

A, (2)

a is played by at least one agent, and (3)

a maximizes

`fn()` subject to (1) and (2).

`default` is returned if no agent plays an action in

A.

-

`a.N.argmin(A, fn = lambda x:x, default = None)` – returns some action in

A or

`default` (see

`argmax`)

-

`a.N.count(a)` – returns a count of how many agents played action

a

-

`a.N.max(A, fn = lambda x:x, default = None)` – returns

`fn(a)` or

default (see

argmax)

-

`a.N.min(A, fn = lambda x:x, default = None)` – returns

`fn(a)` or

default (see

argmax)

-

`a.N.plays(j, a_j)` – returns

True iff agent

`j` played action

`a_j` Optionally,

`j` can be a collection of agents.

-

`a.N.sum(A)` – returns a count of how many agents played any action in collection

`A`.

-

`a.N.weightedSum(A, W)` – returns a weighted sum of how many agents played any action in collection

`A` (The weight for action

`A[j]` is

`W[j].)`

The accessor methods for a

`TypeProfile` are:

-

`argmax(T, fn = lambda x:x, default = None)` – returns some type

`t` where (1)

`t` is in

`T`, (2)

`t` is the type of at least one agent, and (3)

`t` maximizes

`fn()` subject to (1) and (2).

`default` is returned if no agent has any type in

`T`.

-

`theta.any(T)` – returns

`True` iff at least one agent has a type in

`T`

-

`theta.argmin(T, fn = lambda x:x, default = None)` – returns some type in

`T` or default (see

`argmax`)

-

`theta.count(t)` – returns a count of how many agents have type

`t`

-

`theta.hasType(j, theta)` – returns

`True` iff agent

`j` has type

`theta`

-

`theta.max(T, fn = lambda x:x, default = None)` – returns

`fn(t)` or

`default` (See

`argmax`)

-

`theta.min(T, fn = lambda x:x, default = None)` – returns

`fn(t)` or

`default` (See

`argmax`)

-

`theta.sum(T)` – returns a total of how many agents have types in

`T`

-

`theta.type(j)` – returns the type of agent

`j`

-

`theta.weightedSum(T, W)` – returns a weighted sum of how many agents had any type in collection

`T` (The weight for type

`T[j]` is

`W[j].)`

Appendix B

Documentation of the PosEc API

B.1 Module posec.bbsi

B.1.1 Functions

preprocess (<i>agg</i>)
collapseTest (<i>f</i> , <i>C</i> , <i>t</i>)
testCut (<i>agg</i> , <i>act</i> , <i>arcIndex</i>)
tryCutNode (<i>agg</i> , <i>act</i> , <i>node</i>)
tryCut (<i>agg</i> , <i>act</i> , <i>arcIndex</i> , <i>strict=False</i>) If possible (and, optionally, strictly improving) cut an arc
findArcIndex (<i>agg</i> , <i>act</i> , <i>otherNode</i>)
makeSum (<i>agg</i> , <i>act</i> , <i>existingSums</i> , <i>weights</i> , <i>functionName</i>) Create a new weighted sum node that combines a bunch of existing inputs
makeOr (<i>agg</i> , <i>act</i> , <i>sumNode</i> , <i>functionName</i>) Create a new weighted sum node that combines a bunch of existing inputs
makeMax (<i>agg</i> , <i>act</i> , <i>existingMaxes</i> , <i>weights</i> , <i>functionName</i>)
compressByILS (<i>agg</i> , <i>seed=None</i>)
anonymityCuts (<i>agg</i>) pass

B.1.2 Variables

Name	Description
SUM_LOG	Value: []
__package__	Value: 'posec'

B.2 Module posec.mathtools

B.2.1 Functions

isDistribution (d)

d is a list of probabilities

isDelta (d)

cartesianProduct ($listOfLists$)

permutations (S)

subsetPermutations (S)

product (S)

intToBitVector (n , $minBits=0$)

powerSet (S , $cast_fn=<type\ 'list'>$)

B.2.2 Variables

Name	Description
<code>--package--</code>	Value: 'posec'

B.3 Module `posec.posec_core`

B.3.1 Functions

makeAGG(*setting, mechanism, symmetry=False, transform=None, quiet=False, bbsi_level=0*)

Takes a Setting and a Mechanism, returns a corresponding `pyagg.AGG` object

`transform` is a function (`setting,i,theta_i,a_i,o,theta_N`) that returns a real value `quiet==True` produces no standard output `bbsi_level==0` means to do no BBSI `bbsi_level==1` means to do limited BBSI (suitable for fine-tuning games) `bbsi_level==2` means to do extensive BBSI (suitable for discovering coarse structure)

explain(*agg, acts=None*)

Describes the set of function-calls that can be used to produce a strategically equivalent AGG. Optionally, `acts` can cover a specific subset of action nodes.

B.3.2 Variables

Name	Description
<code>__package__</code>	Value: <code>'posec'</code>

B.3.3 Class `ActionProfile`

`posec.posec_core.InstrumentedDataStore` —
ActionProfile

Representation of an action profile with efficient ways of accessing the data.

