2021-08-01

Selecting Robust Strategies when Players do not Know Exactly What Game They are Playing

Oscar Samuel Veliz
University of Texas at El Paso

SELECTING ROBUST STRATEGIES WHEN PLAYERS DO NOT KNOW EXACTLY WHAT GAME THEY ARE PLAYING

OSCAR SAMUEL VELIZ

Doctoral Program in Computer Science

APPROVED:

Christopher Kiekintveld, Ph.D., Chair

Mahmud Shahriar Hossain, Ph.D.

Vladik Kreinovich, Ph.D.

Art Duval, Ph.D.

Stephen L. Crites, Jr., Ph.D.
Dean of the Graduate School

to my

BROTHER

with love

SELECTING ROBUST STRATEGIES WHEN PLAYERS DO NOT KNOW EXACTLY WHAT GAME THEY ARE PLAYING

by

OSCAR SAMUEL VELIZ

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

August 2021

Abstract

Game theory is a tool for modeling multi-agent decision problems and has been used to great success in modeling and simulating problems such as poker, security, and trading agents. However, many real games are extremely large and complex with multiple agent interactions. One approach for solving these games is to use abstraction techniques to shrink the game to a form that can be solved by removing details and translating a solution back to the original. However, abstraction introduces error into the model. This research studies ways to analyze games, abstractions, and strategies that are robust to noise in the game.

Gaining a better understanding of the effect of abstraction in game-theoretic analysis requires a focus on the *strategy selection problem*: how should an agent choose a strategy to play in a game, based on an abstracted game model? This problem has three interacting components: (1) the method for game abstraction, (2) the strategy solution method, and (3) the method for reverse mapping this strategy. This approach has been studied extensively for poker, which is a 2-player, zero-sum game. However, much less is known about how abstraction interacts with strategy selection in more general games.

This research introduces a formal model for studying the situation where players select strategies based on noisy game models, including abstraction. I use two generalized types of abstraction techniques and over a dozen found from the literature. Experimental results of tournaments between agents using abstraction on different classes of games demonstrates that each of these elements has a strong influence on the results. Then I design and develop new agents that are robust against the noise in games.

This research has identified properties of robust solution techniques, evaluated the impact of abstraction on solution quality, presented useful definitions of robustness, and created a new family of solution techniques that allow players to select strategies that are more robust to noise in game models.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Artificial Intelligence (AI) tries to develop automated methods to reason about the world and make intelligent decisions in challenging real-world domains. There are countless ways to model these complex problems and compute solutions to them. Popular examples include classic search problems like the traveling salesman, recommendation engines like Netflix, virtual personal assistants like Amazon's Echo, and even self-driving cars like the one developed by Google.

An important area of research has focused on playing and solving games like checkers [74], chess [14], go [78], and poker [7]. While these seem like simple toy problems they are in reality large and complex, scaling dramatically with the number of players, the size of the board, or the amount of unknown information. Finding solution methods to these kinds of games can translate to solutions for other kinds of multiagent systems. Such applications include trading agents in a market [52], assigning security resources to protect targets [54], managing risk in a computer network [4], and even modeling dynamics of biological systems [3] among many other applications. Any scenario where multiple agents interact and make decisions that impact the outcome for all can be modeled as a game. Expressions like "gaming the system", "two can play this game", "a whole new ballgame", and "endgame" all imply the importance of knowing and understanding strategic reasoning within a "game".

1.2 Brief State of the Art

Many research projects have focused on trying to solve games. In July 2007 Schaeffer, et al., published "Checkers is solved" [74] in Science where they essentially evaluated every possible branch of checkers and showed how with perfect play the game will end in a draw. No such solution has been found the more complex game of chess or even the demonstrably larger game of go [29]. Computers have become very good (even better than humans) at these games because they are games of perfect information, where all agents have access to the entire information state. Games like poker with imperfect information have also been studied and smaller versions of the game like heads-up limit Texas hold'em poker [7] have been solved. Real-world problems modeled as games like security resource allocation [51], trading agent strategies in a smart grid [69], and cryptographic mediator protocols [50] are often larger than games like poker.

It is *not feasible* to try and solve every game but this does not mean we cannot or should not try to find strategies for playing in these games. One approach that can work well is to use machine learning to create a game playing agent [78] which treats the game like a black box. Another approach to playing large games is to use abstraction [71]. Games that are considered intractable to solve can be made tractable through abstraction [47]. The size of a game can be reduced greatly to take up less space and memory [15]. Solving these smaller games requires less computational resources and time than solving larger games [9]. These solutions can be applied to the original game using a reverse mapping process [28]. The solutions are impacted by the abstraction and reverse mapping process which can add a lot of uncertainty and could result in a noisy solution [2].

Although abstraction adds uncertainty, recall that the original game is large and infeasible to solve which leads to the reason for abstraction. A benefit, though, is that abstracted games take up considerably less space and are easier to reason about. Such a benefit is necessary for evaluating and analyzing large real-world problem modeled as a game. Additionally choosing robust strategies that can still perform well given the uncertainty is

incredibly difficult.

1.3 Main Problem and Research Questions

The main goal of this thesis is to address the problem of making robust strategy selections in games. In particular, I focus on situations where players have uncertainty in games and propose a meta-game model represented through added noise or the use of abstraction. Noise can be inherent in a game through differences in the agent's perception of reality, measurement error, or even out-of-date intelligence as a few examples. Abstraction also adds noise since it simplifies the game before analysis, so agents must be robust to the error it introduces. The following are my main research questions which I have investigated and made contributions to research on this problem.

Q1: How can we formally model situations where players use abstraction to simplify games, and then make strategy selections based on these abstracted models? See Chapter 3.

- Developed formal models of meta-games where players use abstraction
- Analyzed the meta-game model

Q2: How can we define strategy robustness, and how can it be evaluated in a rigorous way so we can compare the robustness of different strategy selection methods? See Chapters 2, 3 and 5.

- Defined definition and metrics for robustness
- Developed a testbed for empirical evaluation
- Generated/collected diverse libraries of strategies to support evaluation

Q3: How does the type of abstraction used by the players interact with the methods for strategy selection? See Chapters 5 and 7.

- Tested combinations of different types of abstraction, strategy selection, and classes of games
- Identified interactions, looking for the best combinations of abstraction method and strategy selection

Q4: Perform experiments to evaluate the robustness properties of known strategy selection methods to identify the state of the art, and then develop new strategy selection methods focused on robustness that will improve over the best known methods of robustness. See Chapters 8 and 9.

- Performed simulations with existing strategies selected from the literature
- Analyzed results
- Developed and test new methods based on theoretical/empirical analysis
- Demonstrated the performance of this approach in a worst-case setting

1.4 Methodology Overview

In order to answer these questions I have created a simulation testbed which is able to pit agents against each other in large tournaments while allowing for multiple kinds of strategies, abstractions, levels of abstraction, and controlled noise meta-games. The methodology used to empirically evaluate these strategies, abstraction methods, and robustness is described in the next two sections. For simplicity, this methodology only considers two-player normal form games. This work can be expanded to more players, as well as extensive-form games by translating into the normal form.

I am able to change the parameter inputs to tournaments, specifying the agent's strategy, if the agent is using a form of abstraction or is played in an obfuscated noisy game. I also identify the type of normal form game these agents are playing and the number of games they will play. The system then executes a tournament based on these parameters

and performs a sophisticated analysis computing several robustness properties which I can compare across different tournaments. Most of this has been used to answer Q1 & Q2 but it is also important for answering the other research questions.

Most of this empirical study involved using known and well studied solution methods (Q4) and abstraction techniques (Q3) from the literature. It has also helped identify robustness properties (Q2 & Q4) and to create new strategies that use these properties (Q3). I also did not simply test against the pool of well studied and equilibrium agents (Q4) but also included oracle-like opponents (Q5).

1.5 Contributions

Much of the groundwork and foundation has been set for empirically answering the questions in Section 1.3 and below I will briefly describe some of the research findings with more explained in chapter 10.

This groundwork comes in the form of the applied formal meta-game model and simulation testbed described earlier. Using this testbed there is a trend of poor performance for equilibrium agents using abstraction in zero sum games which supports some of the conclusions in [85]. However, when using symmetric abstractions like TopN and in general sum games the same equilibrium agents actually perform better. In fact most agents (equilibrium or no) clearly show benefits when using TopN abstraction.

Also most of the equilibrium agents perform poorly under asymmetric abstraction which makes sense because they are perhaps converging to what they believe to be the same outcomes. Ultimately, some of the non-equilibrium agents, like Fair and Social, had relatively good performances in general sum games. The rest of the abstraction findings will be explained in Chapter 7 after covering more background material.

I have also identified several key properties of robust strategies and have provided a unifying definition of robustness. Then I developed a class of agents, based on an extension to the quantal-response logit function, that align to the various aspects of the definition of

robustness. These new class of agents are elaborated on in Chapter 8 and experimented on in Chapter 9 using the noise obfuscation version of the novel meta-game model.