An `ActionProfile` is passed to a Mechanism's outcome function as the argument `a_N`

Methods

count(*self, a*)

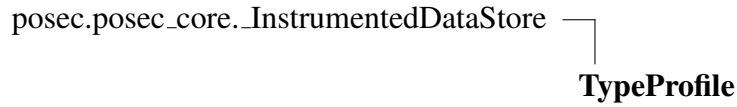
Returns a count of how many agents played action `a`

sum (self, A)
Returns a count of how many agents played any action in collection A
weightedSum (self, A, W)
Returns a weighted sum of how many agents played any action in collection A (The weight for action A[j] is W[j].)
argmax (self, A, fn=<function <lambda> at 0x10147bd70>, default=None)
Returns some action a where (1) a is in A, (2) a is played by at least one agent, and (3) a maximizes fn() subject to (1) and (2). <default> is returned if no agent plays an action in A.
max (self, A, fn=<function <lambda> at 0x10147be60>, default=None)
Returns fn(a) or default (See argmax)
argmin (self, A, fn=<function <lambda> at 0x10147bf50>, default=None)
Returns some action in A or default (See argmax)
min (self, A, fn=<function <lambda> at 0x1014930c8>, default=None)
Returns fn(a) or default (See argmax)
any (self, A, agents=None)
Returns True iff any agent plays an action in list A If the agents parameter is not None, it can be a list of agents to consider
plays (self, j, a_j)
Returns True iff agent j played action a_j Optionally, j can be a list or tuple of agents.
action (self, j)
Returns the action played by agent j
__getitem__ (self, j)
Returns the action of agent j

Inherited from posec.posec_core. InstrumentedDataStore

__init__()

B.3.4 Class TypeProfile



Representation of an type profile with efficient ways of accessing the data.

A TypeProfile is passed to a Setting's utility function as the argument theta_N

Methods

hasType(*self*, *j*, *theta*)

Returns True iff agent *j* has type *theta*

type(*self*, *j*)

Returns the type of agent *j*

__getitem__(*self*, *j*)

Returns the type of agent *j*

count(*self*, *t*)

Returns a count of how many agents have type *t*

sum(*self*, *T*)

Returns a total of how many agents have types in *T*

weightedSum(*self*, *T*, *W*)

Returns a weighted sum of how many agents had any type in collection *T* (The weight for type *T*[*j*] is *W*[*j*].)

any(*self*, *T*)

Returns True iff at least one agent has a type in *T*

```
argmax(self, T, fn=<function <lambda> at 0x1014936e0>,
default=None)
```

Returns some type t where (1) t is in T, (2) t is the type of at least one agent, and (3) t maximizes fn() subject to (1) and (2). <default> is returned if no agent has any type in T.

```
max(self, T, fn=<function <lambda> at 0x1014937d0>,
default=None)
```

Returns fn(t) or default (See argmax)

```
argmin(self, T, fn=<function <lambda> at 0x1014938c0>,
default=None)
```

Returns some type in T or default (See argmax)

```
min(self, T, fn=<function <lambda> at 0x1014939b0>,
default=None)
```

Returns fn(t) or default (See argmax)

Inherited from posec.posec_core.InstrumentedDataStore

```
__init__()
```

B.3.5 Class RealSpace

A set-like object that contains vectors of floating-point numbers

Useful for defining the (projected) outcome space in a Setting

Methods

```
__init__(self, dimensions=None)
```

If dimensions==None, it contains all floating point numbers. If dimensions>=0, it contains all dimensions-length lists & tuples of floating point numbers.

<code>--contains--</code> (<i>self</i> , <i>element</i>)
If dimensions==None, it contains all floating point numbers. If dimensions>=1, it contains all dimensions-length lists & tuples of floating point numbers.
<code>--eq--</code> (<i>self</i> , <i>other</i>)
Returns True iff other is a RealSpace instance with the same dimensions
<code>--repr--</code> (<i>self</i>)

B.3.6 Class CartesianProduct

A set-like object that contains vectors containing elements of the "factor" set

Useful for defining the (projected) outcome space in a Setting

Methods

<code>--init--</code> (<i>self</i> , * <i>factors</i> , ** <i>options</i>)
factors is a list of set-like objects (let k denote its length) contains only k-length vectors where each element is in the corresponding factor options: memberType=<type> - contains only objects of type <type> (also supports tuples of types)
<code>--contains--</code> (<i>self</i> , <i>vector</i>)
Returns True iff vector is a k-length vector where each element is in the corresponding factor if memberType!=None, then vector must also have type <memberType> (
<code>--eq--</code> (<i>self</i> , <i>other</i>)
True iff both inputs are CartesianProducts with the same factors and memberType
<code>--repr--</code> (<i>self</i>)

B.3.7 Class Setting

Known Subclasses: posec.posec_core.BayesianSetting, posec.posec_core.ProjectedExceptionSetting

A data structure representing a (not projected) full-information setting

Methods

<code>__init__(self, n, O, Theta, u)</code>

Class Variables

Name	Description
n	Value: _TypeCheckDescriptor("n", "integer number of agents", lam...
O	Value: _TypeCheckDescriptor("O", "set-like container of outcomes...
Theta	Value: _TypeCheckDescriptor("Theta", "n-length vector of (hash())...
u	Value: _TypeCheckDescriptor("u", "Utility function; u(i,theta,o,...

B.3.8 Class ProjectedExceptionSetting

posec.posec_core.Setting

└─ **ProjectedExceptionSetting**

Known Subclasses: posec.posec_core.ProjectedExceptionBayesianSetting,
posec.applications.position_auctions_externalities.HybridSetting,
posec.applications.position_auctions.NoExternalitySetting

Methods

<code>__init__(self, n, O, Theta, u, Psi, pi)</code> Overrides: <code>posec.posec_core.Setting.__init__</code>

Class Variables

Name	Description
Psi	Value: <code>_TypeCheckDescriptor("Psi", "set-like container of projec...</code>
pi	Value: <code>_TypeCheckDescriptor("pi", "Projection function; pi(i,o) ...</code>
u	Value: <code>_TypeCheckDescriptor("u", "Projected utility function; u(...</code>
<i>Inherited from <code>posec.posec_core.Setting</code> (Section B.3.7)</i>	
O, Theta, n	

B.3.9 Class BayesianSetting

posec.posec_core.Setting

└─ **BayesianSetting**

Known Subclasses: `posec.posec_core.Project BayesianSetting`

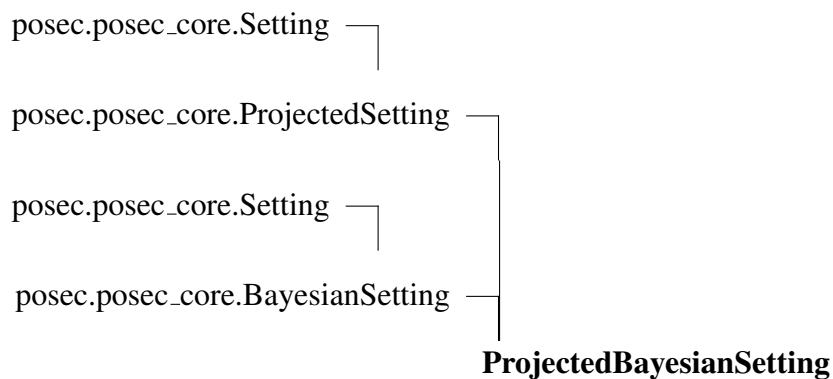
Methods

<code>__init__(self, n, O, Theta, P, u)</code> Overrides: <code>posec.posec_core.Setting.__init__</code>

Class Variables

Name	Description
P	Value: <code>_TypeCheckDescriptor("P",</code> "n-length vector of Distributio...
Theta	Value: <code>_TypeCheckDescriptor("Theta",</code> "Finite collection of (hash...
<i>Inherited from posec.posec_core.Setting (Section B.3.7)</i>	
O, n, u	

B.3.10 Class ProjectedBayesianSetting



Methods

`__init__(self, n, O, Theta, P, u, Psi, pi)`
 Overrides: `posec.posec_core.BayesianSetting.__init__`

Class Variables

Name	Description
<i>Inherited from posec.posec_core.ProjectedSetting (Section B.3.8)</i>	
Psi, pi, u	
<i>Inherited from posec.posec_core.Setting (Section B.3.7)</i>	
O, Theta, n	
<i>Inherited from posec.posec_core.BayesianSetting (Section B.3.9)</i>	
P	

B.3.11 Class Mechanism

Known Subclasses: `posec.posec_core.ProjectedException`,
`posec.applications.voting.AbstractVotingMechanism`


Methods

<code>__init__(self, A, M)</code>

Class Variables

Name	Description
A	Value: <code>_TypeCheckDescriptor("A", "Action function; A(setting, i, ...</code>
M	Value: <code>_TypeCheckDescriptor("M", "Outcome function; M(setting, a...</code>

B.3.12 Class ProjectedException

`posec.posec_core.Mechanism` 
ProjectedException

Known Subclasses: `posec.applications.position_auctions.NoExternalityPositionAuction`,
`posec.applications.basic_auctions.FirstPriceAuction`

Methods

Inherited from `posec.posec_core.Mechanism` (Section B.3.11)

`__init__()`

Class Variables

Name	Description
M	Value: <code>_TypeCheckDescriptor("M", "Outcome function; M(setting, i...</code>
	<i>Inherited from <code>posec.posec_core.Mechanism</code> (Section B.3.11)</i>
A	

B.3.13 Class Distribution

Known Subclasses: `posec.posec_core.UniformDistribution`

Representation of a distribution with finite support

Methods

`__init__(self, support, probabilities)`

support and probabilities are equal-length vectors

`__iter__(self)`

Returns an iterator over 2 tuples representing the probability of an event and the event

`probability(self, event)`

Returns the probability of event Returns 0.0 if event is not in support

B.3.14 Class UniformDistribution

`posec.posec_core.Distribution` —
UniformDistribution

Methods

`__init__(self, support)`

support and probabilities are equal-length vectors

Overrides: `posec.posec_core.Distribution.__init__` extit(inherited documentation)

Inherited from `posec.posec_core.Distribution`(Section B.3.13)

`__iter__()`, `probability()`

B.4 Module posec.pyagg

Python wrapper class for representing action-graph games

B.4.1 Functions

<code>fromLtoLL(<i>L</i>, <i>aSizes</i>)</code>

<code>fromLLtoLC(<i>LL</i>)</code>

<code>fromLLtoString(<i>LL</i>, <i>actionDelim</i>=' ', <i>agentDelim</i>='\t')</code>
--

<code>purgeBarren(<i>agg</i>)</code>

Removes any functions nodes that have no children

B.4.2 Variables

Name	Description
FN_TYPE_SUM	Value: 0
FN_TYPE_OR	Value: 1
FN_TYPE_WEIGHTED_SUM	Value: 10
FN_TYPE_WEIGHTED_MAX	Value: 12
gnm	Value: <code>_Solver("gambit-gnm -n {runs} {filename}", {"runs": 1})</code>
simpdiv	Value: <code>_Solver("gambit-simpdiv -n {runs} {filename}", {"runs": 1})</code>
ipa	Value: <code>_Solver("gambit-ipa -n {runs} {filename}", {"runs": 1})</code>
enumpure	Value: <code>_Solver("gambit-enumpure {filename}")</code>
enumpoly	Value: <code>_Solver("gambit-enumpoly {filename}")</code>

continued on next page

Name	Description
SOLVERS	Value: [gnm, simpdiv, ipa, enumpure, enumpoly]
__package__	Value: 'posec'

B.4.3 Class AGG_File

Known Subclasses: posec.pyagg.AGG, posec.pyagg.BAGG_File

Methods

__init__(*self*, *filename*)

parse(*self*, *strategyString*)

interpretProfile(*self*, *sp*)

fixStrategy(*self*, *stratStr*)

test(*self*, *strategyString*)

Returns a n-length vector of the agents' payoffs

isNE(*self*, *strategyString*)

Tests whether or not a given strategy profile is a (Bayes) Nash equilibrium

__del__(*self*)

B.4.4 Class BAGG_File

```
posec.pyagg.AGG_File └─
                        BAGG_File
```