Chapter 2

Background

2.1 Game Theory

Game theory is a mathematical tool for making decisions. A game definition in the most basic form defines:

- Players - Also referred to them as agents. An entity that can take actions.
- Actions - The choices that each player must make for example Rock, Paper, and Scissors in Roshambo. If choosing only one action we call this a Pure Strategy (PS) such as always playing Rock. If they decide to apply a mixture of probabilities among the actions we call this a Mixed Strategy (MS).
- Payoffs - Also called utilities, it is the resulting rewards of all players' strategies in a game.

Definition 2.1 (Normal-form Game [57])**.** A (finite, n-person) normal-form game is a tuple (N, A, u), where:

- N is a finite set of n players, indexed by i;
- $A = A_1 \times \cdots \times A_n$, where A_i is a finite set of actions available to player i. Each vector $a = (a_1, \ldots, a_n) \in A$ is called an action profile;
- $u = (u_1, \ldots, u_n)$ where $u_i : A \mapsto \mathbb{R}$ is a real-valued utility (or payoff) function for player i.

This simple game definition can be used to model many real-world problems including simple games like tic-tac-toe or more complicated scenarios that can be modeled as games like trading agents in the stock market. Table 2.1 is an example of a two-player normal form game.

Table 2.1: Example Normal Form Game

	C	D
A	4,4	6,2
B	1,7	5,2

The row player can choose action A or action B. If Row selects action A, then they will receive a payoff of either 4 or 6 depending on the action selected by the column player who can pick from either action C or action D. This game can be evaluated logically assuming both players are playing rationally. In this case, the row player would choose action A because the payoffs of 4 and 6 dominate the payoffs of 1 and 5 which he would get if he selected action B. The column player would choose action C because those payoffs 4 and 7 are higher than the possible payoffs of action D (2 and 2). The resulting outcome of A,C is a stable equilibrium because neither player would opt to change their strategy. When one player selects an action that maximizes his own payoff given the others' actions this is known as a *best response*.

Definition 2.2 (Mixed Strategy [57])**.** Let (N, A, u) be a normal-form game and for any set X let $\Pi(X)$ be the set of all probability distributions over X. Then the set of mixed strategies for player i is $S_i = \Pi(A_i)$.

Definition 2.3 (Best Response [57])**.** Player i's best response to other players ($-i$)'s strategy profile s_{-i} is a mixed strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.

When all players are performing a best response to all other players this is known as a Nash Equilibrium (NE). This is described more formally in Definition 2.4.

Table 2.2: Rock Paper Scissors

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Definition 2.4 (Nash Equilibrium [57])**.** A strategy profile $s = (s_1, \ldots, s_n)$ is a Nash equilibrium if, for all agents i, s_i is a best response to s_{-i}.

The outcome of the example game is a pure strategy Nash equilibrium because all players are playing the best response to all other players. There can also be Nash equilibrium in a mixed strategy. Take for example the classic game Roshambo, also known as Rock-Paper-Scissors, shown in Table 2.2.

In this example there is no stable pure strategy best response as we can see through evaluating different strategies. If Row plays Rock then Column will want to respond with Paper. However if Row believes Column is playing paper he will pick Scissors instead. Then if Column believes that Row is picking Scissors he will switch to Rock and so on and so forth.

In this case the best response is actually a mixed strategy assigning some probability distribution over all actions. There are many ways to compute this kind of strategy including playing repeated games where each agent picks the best response to the opponent's last action, or to a histogram of previous actions, eventually can converge [57]. Much of game theory assumes that the agents are rational [57] although this does not necessarily mean that they are using a Nash equilibrium strategy.

Take the classic example of the Prisoner's Dilemma (PD). Here an officer catches two criminals after they commit a crime. She can get more jail time for them if she can convince them to confess their guilt and Defect against their comrade. She separates the two into separate rooms and offers them both the same deal. Defect against their fellow criminal

Table 2.3: Prisoner's Dilemma

	C	D
C	-1,-1	-4,0
D	0,-4	-3,-3

and they will be let go while the other will spend 4 years in jail. If they both Defect though, they will both receive 3 years of jail-time. Lastly if they decide to trust the other, Cooperate with each other, they will only get 1 year of jail each. The resulting game is shown in Table 2.3.

There exists a pure strategy Nash equilibrium of this game which is for both to Defect. This is because there is always an incentive to Defect. Put another way, from the (Defect, Defect) outcome there is no incentive for one player to have changed his strategy to Cooperate.

Yet in the repeated version of this game it is actually better for both agents to Cooperate [21] assuming they do not know when the game will end. Notice this goes completely against the Nash equilibrium of the game. This is because the -1,-1 outcome is preferable to the -3,-3 outcome. We still do not know everything about this incredibly simple game [58].

There is a variation on Nash equilibrium called ϵ-Nash which finds the pure strategy with the least benefit to deviating.

Definition 2.5 (ϵ-Nash equilibrium [57])**.** Fix $\epsilon > 0$. A strategy profile $s = (s_1, \ldots, s_n)$ is an ϵ-Nash equilibrium if, for all agents i and for all strategies $s_i' \neq s_i, u_i(s_i, s_{-i}) \geq u_i(s_i', s_{-i}) - \epsilon$.

It makes sense to analyze games using equilibrium techniques but there are other important ways to evaluate a strategy. One of the more common approaches is by trying to minimize regret for an agent.

Definition 2.6 (Regret [57])**.** An agent i's regret for playing an action a_i if the other agents

adopt action profile a_{-i} is defined as

$$\left[\max_{a'_i \in A_i} u_i(a'_i, a_{-i})\right] - u_i(a_i, a_{-i}) \tag{2.1}$$

In other words, this is the amount that i loses by playing a_i rather than playing his best response to a_i. Of course, i does not know what actions the other players will take; however, he can consider those actions that would give him the highest regret for playing a_i.

Another common evaluation technique is exploitability. Exploitability is the difference between an agent's expected payoff in a game and their expected payoff against his hypothetical arch-nemesis who lowers the expected payoff as much as possible.

Definition 2.7 (Exploitability [85])**.**

$$\varepsilon = |u(s_i, s_{-i}), \min(u(s_i, s'))|_2 \tag{2.2}$$

where s' is the arch-nemesis strategy.

2.2 Bounded Rationality

A common approach to model an agent with limited computing resources such as time or memory (commonly called bounded rationality) is to use the Quantal Response Equilibrium (QRE). This can be computed by using Equation 2.3 known as *logit*, where λ is a noise parameter between 0 and ∞, i refers to the player, $-i$ the opponent, EU is the expected utility, and P_{ij} is the agent's final computed probability for action j. When $\lambda = 0$ the logit gives a uniform random response. As $\lambda \to \infty$, logit gives a rational best response.

$$P_{ij} = \frac{\exp\left(\lambda EU_{ij}(P_{-i})\right)}{\sum_k \exp\left(\lambda EU_{ik}(P_{-i})\right)} \tag{2.3}$$

Table 2.4: Prisoner's Dilemma Expected Payoff assuming Column is playing {0,1}

	0	1
C	0,0	-4,0
D	0,0	-3,-3

The steps to compute the logit response assuming $\lambda = 0$ is:

$$P_C = \frac{\exp\left(0 \times (0 + -4)\right)}{\sum_{ij} \exp\left(\lambda EU_{ik}(P_{-i})\right)}$$
$$P_D = \frac{\exp\left(0 \times (0 + -3)\right)}{\sum_{ij} \exp\left(\lambda EU_{ik}(P_{-i})\right)}$$

$$P_C = \frac{1}{1+1} = \frac{1}{2}$$
$$P_D = \frac{1}{1+1} = \frac{1}{2}$$

Not surprisingly the result is uniform random. Such a value represents complete uncertainty in the data or can be thought of as having a very low bound on rationality. When $\lambda = 10$:

$$P_C = \frac{\exp\left(10 \times (0 + -4)\right)}{\sum_{ij} \exp\left(\lambda EU_{ik}(P_{-i})\right)}$$
$$P_D = \frac{\exp\left(10 \times (0 + -3)\right)}{\sum_{ij} \exp\left(\lambda EU_{ik}(P_{-i})\right)}$$

$$P_C = \frac{4.28 \times 10^{-18}}{4.28 \times 10^{-18} + 9.36 \times 10^{-14}} \approx 0.00004$$
$$P_D = \frac{9.36 \times 10^{-14}}{4.28 \times 10^{-18} + 9.36 \times 10^{-14}} \approx 0.99995$$

This results in a solution strategy much closer to the a completely rational best response akin to the Nash equilibrium. Notice that the pure best response is to such a strategy is

{0,1} compared to {0.00004,0.99995} so such a strategy still has some bounds on rationality. As $\lambda \to \infty$ the solution strategy approaches the actual best response.