Known Subclasses: posec.pyagg.BAGG

```
>>> import AGG.Examples
```



```

>>> bagg = AGG.Examples.MixedValueChicken()
>>> bagg.saveToFile("mvc.bagg")
>>> bagg.isNE("1 0 1 0 1 0 1 0")
False
>>> bagg.testExAnte("1 0 1 0 1 0 1 0")
[0.0, 0.0]
>>> baggf = BAGG_File("mvc.bagg")
>>> baggf.Theta
['High', 'Low']
>>> baggf.S["High"]
["('High', 'Swerve')", "('High', 'Straight')"]
>>> baggf.P
{'1': {'High': 0.5, 'Low': 0.5}, '2': {'High': 0.5, 'Low': 0.5}}
>>> baggf.isNE("1 0 1 0 1 0 1 0")
False
>>> baggf.testExAnte("1 0 1 0 1 0 1 0")
[0.0, 0.0]

```

Methods

test (<i>self</i> , <i>strategyString</i>)

Returns an ex interim expected payoff profile

Overrides: posec.pyagg.AGG_File.test

testExAnte (<i>self</i> , <i>strategyString</i>)

Inherited from posec.pyagg.AGG_File(Section B.4.3)

`__del__()`, `__init__()`, `fixStrategy()`, `interpretProfile()`, `isNE()`, `parse()`

B.4.5 Class AGG

posec.pyagg.AGG_File

```

└─
  AGG

```

Known Subclasses: posec.pyagg.BAGG

Methods

sizeAsAGG (<i>self</i>)

sizeAsNFG (<i>self</i>)

`__init__(self, N, A, S, F, v, f, u, title=None)`

N is a list of players A is a list of actions S is a mapping from players to lists of actions F is a list of projection (aka function) nodes v is a list of arcs (2-tuples of start,end nodes) f is a mapping from projection node to type of projection (integer) u is a mapping from an action node to a payoff mapping (tuples of inputs to real values)

```
>>> import AGG_Examples
>>> agg = AGG_Examples.PrisonersDilemma()
>>> agg.saveToFile("pd.agg")
>>> agg.test("NE,1,0,1,0")
[3.0, 3.0]
>>> agg.isNE("NE,1,0,1,0")
False
>>> string.strip(simpdiv.solve(agg).next())
'NE,0,1,0,1'
>>> string.strip(gnm.solve(agg).next())
'NE,0.000000,1.000000,0.000000,1.000000'
>>> f = open("pd2.agg", "w")
>>> f.write(open("pd.agg", "r").read())
>>> f.close()
>>> aggf = AGG_File("pd2.agg")
>>> aggf.test("NE,1,0,1,0")
[3.0, 3.0]
>>> aggf.isNE("NE,1,0,1,0")
False
>>> string.strip(list(enumpoly.solve(aggf))[0])
'NE,0.000000,1.000000,0.000000,1.000000'
```

Overrides: posec.pyagg.AGG_File.__init__

`arcsTo(self, node)`

`neighbours(self, node)`

`saveToFile(self, filename)`

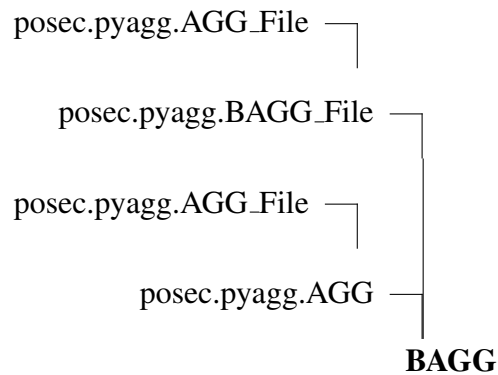
Inherited from posec.pyagg.AGG_File(Section B.4.3)

`__del__()`, `fixStrategy()`, `interpretProfile()`, `isNE()`, `parse()`, `test()`

Class Variables

Name	Description
signature	Value: ' #AGG\n'

B.4.6 Class BAGG



Methods

sizeAsNFG(<i>self</i>)
Returns the size of the games as a Bayesian normal-form game
Overrides: <code>posec.pyagg.AGG.sizeAsNFG</code>
<code>__init__(<i>self</i>, <i>N</i>, <i>Theta</i>, <i>P</i>, <i>A</i>, <i>S</i>, <i>F</i>, <i>v</i>, <i>f</i>, <i>u</i>, <i>title</i>=None)</code>
<i>N</i> is a list of players <i>Theta</i> is a list of types <i>P</i> is a mapping from players to mappings from types to probability <i>A</i> is a list of actions <i>S</i> is a mapping from types to lists of actions <i>F</i> is a list of function nodes <i>v</i> is a list of arcs (2-tuples of start,end nodes) <i>f</i> is a mapping from projection node to type of projection (integer) <i>u</i> is a mapping from an action node to a payoff mapping (tuples of inputs to real values)
Overrides: <code>posec.pyagg.AGG.__init__</code>

Inherited from `posec.pyagg.BAGG_File`(Section B.4.4)

test(), testExAnte()

Inherited from posec.pyagg.AGG_File(Section B.4.3)

__del__(), fixStrategy(), interpretProfile(), isNE(), parse()

Inherited from posec.pyagg.AGG(Section B.4.5)

arcsTo(), neighbours(), saveToFile(), sizeAsAGG()

Class Variables

Name	Description
signature	Value: ' #BAGG\n'

Appendix C

Documentation of the Included PosEc Applications

C.1 Package applications

C.1.1 Modules

- **basic_auctions** (*Section C.2, p. 183*)
- **position_auctions**: This is strictly for no-externality position auctions.
(*Section C.3, p. 188*)
- **position_auctions_externalities** (*Section C.4, p. 192*)
- **voting** (*Section C.5, p. 195*)

C.1.2 Variables

Name	Description
<code>--package--</code>	Value: None

C.2 Module applications.basic_auctions

C.2.1 Functions

ProjectedBayesianSetting(*typeDistros*)

typeDistros is an n-length vector of vectors of floats for each of these vectors, a value of *x* in the *i*th position denotes that an agent has valuation of *i* with probability *x*

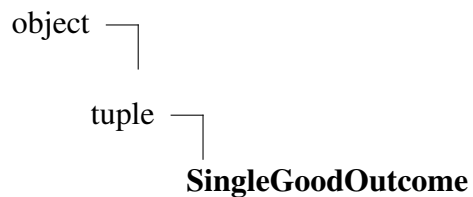
welfareTransform(*setting, i, theta_N, o, a_i*)

paymentTransform(*setting, i, theta_N, o, a_i*)

C.2.2 Variables

Name	Description
SCALE	Value: 10
__package__	Value: 'applications'

C.2.3 Class SingleGoodOutcome



SingleGoodOutcome(*allocation, payments*)

Methods

__getnewargs__(*self*)

Return self as a plain tuple. Used by copy and pickle.