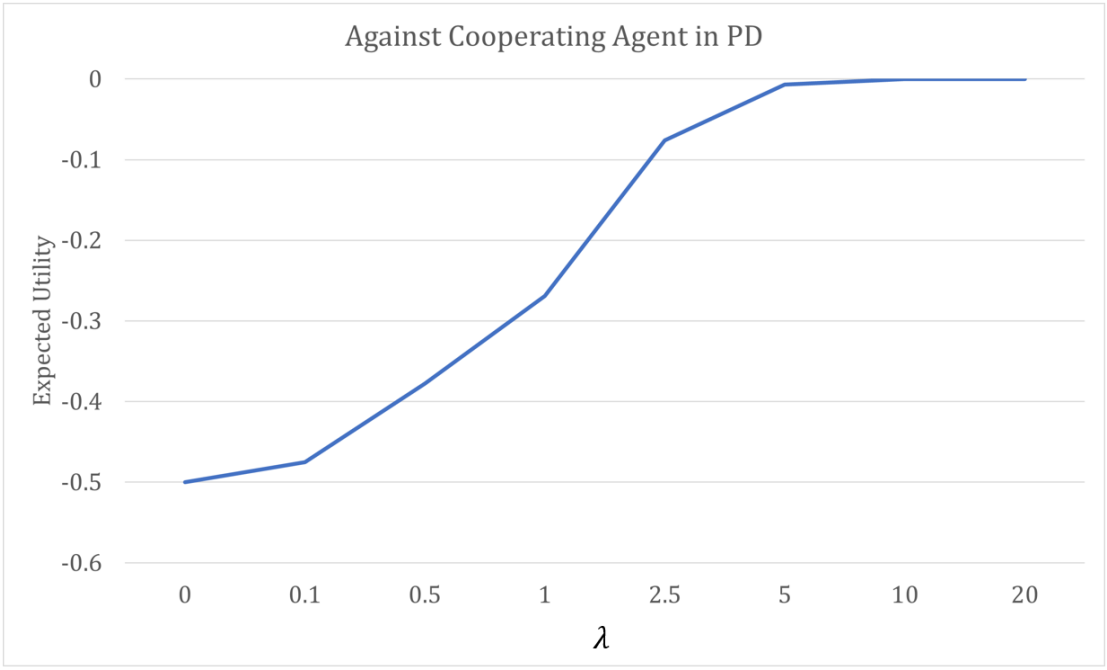


Figure 2.1: Expected Utility of QBR agent against [1,0] as λ increases

An example demonstrating the change in λ against the expected utility is given in Figure 2.1. With $\lambda = 0$ the strategy is uniform random and the payoff in the prisoner's dilemma, against a cooperating, agent is expected as -0.5, which slowly increases as λ increases toward infinity. A table of the strategies used in Figure 2.1 is given in Table 2.5.

An example demonstrating the probability distribution per action as λ changes is given in Figure 2.2. When λ is near zero the probability assigned to each action is nearly uniform. Then as λ increases, more of the distribution is given to actions with higher values.

Table 2.5: QBR strategy in Figure 2.1

λ	QBR
0	[0.5, 0.5]
0.1	[0.47502, 0.524979]
0.5	[0.377546, 0.622459]
1.0	[0.26894, 0.731059]
2.5	[0.075858, 0.9241418]
5	[0.00669, 0.993307]
10	[∼0,0.99995]
20	[∼0.0,0.99999]

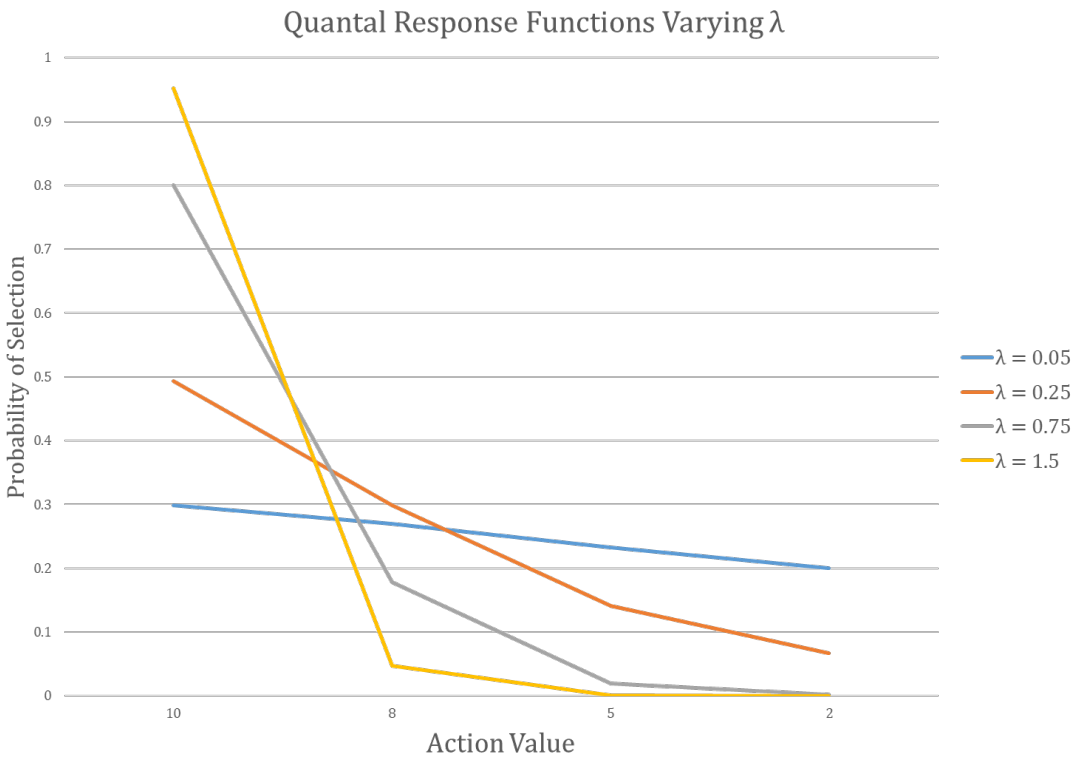


Figure 2.2: Example of changing λ

Chapter 3

Problem Statement

There are many real world scenarios that we would like to model using game theory, however, these models can be quite large and solving every large game is infeasible. Also, agents within these models do not always know other players' strategies are which can be a challenge for maximizing utility. Agents can use abstraction to help reason about large games yet they could use different abstraction methods or even different levels of abstraction. Given a large game model, abstraction, and numerous other agents what is the most robust way to strategically analyze these games?

3.1 Large games, abstraction, and robustness

Example games like chess and Go are incredibly massive in size [29] with equally large sets of strategies. Researchers have been able to solve games like texas hold'em poker [7] because it considers a limited version of the game with only two players and limited bets. Adding additional players exponentially increases the number of possible combinations to consider. Removing the limits on bets would allow any arbitrary bet (consistent with the rules of betting) creating a large number of betting strategies.

This does not mean we can not play these games or even perform well against humans. There are many examples of computer chess playing agents the most famous of which is Deep Blue [74]. Even in go computer agents are able to play at an expert level [29] usually on smaller boards. Go has about 10^{761} possible moves on a 19x19 board compared to chess which has about 10^{120} [29].

Table 3.1: Example Game

	E	F	G	H
A	-4,-4	-6,2	1,3	-10,-10
B	2,7	3,9	3,3	4,5
C	1,7	5,2	1,4	5,-2
D	2,6	4,1	6,5	7,8

Table 3.2: Abstractions

(a) Row Player

	E	F
A	-4,-4	6,2
C	1,7	5,2

(b) Column Player

	G	H
B	3,3	4,5
D	6,5	7,8

3.2 Meta-Game Example

Let us consider Table 3.1 which we will imagine is a large game. This game can be abstracted into a smaller game by all agents and there is no guarantee that the agents will use the same kind of abstraction.

Table 3.2 shows two different abstractions of Table 3.1 with Subtable 3.2a showing the row player's abstraction and Subtable 3.2b representing the column player's. In this example, we have two players who are playing the one-shot normal form game shown at the top of the figure; this is the *base game*. Each player has four possible actions in the original game, and the payoffs are listed in the the matrix. Each player uses a different (unspecified) abstraction method to reduce the size of the game to only two actions for each player, as shown. Now the players analyze these smaller games to select a strategy to play. Here, both of the small games can be solved using dominance to find a unique Nash equilibrium. The players select these strategies (A and H) to play. However, when these strategies are played in the base game they result in the outcome $-10, -10$, which is the

worst possible payoff for both players!

In addition to illustrating the structure of the abstraction meta-game, this example shows one of the possible problems with using abstraction in game analysis. Note that we could replace the payoffs $-10, -10$ with arbitrarily small values. By playing according to a (dominant strategy) Nash equilibrium of the smaller abstracted game, the payoffs the players actually receive are less than the payoffs they expect to received by an unbounded amount. They can also be lower than the equilibrium payoffs in the original game by an arbitrary amount.

This emphasizes the problem of selecting strategies based on abstracted games. Many existing applications of automated abstraction (e.g., in Poker [34, 70, 72, 48]) still analyze the abstracted games to find an (approximate) Nash equilibrium, and play the original game using the equilibrium strategy for the abstracted game. Much of this work focuses on zero-sum games (games where all the outcomes in each payoff sum to zero) where Nash equilibrium is equivalent to the pessimistic minmax solution concept, and has a unique value. However, for general-sum games (games where there is at least one non-zero sum outcome) it is clear from our example that the simple approach of analyzing abstracted games using solution concepts that assume perfect knowledge of the game model is highly problematic. Even in the case of poker, there have been pathologies shown in using abstraction [84], which has led to the exploration of alternative methods for playing games using abstraction that are not based on playing according to an approximate Nash equilibrium [28, 47].

3.3 Robustness

There are many different definitions of the term "robust" depending on context and subject area. Merriam-Webster also defines *robust* with many meanings [61] the most applicable being "capable of performing without failure under a wide range of conditions · *robust* software" although this definition lacks nuance. Hubert et. al claim that "Robustness

signifies insensitivity to small deviations from the assumptions" [44]. We can adapt and expand on this definition to strategic game-playing agents with commonalities among the following related definitions:

> "If a safety-level strategy of an agent guarantees an expected payoff that equals its expected payoff in a Nash Equilibrium, then it can serve as a desirable robust protocol for the agent. ... We are interested in whether an optimal safety level strategy leads to an expected payoff similar to the one obtained in a NE of simple games that represent basic variants of classical CS problems." [82]

> "Our approach does not require an exact model of the game. We only need a lower bound on the gifts (mistakes) that the opponent has given us and an upper bound on the loss from our exploitation. This would be especially useful in security games, since it guarantees robustness even when the game model is not accurate." [27]

> "... robust players seek the maximum guaranteed payoff given the strategy of others. ... we provide robust optimization equilibrium's existence result for a quite general class of games and we prove that it exists a suitable value ϵ such that robust optimization equilibria are a subset of ϵ -Nash equilibria of the normal version." [20]

> "We prove that the robust optimization equilibria of an incomplete information game subsume the *ex post* equilibria of the game and are, unlike the latter, guaranteed to exist when the game is finite and has bounded payoff uncertainty set." [1]

These examples all describe 'robustness' for use in context, and with varying semantics, setting next to terms like 'safety', 'maximum guaranteed payoff', 'bounded payoff uncertainty', 'upper bound on loss', 'not accurate', 'payoff similar to ... NE', rather than trying

to define the actual term. We identified three commonalities of these uses and developed Definition 3.1 for Robustness. This definition is not meant to be all-encompassing, nor apply to every scenario, but as a guidepost for comparing, measuring, and creating robust agents.

Definition 3.1 (Robustness)**.**

1. Maximizing payoffs against unknown opponents when outcomes are uncertain
2. Minimizing loss or exploitability against a worst-case nemesis strategy
3. Earning near-equilibria expected utility against non-equilibrium agents

We propose and discuss several robust (as defined by Definition 3.1) solution techniques in Chapter 8.

Chapter 4

Related Work

An important goal of AI is to make strong game-playing agents [83]. This chapter will focus on the major achievements of computer science and game playing agents motivating the need for abstraction and robust decision making.

4.1 A History of Game Playing

A straightforward way to determine the planning and strategic reasoning ability of a computer, specifically a game playing agent, is to test it against an expert human [73].

4.1.1 Checkers

Starting in 1989 Jonathan Schaeffer and his team built Chinook, a checkers playing agent with a large openings book, strong search and evaluation functions, and an extensive endgame databases [75]. It played against the undisputed greatest Checkers player ever, Marion Tinsley, in 1992 and lost 4-2 with 33 draws [74]. Then during their rematch in 1994 after six games, all draws, Tinsley withdrew because of his health relinquishing his title; he died several months later [75].

Schaeffer et al wrote "The CHINOOK team's success is owed to Marion Tinsley" explaining that at any point Tinsley could have justifiably refused to play the computer yet he welcomed the challenge [75]. They hoped that Tinsley's example would crack the stigma against playing the machine.

The team would eventually go on to solve checkers in their landmark paper "Checkers is Solved" published in *Science* [74]. It turns out the game is a draw if both players are

playing perfectly. Going from checkers to chess Schaeffer writes "[g]iven the effort required to solve checkers, chess will remain unsolved for a long time" due to the shear size of the game and limitations of current technology.

4.1.2 Chess

Regardless the solvability of chess, researches have had immense success at creating strong chess playing agents. The most famous of these is IBM's Deep Blue, a chess playing supercomputer that in 1997 was able to beat then World Champion, and possibly the greatest chess player of all time, Garry Kasparov (3 ½ - 2 ½) [14]. Deep Blue had faced Kasparov before in 1996 and lost the match (4 - 2). What followed was a year of increased engineering and computational power including 480 custom chess chips with the help of multiple grandmasters (Grandmaster Joel Benjamin is specifically acknowledged for his helpful insight) [24].

Deep Blue was very much a brute force machine [23]. As Kasparov writes:

> "It was an impressive achievement, of course, and a *human* [emphasis his] achievement by the members of the IBM team, but Deep Blue was only intelligent the way your programmable alarm clock is intelligent. Not that losing to a $10 million alarm clock made me feel any better." [49]

This alarm clock evaluated 10^6 moves per second which in a game with 10^{134} positions it would not have been able to solve the game even if it started evaluating at the Big Bang 10^{18} seconds ago [23]. IBM retired Deep Blue after its victory [45]. Part of it is in viewing at the National Museum of American History Smithsonian Institution [80].

Since the days of Deep Blue, computers have gotten smarter about which moves they evaluate; so-called "super-grandmaster on your laptop" [49]. The PC program Deep Fritz drew against world champion Vladimir Kramnik (4-4) in 2002 then won against him in 2006 (4-2) [38] [35]. Computer scientists have used machine learning to teach programs to play chess [25]. Deep Junior used machine learning to create its evaluation function [11].

As Ensmenger writes "the construction of a robust evaluation function is more of an art than a science" (there's that word "robust" again) [23].

Before moving onto Go a final word from Garry Kasparov:

> "Perhaps chess is the wrong game for the times. Poker is now everywhere ... a card game whose complexities can be detailed on a single piece of paper. But while chess is a 100 percent information game—both players are aware of all the data all the time—and therefore directly susceptible to computing power, poker has hidden cards and variable stakes, creating critical roles for chance, bluffing, and risk management.
>
> These might seem to be aspects of poker based entirely on human psychology and therefore invulnerable to computer incursion. A machine can trivially calculate the odds of every hand, but what to make of an opponent with poor odds making a large bet? And yet the computers are advancing here as well. Jonathan Schaeffer, the inventor of the checkers-solving program, has moved on to poker and his digital players are performing better and better against strong humans—with obvious implications for gambling sites. "[49]

We will discuss poker research further in Section 4.1.4.

4.1.3 Go

Go is considerably larger than chess with a game tree of size 10^{360}, compared to chess's 10^{123} [83]. Computer Scientists had developed sophisticated Monte Carlo Tree Searching techniques to try and reduce the relevant number of branches [41]. The strongest computer Go agents would compete in the Computer Go UEC Cup with the finalists earning the right to compete in an exhibition game against a professional Go player. In 2013 the program Crazy Stone was able to beat Ishida Yoshio 9-dan (the equivalent of a Grandmaster in chess) with a 4 stone handicap [46]. Crazy Stone and its longtime rival Zen are both ranked as Amateur 6-dan around 1900 Elo ratings [78]. The two programs met in the finals of the

2014 Computer Go UEC Cup where Zen won [56]. As a finalist Crazy Stone was allowed to compete in the exhibition game with a handicap against a pro, this time it was Norimoto Yoda 9-dan, and Crazy Stone won again [19]. In an interview with *Wired* afterwards the creator of Crazy Stone, Rémi Coulom, was asked when a computer would be able to beat a pro without a handicap to which his reply was "I think maybe ten years" [56].

The AI community was incredibly shocked when in January 2016 a Google subsidiary, DeepMind, created a powerful Go playing program, called AlphaGo, that bested professional human player Fan Hui (2-dan and European Champion), without a handicap, winning all five games in the match on a full sized 19x19 board [31]. That same day the DeepMind team published "Mastering the Game of Go with Deep Neural Networks and Tree Search" in *Nature* detailing their approach using Machine Learning without explicitly trying to teach it the game, strategies, openings, nor endings [78]. They reasoned that human players do not examine the entire board with all possibilities, but rather an intuition they've learned over years of play. They also noted that it is difficult to determine who is winning in a game of Go, even for experts. To overcome these two challenges the teams built two large neural networks; one to learn what are good or likely moves given a board position, and the other to determine how likely a player is to win given a board position. From there they used Monte Carlo Tree Search over the most likely moves then evaluated the strength a branch based on how likely it is to lead to a winning position.

Then, like Babe Ruth calling his shot, DeepMind scheduled a match against the world's strongest Go player Lee Sedol (9-dan) of South Korea[31]. Sedol predicted that he would win against AlphaGo 5-0 or 4-1 [10]. Jonathan Schaeffer also sided with Sedol saying:

> "... no offense to the AlphaGo team, but I would put my money on the human. Think of AlphaGo as a child prodigy. All of a sudden it has learned to play really good Go, very quickly. But it doesn't have a lot of experience. What we saw in chess and checkers is that experience counts for a lot."[30]

Demiss Hassabis of DeepMind said that AlphaGo may have gotten that experience,

noting that "AlphaGo has already done the equivalent of over a thousand years of human playing" [55]. The games were broadcast live on YouTube in March of 2016 [22]. AlphaGo won four out of five games and was made an honorary 9-dan [65][12]. Of the experience Sedol said "I misjudged the capabilities of AlphaGo and felt powerless" and "I Lee Sedol lost but mankind did not" [6] [81]. The number one player in the world Ke Jie of China said that "[AlphaGo] can't beat me," adding that even though Hassabis was willing to schedule a match he did not want to play against it saying "I don't want AlphaGo to copy my style" [17]. The latter statement reflects a serious misunderstanding of the machine learning used by AlphaGo.