Overrides: tuple.__getnewargs__

<code>__new__</code> (<i>_cls, allocation, payments</i>)
Create new instance of SingleGoodOutcome(allocation, payments)
Return Value
a new object with type S, a subtype of T
Overrides: object. <code>__new__</code>

<code>__repr__</code> (<i>self</i>)
Return a nicely formatted representation string
Overrides: object. <code>__repr__</code>

Inherited from tuple

`__add__()`, `__contains__()`, `__eq__()`, `__ge__()`, `__getattr__()`, `__getitem__()`,
`__getslice__()`, `__gt__()`, `__hash__()`, `__iter__()`, `__le__()`, `__len__()`, `__lt__()`, `__mul__()`,
`__ne__()`, `__rmul__()`, `__sizeof__()`, `count()`, `index()`

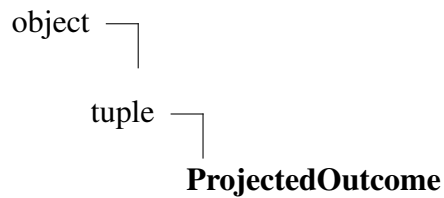
Inherited from object

`__delattr__()`, `__format__()`, `__init__()`, `__reduce__()`, `__reduce_ex__()`, `__setattr__()`, `__str__()`,
`__subclasshook__()`

Properties

Name	Description
allocation	Alias for field number 0
payments	Alias for field number 1
<i>Inherited from object</i>	
<code>__class__</code>	

C.2.4 Class ProjectedOutcome



ProjectedOutcome(i_win, my_payment)

Methods

__getnewargs__(self)

Return self as a plain tuple. Used by copy and pickle.

Overrides: tuple.__getnewargs__

__new__(_cls, i_win, my_payment)

Create new instance of ProjectedOutcome(i_win, my_payment)

Return Value

a new object with type S, a subtype of T

Overrides: object.__new__

__repr__(self)

Return a nicely formatted representation string

Overrides: object.__repr__

Inherited from tuple

__add__(), __contains__(), __eq__(), __ge__(), __getattr__(), __getitem__(),
__getslice__(), __gt__(), __hash__(), __iter__(), __le__(), __len__(), __lt__(), __mul__(),
__ne__(), __rmul__(), __sizeof__(), count(), index()

Inherited from object

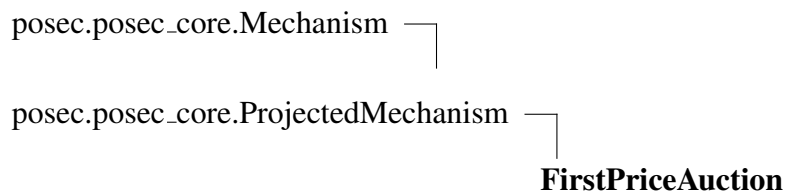
__delattr__(), __format__(), __init__(), __reduce__(), __reduce_ex__(), __setattr__(), __str__(),

`__subclasshook__()`

Properties

Name	Description
<code>i_win</code>	Alias for field number 0
<code>my_payment</code>	Alias for field number 1
<i>Inherited from object</i>	
<code>__class__</code>	

C.2.5 Class FirstPriceAuction



Known Subclasses: `applications.basic_auctions.AllPayAuction`

Methods

`__init__(self, scale=10)`

Overrides: `posec.posec_core.Mechanism.__init__`

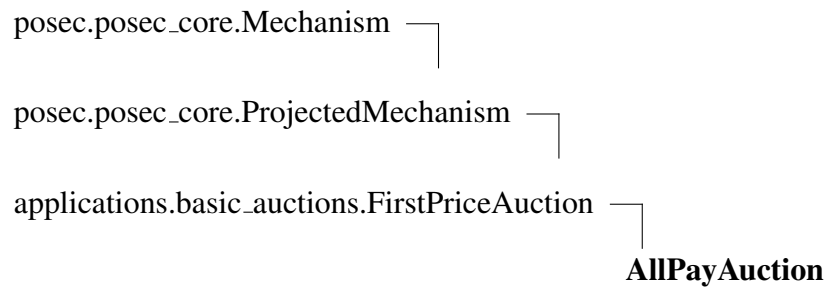
`A(self, setting, i, theta_i)`

Overrides: `posec.posec_core.Mechanism.A`

`M(self, setting, i, theta_i, a_N)`

Overrides: `posec.posec_core.Mechanism.M`

C.2.6 Class AllPayAuction



Methods

M(*self, setting, i, theta_i, a_N*)

Overrides: posec.posec_core.Mechanism.M

Inherited from applications.basic_auctions.FirstPriceAuction(Section C.2.5)

A(), `__init__()`

C.3 Module applications.position_auctions

This is strictly for no-externality position auctions.