After the match, DeepMind continued to improve AlphaGo and secretly tested a new version under the alias "Master" in unofficial online games winning 50 in a row, 60 wins overall, no losses, and one draw (because of a connection timeout) [32][42].

Three of those games AlphaGo Master won were against Ke Jie [43]. He claimed to also know Master's identity and that is why "he 'hoped so much' that humans could win at least one game" [43]. Ke Jie posted to Weibo (Chinese site similar to Twitter):

> "I have studied Go softwares for over half a year since March, learning theories and putting them into practices countless times. I only wondered why computers are better... Humans have evolved in games in thousands of years—but computers [are] now tell us humans are all wrong. I think no one is even close to know[ing] the basics of Go."[42]

Ali Jabarin, a Go player, met Ke Jie shortly afterwards recalling that the shocked Jie kept saying "it's too strong" [63].

Ke Jie then played AlphaGo Master again in May of 2017 at the Future of Go Summit, losing against the computer in all three matches [13]. AlphaGo Master was awarded an honorary 9-dan [37]. Deep Mind then retired AlphaGo moving on to AlphaGo Zero and the even stronger AlphaZero [79]. In November 2019 Lee Sedol retired from playing Go [8].

4.1.4 Poker

Games like checkers, chess, and go are games of perfect information where the agents know everything that the other agents know [83]. Poker is a game with hidden information where agents' hands are hidden from each other [36]. In a game of Texas Hold'em poker you are dealt two private cards out of a 52 card deck and two are dealt to your opponent, then there is a round of betting, followed by three community cards (the flop), another round of betting, another community card (the turn), another round of betting, one more card (the river), and a final round of betting. In a 2-player game of limit Hold'em there are 10^{14} decision points whereas with no-limit the size jumps to 10^{160} [62]. One technique for dealing with the uncertainty and game size is to use Abstraction [84, 28, 34, 47, 48, 72, 76]. We will discuss Abstraction more in section 4.2.

Every year the Association for the Advancement of Artificial Intelligence (AAAI) has held the Computer Poker Competition which had been focused on the limit version of Texas Hold'em [85]. Things changed when heads-up (two-player) limit Hold'em was weakly solved by Michael Bowling and his team from the University of Alberta which they published in *Science* in 2015 [7]. Then in late 2016 and early 2017 everything changed again when computers started to beat the pros in two separate and independent tournaments [40, 33].

The first was Michael Bowling and his team with their agent DeepStack [62]. The second was Tuomas Sandholm and his PhD student Noam Brown with their agent Libratus [66]. Both of these agents deserve some discussion.

DeepStack In November 2016 DeepStack played 3,000 hands of heads up no-limit Texas Hold'em against 11 professionals beating 10 of them [33]. The 11th match was also a win but not by a statistically significant margin [18]. The DeepStack team described their agent in an article published in *Science* as:

> "[DeepStack] combines recursive reasoning to handle information asymmetry, decomposition to focus computation on the relevant decision, and a form of intuition that is automatically learned from self-play using deep learning." [62]

The team built upon their earlier work in weakly solving Hold'em by using Counterfactual Regret Minimization (CFR) to solve subgame trees [7]. Where they deviate though is that rather than try and abstract and solve subgames they only looked at the next level in the game tree [62]. This only works for the last major decision (the river) in the tree (at the River or last card); for the flop and turn they do not try to solve the entire subgame strategy. Instead they use heuristics computed from machine learning to estimate the counterfactual values while doing a shallow search in the game tree. The team devotes a lot of their paper to explain how they are *not* using abstraction even though it looks like they are, which we will discuss more in section 4.2.

Libratus Sandholm and Brown had tried to beat human pros before in 2015 and lost with their agent Claudico, although statistically the margin was too close to call [26] [39]. They used that experience in creating Libratus which bested four of the world's top players in 30,000 hands each [67]. While they have not published how Libratus works we can speculate based on news reports, what we know of Claudico, and comments made by the researchers.

Science News reported:

> "Libratus computes a strategy for the game ahead of time and updates itself as it plays to patch flaws in its tactics that its human opponents have revealed. Near the end of a game, Libratus switches to real-time calculation, during which it further refines its methods. Libratus is so computationally demanding that it requires a supercomputer to run. (DeepStack can run on a laptop.)"[18]

That comment about the end game is reminiscent of Claudico's end-game solver on the river which was where it played strongest [26].

IEEE Spectrum reported:

> "[Libratus] would perform 'end-game solving' during each hand to precisely calculate how much it could afford to risk in the third and fourth betting rounds

> ... Sandholm credits the end-game solver algorithms as contributing the most to the AI victory."[40]

We also know that it is not trying to exploit its opponents because according to Sandholm "When you exploit opponents, you open yourself up to exploitation more and more." [40]

Nature reported:

> "Early in a hand, [Libratus] seems to use previously calculated possibilities and the 'translation' approach, although it refines the strategy as the game gives up more information."[33]

This "translation approach" is referring to previous research on abstraction and how solutions of abstracted games are translated into solutions for an original game discussed further in section 4.2. The last part of Libratus's overall strategy was an autonomic approach to fixing itself:

> "After play ended each day, a meta-algorithm analyzed what holes the pros had identified and exploited in Libratus' strategy ... It then prioritized the holes and algorithmically patched the top three using the supercomputer each night." -Sandholm[66]

DeepStack vs Libratus Both of these agents took different approaches towards building strong agents. DeepStack relies on machine learning to give it a sort of "intuition" to help guide its search and runs on a laptop. Meanwhile Libratus ran on Bridge at Pittsburgh's Supercomputing Center and appears to use abstraction, at least pre-flop, before switching to an advanced end-game solver. Additionally, it patched holes in its own play. Both were able to defeat some of the world's best pros. If the two agents ever face-off, which might require equalizing the computing power, the results might not be that telling as "the strategy closest to [the perfect game] doesn't always come out in head-to-head play" and one bot could find an unexpected hole in the other (which might be more troublesome for DeepStack) [33].

4.2 Abstraction

Much of the research into solving large games involves first simplifying the game in a process known as abstraction, then finding a solution of this abstracted and smaller game, and lastly mapping this solution back into the original game [28]. This makes sense because often the problems modeled as games are so large and intractable that trying to find a solution in such a game can take a very long time as well as require a lot of memory and computational resources [47].

However, the solution in the abstracted game is no longer guaranteed to be an optimal solution as the abstraction process adds a lot of noise [15]. The abstractions can also be asymmetric [5]. Then is it still possible to perform an abstraction and get an assurance the solution to the abstraction is even close to optimal? Fortunately it is possible to bound the amount of error introduced into by abstraction and provide some guarantees as to the solution [72, 15].

4.3 Common Themes

Much of the recent literature involving abstraction can be looked at as falling into one of these three themes.

1. Evaluating Equilibria and Solutions
2. Game Reduction and Abstraction
3. Solution Mapping

The first two themes will be looked at in subsection 4.3.1 and 4.3.2 while solution mapping is a theme that is interspersed and dependent on the first two. The literature often uses poker as a proving ground.

4.3.1 Evaluating Equilibria and Solutions

Before Dr. Michael Bowling et al., published their landmark paper in *Science* entitled "Heads-up limit Hold'em poker is solved" [7], Bowling also looked at "Strategy Grafting" [84] with Nolan Bard and Kevin Waugh. This work focused on computing strategies within an abstracted extensive form games (poker) with strong results and a nice theoretical framework.

Many of these same researchers tried to "find optimal abstract strategies" applied to Texas Hold'em [47]. This work is incredibly important as it focused on using Counterfactual Regret Minimization Best Response and showed its relationship with ϵ-Nash equilibrium. It also made certain claims about exploitability of abstracted strategies which are important.

Ganzfried created a solution technique that given an abstracted game would find a non-equilibrium solution and weight it towards higher payoff outcomes. He called this "purification" and "thresholding", one of the first attempts at really creating robust solutions given abstraction [28].

While not necessarily using abstraction, researchers at the University of Michigan performed empirical experiments and tried to solve games with noise and uncertainty [53]. Much of the work of this dissertation will be based on their meta-game testing framework which will become necessary to evaluate robustness.

4.3.2 Game Reduction and Abstraction

Johanson proposes a method for evaluating "abstraction quality" and compares various versions of information recall in heads-up Texas Hold'em [47]. He goes into great detail of how different metrics effect the abstraction and resulting strategies [48] so this paper covers two themes. Researchers (including Bowling) have looked at abstraction methods that try to reduce the number of actions a player can make in state-modeled game, again using Texas Hold'em poker [76]. Gilpin identified two "families" of abstraction algorithms called "expectation-based" and "potential aware" and performed experiments using Rhode

Island Hold'em poker [34] which is another simplified version of poker.

Another top researcher in this area, Tuomas Sandholm, later proved that the abstraction problem is NP-Complete [72]. He also created a framework that guarantees a level of solution quality even when games have been abstracted [72, 71]. In a recent paper he, Brown, and Ganzfried created a hierarchical clustering algorithm and a way to find equilibrium of this game based on Counterfactual Regret (CFR), which when implemented won the 2014 Annual Computer Poker Competition [9].

Chapter 5

Methodology

Section 3.2 presented an example of the meta-game model [53] applied to abstraction. Agents can have their own type of abstraction as well as their own level of abstraction. Their solutions in the abstracted games are mapped back into the original game for evaluating expected payoffs. It is possible that if the abstraction is based on randomness (such as an initial random clustering) the resulting abstractions can be different. We now present a more formal description of abstraction meta-games.

5.1 Formal Model

Both the base game and abstracted games are instances of normal-form games, which can be defined by a tuple $\{I, \{S_i\}, \{u_i(s)\}\}$. The set I represents the players, and the sets S_i are the pure strategies available for each player. The utility function u_i maps each outcome to a real number representing the payoff for the player if this outcome is played. We extend the model to allow mixed strategies in the standard way, defined using the notation σ_i to refer to a mixed strategy, meaning a probability distribution over the pure strategies. Payoffs for mixed strategies are defined using the expected utilities based on the pure-strategy outcomes.

A *Nash equilibrium* is a strategy profile in which every player is playing a best-response to the opponent's strategies. That is, for all players i and all strategies $s_i \in S_i$, $u_i(\Sigma_i, \Sigma_{-i}) \geq u_i(s_i, \Sigma_{-i})$. We also define the *regret* for a strategy profile to the maximum benefit to be the maximum gain that any player can gain by deviating to a different pure strategy. Any strategy profile with zero regret is Nash equilibrium.

An abstraction meta-game has a base game G in the normal form. This is the game that the players will actually play, and their strategy choices in this game define the payoffs they receive. The meta-game extends the description of the base game to include a description of how players select their strategies in the base game. This description has three components. The *abstraction function* (ω_i) for a player maps the base game G into a smaller abstracted game G'_i that the player will analyze. The *selection function* for a player maps from the abstracted game G'_i into a strategy for the player to play in the abstracted game, σ'_i. The *reverse mapping function* for a player maps from the abstracted game strategy σ'_i back to a strategy for the original game σ_i. Collectively, we refer to these three functions as a player's *meta-strategy*. The meta-strategy represents a detailed description of how a player analyzes the base game to select a strategy to play.

5.2 Simulation Design

We have developed a meta-game testbed that allows us to empirically evaluate multiple strategies against each other with or without abstraction and on different classes of games. The simulation first generates a seeded random game based on the game class which is described in more detail later in this section. Then each meta-strategy is a combination of an abstraction method and a solution algorithm. During the simulation the testbed will compute the strategies selected by each meta-strategy and then play a round-robin tournament among these strategies. The payoffs are based on the strategy profile that is played in the original game.

The result of the tournament is a payoff matrix for the meta-game where each meta-strategy can be played by either player. To estimate this matrix for a class of base games we average the payoffs over a large number of sampled games. We can analyze this payoff matrix to generate several performance metrics for the meta-strategies. Average payoffs are quite biased, in that they can reward strategies that only do well against other weak strategies. Instead, we present our results using *stability* and *average regret* measures.

Stability is a measure of how close the pure strategy profile where all players use the same meta-strategy is to being a pure Nash equilibrium. It is calculated by finding the maximum gain for deviating to another meta-strategy from this profile. Any value less than or equal to zero indicates that the profile is a pure equilibrium of the estimated meta-game. Average regret is the average of the regret for playing a given meta-strategy against each other meta-strategy in the estimated game.

We use three classes of randomly-generated games: general sum, zero sum, and logical games. General sum and zero sum are generated using GAMUT [64]. Players have 20 pure strategies, and payoffs are real numbers in the range $[-100, 100]$. Logical games have more structure and should be more amenable to abstraction. Pure strategies are based on choosing either 0 or 1 for a set of boolean variables. Payoffs are based on randomly generated oppositions with varying numbers of clauses, literals, and values. If a proposition is true for a given outcome in the game, the payoff associated with the proposition is added to the payoff for that player. The payoffs are additive over all true propositions for a given outcome. Finally, payoffs are normalized to the range [0,100]. We also could not possibly evaluate every level of abstraction so we picked several interesting abstraction levels. An abstraction level of 10 means that there are now 10 actions in the game, Top10 or 10 clusters, and 20 indicates these are the original 20 pure strategies with no abstraction.

Chapter 6

Methods

6.1 Abstraction

We define an abstraction method as a function that maps one normal-form game into a second (smaller) normal-form game. A good abstraction should be simpler than the original, but also retain as much information about the strategic interaction as possible. We identify two broad categories of abstractions that are common in the literature: *strategy elimination* and *strategy combination*.

Strategy elimination abstractions remove some strategies completely to reduce the size of the game. The well-known method of iterated elimination of dominated strategies [77] is one example of this approach. Weakening the criteria for dominance so that weakly or approximately dominated strategies are also removed leads to even smaller games at the cost of potentially removing some equilibria of the original game [15].

Abstractions methods based on strategy combination simplify games by merging multiple strategies into a single representative strategy. In the extreme case where two strategies have exactly identical payoffs for a player in every outcome merging these strategies results in a lossless abstraction. The idea can be extended to merge strategies that are only similar rather than identical, but the result is a lossy abstraction. A common approach in the literature is to use clustering algorithms to merge similar strategies into clusters. In extensive form games, clustering can also be applied to information states [34]. A different type of similarity between strategies can be used to solve a game by reducing it to sub games [16].

We are interested in the interaction between abstractions and the solution methods applied to abstracted games. Therefore, we selected two simple abstraction methods for

our initial study that are representative of the broad categories described above. We also do not try to guarantee any bounds on these general abstractions as shown in Table 3.1 with potentially unbounded error. It is also the case that the simulated agents may not be using an equilibrium or convergent strategy.

6.1.1 TopN Abstraction

The first abstraction method we consider is *TopN*. This method creates a smaller game by selecting a subset of size N of the strategies for each player to form the abstracted game. For each strategy in the game, we first calculate the expected payoff of the strategy against a uniform random opponent strategy. We then select the N strategies with the highest expected payoffs, breaking ties randomly. The abstracted game is simply the game where players are restricted to playing only the N selected strategies; the payoffs are unchanged. Since each strategy in the abstracted games is also a strategy in the original game the reverse mapping of the strategies back to the original game is trivial.

6.1.2 K-Means Clustering Abstraction

The second abstraction method we use is *KMeans*, which is representative of strategy combination methods. This method uses the k-means clustering algorithm to group strategies into clusters based on similarity of their payoffs. Each strategy is represented in the clustering algorithm by a vector of the payoffs for the strategy in every outcome that can result when the strategy is played. We cluster the strategies for each player separately, following the standard k-means algorithm. Each strategy is initially assigned randomly to one of the k clusters. The centroid is calculated for each cluster, and then the Euclidean distance between every strategy and every cluster centroid is calculated. Strategies are re-assigned to the closest cluster, ensuring that no cluster becomes empty. This process iterates until there are no further changes in the clusters. We run the clustering 100 times with different initial clusterings, and select the one with the smallest maximum distance between any

strategy in a cluster and the centroid.

Once the strategies are clustered we create the abstracted game as follows. Each cluster maps to a pure strategy in the abstracted game (so the number of strategies is controlled by the k parameter used in k-means). The payoffs for each outcome in the abstracted game are computed by averaging the payoffs for all of the outcomes in the cluster. In other words, we assume that players will play each strategy in a given cluster with equal probability. The reverse mapping also assumes that players play strategies in the same cluster with uniform random probability. For example, if a strategy in the abstracted game places a probability of *0.5* on playing a given pure strategy, this probability would be distributed uniformly over all of the strategies that comprise that cluster in the strategy used to play the original game.

6.2 Solution Methods

There are many distinct strategies that an agent might use to solve a game. If you consider that a mixed strategy assigns a real number to each action probability then there are an infinite number of strategies [57]. A rational agent might try to be unpredictable and use a Uniform Random (UR) strategy. Or he might assume that his opponent is using UR and therefore decide to use a Best Response to Uniform (BRU).

Definition 6.1 (Uniform Random)**.** The uniform random strategy is a mixed strategy profile

$$s = \left(\frac{1}{|S|}, \ldots, \frac{1}{|S|} \right) \tag{6.1}$$

Definition 6.2 (Best Response to Uniform Random)**.** This is a mixed strategy profile the computes the best response assuming other players are using UR.

$$s_i = BR_i^G(UR) \tag{6.2}$$

Consider another common strategy for playing in a game, the MaxMin strategy.