C.3.1 Functions

normalizeSetting (<i>setting</i> , <i>k</i>)

Normalizes values and CTR so that the highest valuation is exactly <i>k</i> and the highest ctr is exactly 1. Makes to change to quality scores.
--

makeAlpha (<i>n</i> , <i>m</i> , <i>r</i>)

makeLNAAlpha (<i>n</i> , <i>m</i> , <i>r</i>)
--

EOS (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)
--

Varian (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)

BHN (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)
--

BSS (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)
--

BHN_LN (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)

gauss2 (<i>rand</i> , <i>rho</i>)
--

Sample from a 2D Gaussian (with correlation <i>rho</i>)
--

LNdistro (<i>rand</i> , <i>k</i>)
--

LP (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)

EOS_LN (<i>n</i> , <i>m</i> , <i>k</i> , <i>seed</i> =None)

C.3.2 Variables

Name	Description
LN_CORR	Value: 0.4
LN_params	Value: [0.0, 1.0, 0.0, 1.0]
GENERATORS	Value: {"BHN-LN": BHN_LN, "V-LN": LP, "EOS-LN": EOS_LN, "EOS": E...}
__package__	Value: 'applications'

C.3.3 Class Permutations

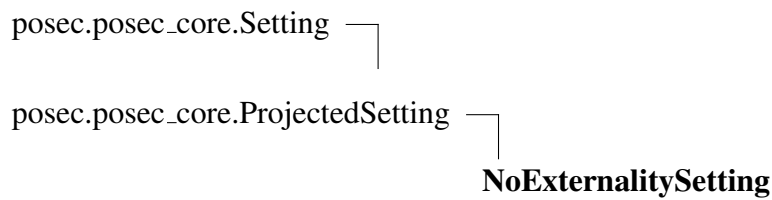
Methods

```
__init__(self, members, lengths=None)
```

```
__contains__(self, element)
```

```
__eq__(self, other)
```

C.3.4 Class NoExternalitySetting



Methods

```
__init__(self, valuations, ctrs, qualities)
```

Overrides: posec.posec_core.Setting.__init__

```
pi(self, i, o)
```

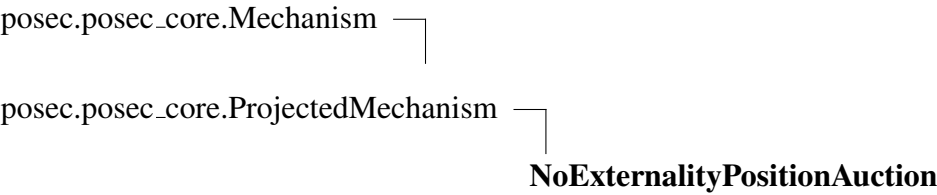
Overrides: posec.posec_core.PjectedSetting.pi

u (self, i, theta, po, a_i)
Overrides: posec.posec_core.Setting.u

Class Variables

Name	Description
<i>Inherited from posec.posec_core.ProjectedException (Section B.3.8)</i>	
Psi	
<i>Inherited from posec.posec_core.Setting (Section B.3.7)</i>	
O, Theta, n	

C.3.5 Class NoExternalityPositionAuction



Methods

_init__ (self, reserve=1, squashing=1.0, reserveType=' UWR' , rounding=None, tieBreaking=' Uniform' , pricing=' GSP')
Overrides: posec.posec_core.Mechanism._init__
q (self, theta_i)
reservePPC (self, i, theta_i)
makeBid (self, theta_i, b, eb)
A (self, setting, i, theta_i)
Overrides: posec.posec_core.Mechanism.A

projectedAllocations(*self, i, theta_i, a_N*)

Returns a list of positions (all the positions that can happen depending on how tie-breaking goes) the last element in the list means losing every tie-break

ppc(*self, i, theta_i, a_N*)

makeOutcome(*self, alloc, price*)

M(*self, setting, i, theta_i, a_N*)

Overrides: posec.posec_core.Mechanism.M

C.4 Module applications.position_auctions_externalities

C.4.1 Functions

`cascade_UNI(n, m, k, seed=None)`

`cascade_LN(n, m, k, seed=None)`

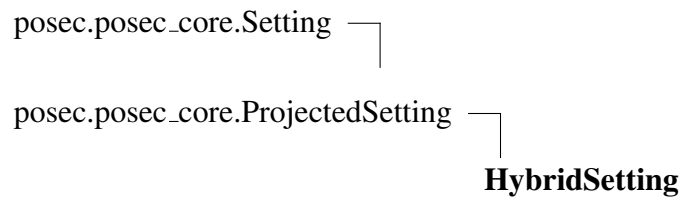
`hybrid_UNI(n, m, k, seed=None)`

`hybrid_LN(n, m, k, seed=None)`

C.4.2 Variables

Name	Description
GENERATORS	Value: {"cascade_UNI": cascade_UNI, "cascase_LN": cascade_LN, "h...}

C.4.3 Class HybridSetting



Methods

`__init__(self, valuations, ctrs, qualities, continuation_probabilities)`

Overrides: posec.posec_core.Setting.*__init__*

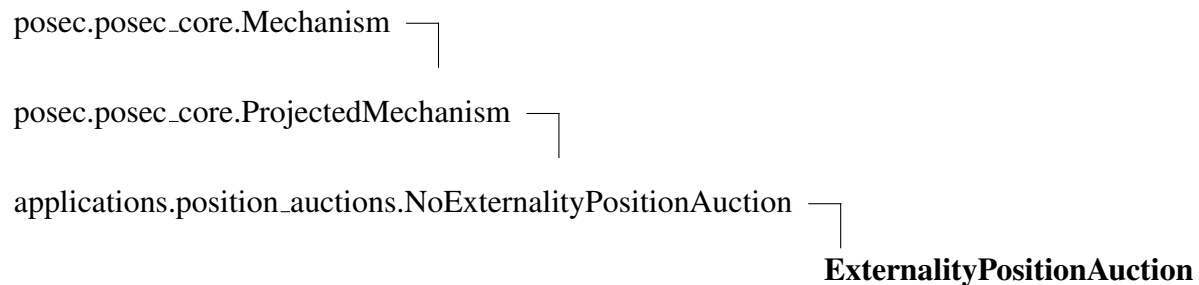
`ctr(self, i, theta, projectedAllocation)`

u (<i>self, i, theta, po, a_i</i>)
Overrides: posec.posec_core.Setting.u

Class Variables

Name	Description
<i>Inherited from posec.posec_core.ProjectedListSetting (Section B.3.8)</i>	
Psi, pi	
<i>Inherited from posec.posec_core.Setting (Section B.3.7)</i>	
O, Theta, n	