Definition 6.3 (MaxMin [57])**.** The maxmin strategy for player i is

$$s_i = \arg\max_{s_i} \min_{s_{-i}} \; u_i(s_i, s_{-i}), \tag{6.3}$$

Social and Fair agents are also common strategies found in the literature [68]. Fair is a heuristic strategy that focuses on outcomes that are "fair" in that there is a small difference between the payoffs for the players. For every strategy profile we calculate the difference between the payoffs and select an outcome that minimizes this difference. Ties are broken in favor of outcomes with a higher sum of payoffs, and then divided uniformly. Social agents play according to the outcome that maximizes the sums of the payoffs for all players. If there are ties the strategy plays a uniform random strategy over the strategies in the tied outcomes

Definition 6.4 (Social [68])**.** Selects action that maximizes social welfare for all agents selecting the outcome with the highest overall utility sum.

$$s_i = \arg \max_{j \; s.t. \; j \in A_i, k \in A_{-i}} (u_i(a_j, a_k) + u_{-i}(a_j, a_k)) \tag{6.4}$$

Note that in a zero sum game all actions are identical to a Social agent (every outcome sums to zero).

Definition 6.5 (Fair [68])**.** Selects action(s) that maximizes fairness for all agents.

$$s_i = \arg \min_{j \; s.t. \; j \in A_i, k \in A_{-i}} (|u_i(a_j, a_k) - u_{-i}(a_j, a_k)|) \tag{6.5}$$

If there is a tie, agents will break ties for outcomes with higher payoff sums.

Quantal Response Equilibrium [60] (QRE) also originated in behavioral game theory. It incorporates a model of *noisy best-response* where players use a noisy best-response instead of strict best-response. A logistic function is normally used to specify this response, and it has a parameter λ that interpolates between a uniform random strategy when $\lambda = 0$ and a best response as $\lambda \to \infty$. A QRE is defined similarly to a Nash equilibrium, except that both players play noisy best-responses to each other for some value of λ. QRE has been

shown to provide a better fit for experimental data on human behavior in some games [60]. QRE has also been shown to have more robust strategy choices than NE in situations where players make choices based on noisy observations of an underlying game [53]. We compute a QRE using Gambit [59] and play a best-response to the predicted QRE strategy of the opponent.

Definition 6.6 (Quantal Best Response [60, 86])**.** Let $u_i(a_i, s_{-i})$ be agent i's expected utility in game G when playing action a_i against strategy profile s_{-i}. Then a *(logit) quantal best response* $QBR_i^G(s_{-i}, \lambda)$ by agent i to s_{-i} is a mixed strategy s_i such that

$$s_i(a_i) = \frac{\exp[\lambda \cdot u_i(a_i, s_{-i})]}{\sum_{a_i'} \exp[\lambda \cdot u_i(a_i', s_{-i})]} \tag{6.6}$$

where λ (the *precision* parameter) indicates how sensitive agents are to utility differences, with $\lambda = 0$ corresponding to uniform randomization and $\lambda \to \infty$ corresponding to best response. Note that unlike best response, which is a set-valued function, quantal best response always returns a unique mixed strategy.

We can generalize Nash equilibrium to form the *quantal response equilibrium* [60, 86].

Definition 6.7 (Quantal Response Equilibrium [60, 86])**.** A *quantal response equilibrium* with precision λ is a mixed strategy profile s^* in which every agent's strategy is a quantal best response to the strategies of the other agents. That is, $s_i^* = QBR_i^G(s_{-i}^*, \lambda)$ for all agents i.

There is another common strategy called Cognitive Hierarchy (CH) that uses an iterated best response function that can be used to model how humans might think. This is derived from behavioral game theory and studies in Psychology. It models a recursive style of reasoning where Level-0 agents play a uniform random strategy, Level-1 agents play a best response to the Level-0 agents, Level-2 agents play a best response to a mixture over level 0 and 1 agents, etc. Agents at each level use a Poisson distribution, based on a parameter τ, to predict the probabilities of playing agents at lower levels, and play a best response to this mixture.

Definition 6.8 (Poisson-CH model [86])**.** Let $\pi^{PCH}_{i,m} \in \Pi A_i$ be the distribution over actions predicted for an agent i with level m by the Poisson-CH model. Let $f(m) = \text{Poisson}(m; \tau)$. Let $BR^G_i(s{-}i)$ denote the set of i's best responses in game G to the strategy profile s_{-i}. Let

$$\pi^{PCH}_{i,0:m} = \sum_{\ell=0}^{m} f(\ell) \frac{\pi^{PCH}_{i,\ell}}{\sum_{\ell'=0}^{m} f(\ell')} \tag{6.7}$$

be the "truncated" distribution over actions predicted for an agent conditional on that agent's having level $0 \leq \ell \leq m$. Then π^{PCH} is defined as

$$\begin{aligned} \pi^{PCH}_{i,0}(a_i) &= |A_i|^{-1} \\ \pi^{PCH}_{i,m}(a_i) &= \begin{cases} |BR^G_i(\pi^{PCH}_{i,0:m-1})|^{-1}, & \text{if } a_i \in BR^G_i(\pi^{PCH}_{i,0:m-1}); \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{6.8}$$

The overall predicted distribution of actions is a weighted sum of the distributions for each level,

$$\Pr(a_i \mid G, \tau) = \sum_{\ell=0}^{m} f(\ell) \cdot \pi^{PCH}_{i,\ell}(a_i). \tag{6.9}$$

The mean of the Poisson distribution, τ, is thus this model's single parameter.

Quantal Level-k is a method that combines the features of QRE and CH. It uses a recursive reasoning model identical to CH, except that in place of the best-response at each level agents play a noisy best-response using the same logit function used in QRE. It has parameters for both λ and τ. We play a best-response to the predicted strategy of the opponent, based on the specified level of reasoning.

Definition 6.9 (Quantal Level-k [86])**.** The probability distribution $\pi^{QLK}_{i,k} \in \Pi(A_i)$ over actions that QLK predicts for a Level-k agent i is

$$\begin{aligned} \pi^{QLK}_{i,0}(a_i) &= |A_i|^{-1}, \\ \pi^{QLK}_{i,1}(a_i) &= QBR^G_i(\pi^{QLK}_{-i,0}; \lambda_1), \\ \pi^{QLK}_{1(2)}(a_i) &= QBR^G_i(\pi^{QLK}_{-i,0}; \lambda_{1(2)}), \\ \pi^{QLK}_{i,2}(a_i) &= QBR^G_i(\pi^{QLK}_{i,1(2)}; \lambda_{(2)}), \end{aligned} \tag{6.10}$$

where $\pi_{1(2)}^{QLK}(a_i)$ is a mixed-strategy profile representing Level-2 agents' prediction of how other agents will play. The overall predicted distribution of actions is the weighted sum of the distributions for each level,

$$Pr(a_i|G, \alpha_1, \alpha_2, \lambda_1, \lambda_2, \lambda_{1(2)}) = \sum_{k=0}^{2} \alpha_k \pi_{i,k}^{QLk}(a_i), \tag{6.11}$$

where $\alpha_0 = 1 - \alpha_1 - \alpha_2$. The QLk model thus has five parameters: $\alpha_1, \alpha_2, \lambda_1, \lambda_2, \lambda_{1(2)}$.

For simplicity in testing we set $\alpha_1 = \alpha_2$ and $\lambda_1 = \lambda_2 = \lambda_{1(2)}$ so in reality the only parameter to set is a single λ which means that QLK is the same as Poisson-CH except that it uses a quantal best response at each level.

6.3 Strategy Selection Problem

Game theorists usually presume that all agents are behaving rationally but what does 'rational' really mean? We specify a model of the strategy selection problem when players use asymmetric abstractions as a *meta-game*. In this model players can use different methods for abstracting the game, solving the game, and reverse-mapping the solution.

In this sense even though all of the strategies defined in Section 6.2 all behave and model the problem differently, all of these solutions are rational. There is no solution that is inherently more rational than any other strategy although some strategies are expected to perform better than others. Take Nash equilibrium as an example. It assumes that all agents are playing a best response to all other agents so if there is one agent that is not using an equilibrium strategy then there is no guarantee that the outcome will be an equilibrium.

Yet in the literature equilibrium or approximate equilibrium are the most common solution techniques evaluated. Abstraction researchers seem to have focused less on other approaches to solving games. This is why we have chosen to include them because they have a chance to be more robust.

Chapter 7

Abstraction Experiments

7.1 Abstraction Results

Our experiments are designed to test the relationships between abstraction methods and solution concepts across different classes of games, and with varying levels of abstraction. In total, we experiment with three different classes of games, two types of abstraction, four levels of abstraction, and more than 30 different solution methods (including multiple parameter settings for QRE, CH, and QLK). The main results span 24 different tournaments with 500 sample games played in each one.

The full data set is difficult to visualize, so we will present results focusing on selected subsets of agents to illustrate key points. We focus on measures of stability and regret here as we believe they are the most informative, but we have also looked at measures of average payoffs and exploitability. Average payoffs can be misleading for agents that exploit weak players but perform poorly against strong players. Exploitability does not differentiate well among the meta-strategies, since almost all of them are highly exploitable in the worst case.

We first present a set of results that demonstrates how we compare the different parameter settings for QRE, CH, and QLK. The purpose of this analysis is to discover which parameter settings are strongest. In later results we will include only best parameter settings for each solution method to aid in visualization.