C.4.4 Class ExternalityPositionAuction



Methods

makeOutcome (<i>self, alloc, price</i>)
Overrides:
applications.position_auctions.NoExternalityPositionAuction.makeOutcome

makeBid (<i>self, theta_i, b, eb</i>)
Overrides: applications.position_auctions.NoExternalityPositionAuction.makeBid

projectedAllocations(*self, i, theta_i, a_N*)

Returns a list of all the projected allocations (a projected allocation is the list of externality types of agents ranked above i)

Overrides:

applications.position_auctions.NoExternalityPositionAuction.projectedAllocations

Inherited from applications.position_auctions.NoExternalityPositionAuction(Section C.3.5)

A(), M(), __init__(), ppc(), q(), reservePPC()

C.5 Module applications.voting

C.5.1 Functions

leastFavorites(*setting, i, theta_i*)

returns a list of *i*'s least favorite outcomes (or an empty list if the computation can't be done, due to CTD)

mostFavorites(*setting, i, theta_i*)

returns a list of *i*'s most favorite outcomes (or an empty list if the computation can't be done due to CTD)

isRanking(*setting, i, theta_i*)

Returns outcomes listed from *i*'s most to least favorite

settingFromRankings(*rankings, truthfulness=True*)

urn_model_setting(*n, m, a, seed*)

uniform_Setting(*n, m, seed*)

impartialAnonymousCulture_Setting(*n, m, seed*)

threeUrn_Setting(*n, m, seed*)

jMajority_Setting(*n, m, j, seed*)

twoMajority_Setting(*n, m, seed*)

threeMajority_Setting(*n, m, seed*)

isSinglePeaked(*ranking*)

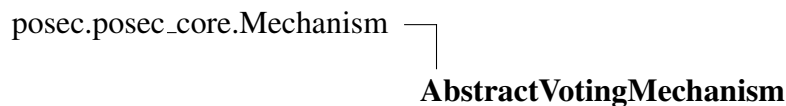
singlePeaked_Setting(*n, m, seed*)

uniformXY_Setting(*n, m, seed*)

C.5.2 Variables

Name	Description
MECHANISM_CLASSES	Value: [Plurality, Approval, kApproval, Veto, Borda, InstantRunoff]
__package__	Value: 'applications'

C.5.3 Class AbstractVotingMechanism



Known Subclasses: applications.voting.Approval, applications.voting.Borda, applications.voting.InstantRunoff, applications.voting.Plurality

Provides a bunch of support features for real voting mechanisms

Methods

__init__(*self*, *randomTieBreak*=True, *removeDominatedStrategies*=False, *allowAbstain*=False)

Overrides: posec.posec_core.Mechanism.__init__

A(*self*, *setting*, *i*, *theta.i*)

returns self.actions(setting, i, theta.i) after (optionally) removing dominated strategies and adding an abstain action

Allows RDS, etc to be re-used

Overrides: posec.posec_core.Mechanism.A

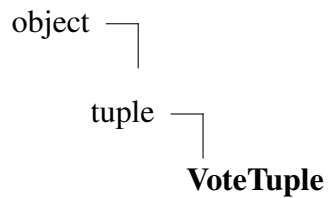
outcome(*self*, *scores*)

Subroutine for M, identifies the maximal-score candidates and breaks ties appropriately

Class Variables

Name	Description
randomTieBreak	Value: True
allowAbstain	Value: False
removeDominatedStrategies	Value: False
<i>Inherited from posec.posec_core.Mechanism (Section B.3.11)</i>	
M	

C.5.4 Class VoteTuple



Like a normal tuple, but it can also mark an action as truthful

Methods

<code>__repr__(self)</code> <code>repr(x)</code> Overrides: <code>object.__repr__</code> <code>extit</code> (inherited documentation)

Inherited from tuple

`__add__()`, `__contains__()`, `__eq__()`, `__ge__()`, `__getattr__()`, `__getitem__()`,
`__getnewargs__()`, `__getslice__()`, `__gt__()`, `__hash__()`, `__iter__()`, `__le__()`, `__len__()`, `__lt__()`,
`__mul__()`, `__ne__()`, `__new__()`, `__rmul__()`, `__sizeof__()`, `count()`, `index()`

Inherited from object

`__delattr__()`, `__format__()`, `__init__()`, `__reduce__()`, `__reduce_ex__()`, `__setattr__()`, `__str__()`,
`__subclasshook__()`

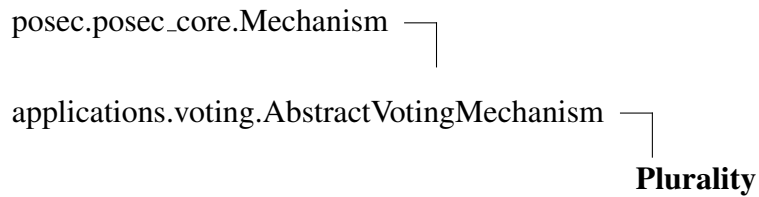
Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

Class Variables

Name	Description
<code>truthful</code>	Value: <code>False</code>

C.5.5 Class Plurality



Methods

actions (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i>)
truthful (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i> , <i>a</i>)
dominated (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i> , <i>a</i>)
M (<i>self</i> , <i>setting</i> , <i>a_N</i>) Overrides: posec.posec_core.Mechanism.M

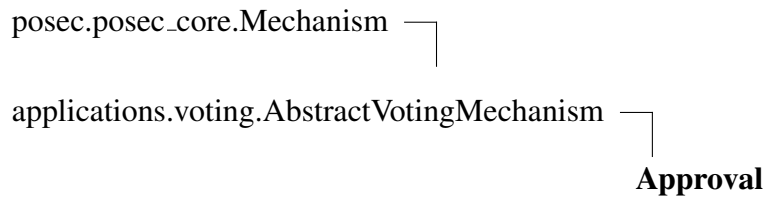
Inherited from applications.voting.AbstractVotingMechanism(Section C.5.3)

`A()`, `--init--()`, `outcome()`

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	

C.5.6 Class Approval



Known Subclasses: applications.voting.Veto, applications.voting.kApproval

Methods

actions(self, setting, i, theta_i)

truthful(self, setting, i, theta_i, a)

dominated(self, setting, i, theta_i, a)

M(self, setting, a_N)

Overrides: posec.posec_core.Mechanism.M

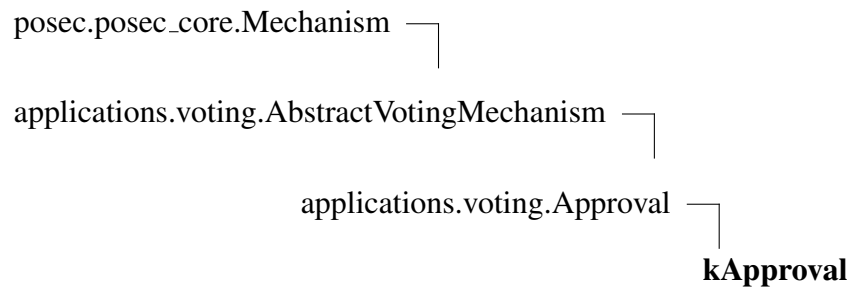
Inherited from applications.voting.AbstractVotingMechanism(Section C.5.3)

A(), __init__(), outcome()

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	

C.5.7 Class kApproval



Methods

`__init__(self, k, randomTieBreak=True, removeDominatedStrategies=False, allowAbstain=False)`

Overrides: `posec.posec_core.Mechanism.__init__`

`actions(self, setting, i, theta_i)`

Overrides: `applications.voting.Approval.actions`

`truthful(self, setting, i, theta_i, a)`

Overrides: `applications.voting.Approval.truthful`

`dominated(self, setting, i, theta_i, a)`

Overrides: `applications.voting.Approval.dominated`

Inherited from `applications.voting.Approval`(Section C.5.6)

`M()`

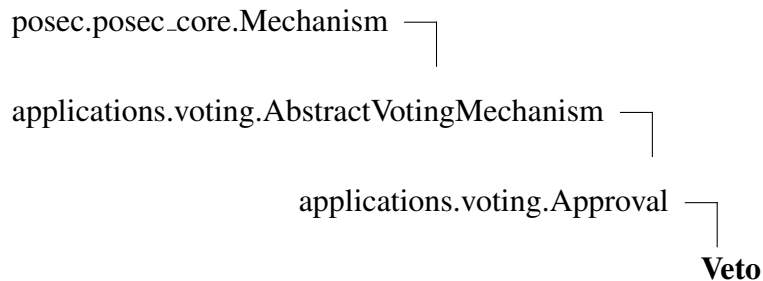
Inherited from `applications.voting.AbstractVotingMechanism`(Section C.5.3)

`A(), outcome()`

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	

C.5.8 Class Veto



Methods

actions(*self, setting, i, theta_i*)
 Overrides: applications.voting.Approval.actions

truthful(*self, setting, i, theta_i, a*)
 Overrides: applications.voting.Approval.truthful

dominated(*self, setting, i, theta_i, a*)
 Overrides: applications.voting.Approval.dominated

Inherited from applications.voting.Approval(Section C.5.6)

M()

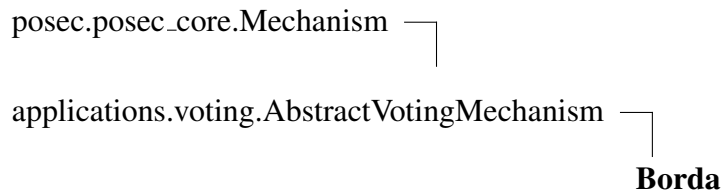
Inherited from applications.voting.AbstractVotingMechanism(Section C.5.3)

A(), __init__(), outcome()

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	

C.5.9 Class Borda



Methods

actions (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i>)

truthful (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i> , <i>a</i>)

dominated (<i>self</i> , <i>setting</i> , <i>i</i> , <i>theta_i</i> , <i>a</i>)
--

M (<i>self</i> , <i>setting</i> , <i>a_N</i>)
--

Overrides: posec.posec_core.Mechanism.M

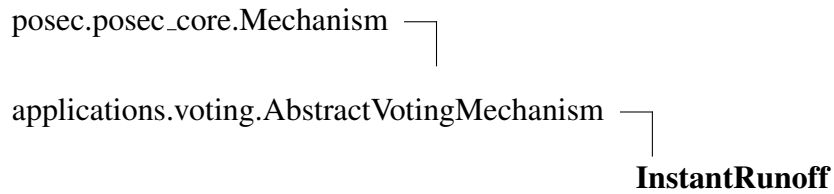
Inherited from applications.voting.AbstractVotingMechanism(Section C.5.3)

A(), __init__(), outcome()

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	

C.5.10 Class InstantRunoff



Methods

<code>__init__(self, allowAbstain=False)</code>
--

FIXME: So far, ties must be broken alphabetically

Overrides: posec.posec_core.Mechanism.__init__
--

<code>actions(self, setting, i, theta_i)</code>
--

<code>truthful(self, setting, i, theta_i, a)</code>
--

<code>dominated(self, setting, i, theta_i, a)</code>

Dominance checking is not implemented by InstantRunoff
--

<code>M(self, setting, a_N)</code>

Overrides: posec.posec_core.Mechanism.M

Inherited from applications.voting.AbstractVotingMechanism(Section C.5.3)

A(), outcome()

Class Variables

Name	Description
<i>Inherited from applications.voting.AbstractVotingMechanism (Section C.5.3)</i>	
allowAbstain, randomTieBreak, removeDominatedStrategies	