Figures 7.1 and 7.2 show results for QRE with different values of the λ parameter. Each plot shows results for all three classes of games. There are separate plots for each type of abstraction. The x-axis shows the level of abstraction where the values are the number of actions in the abstracted game. The point on the far left corresponds to no abstraction,

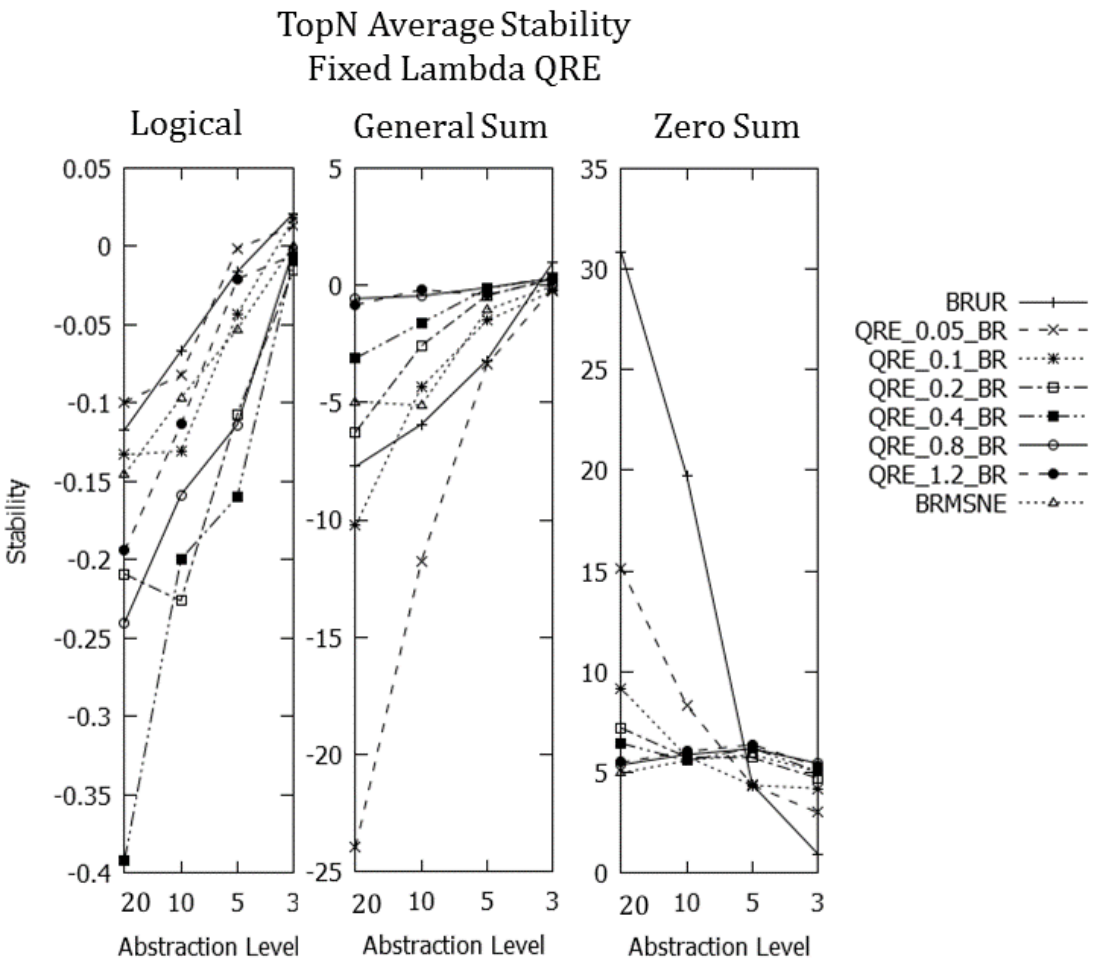


Figure 7.1: TopN Stability Fixed Lamda QRE

and more information is lost for data points further to the right. The y-axis represents the stability (ϵ) value for each solution method.

Lower stability values are better. In particular, any value less than or equal to 0 indicates that a solution method is a pure-strategy Nash equilibrium, and if all players were using this method none of them would benefit by deviating to any other solution method. These stabilities are calculated with respect to the full data set, meaning that players are allowed to deviate to *any* solution method, not just the ones in the figure.

In general sum and logical games, the best QRE parameter settings are closer to playing

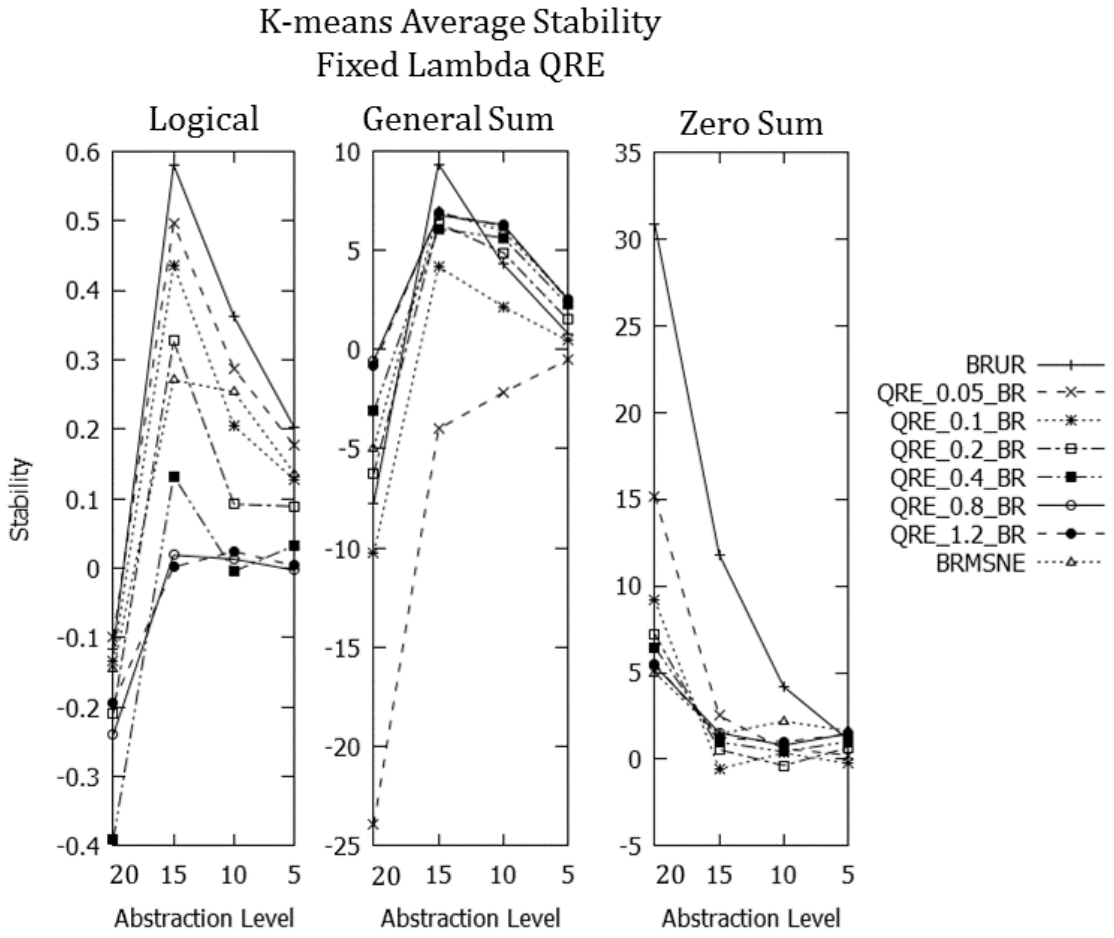


Figure 7.2: K-means Average Stability for Fixed Lambda QRE

a best-response to a more random opponent than playing a best-response to an opponent closer to a Nash equilibrium. For example, the very low parameter setting of $\lambda = 0.05$ performs very well, and this setting is the closest to uniform random. Zero-sum games appear to behave quite differently, where playing closer to an equilibrium strategy performs better.

We perform a similar set of experiment to determine the parameterizations of CH and QLK agents for the values of k, τ, and λ. We use a related procedure to identify the most effective parameter settings to visualize in the later plots. The values of τ considered are 1.5, 2.5, 5, 7.5, and 9, the values of k are 0, 1, 2, 3, 5, 10, and 20, and the values of λ

are 0.8, 1.5, 2.0, 5.0, and 20. For QLK agents we also consider both version that play the raw QLK strategy and ones that play a best response to the strategy of the opponent for a particular level. One results to note from this analysis is that for CH agents with the same τ but different levels of reasoning there is not much difference between agents with mid-level k versus high level k. The main variations between these agents occur at lower levels. However, higher levels of k typically have better, more robust performance.

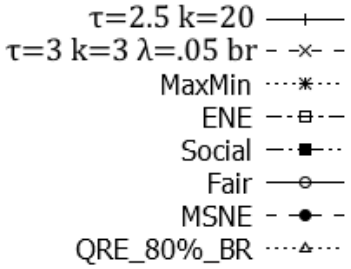
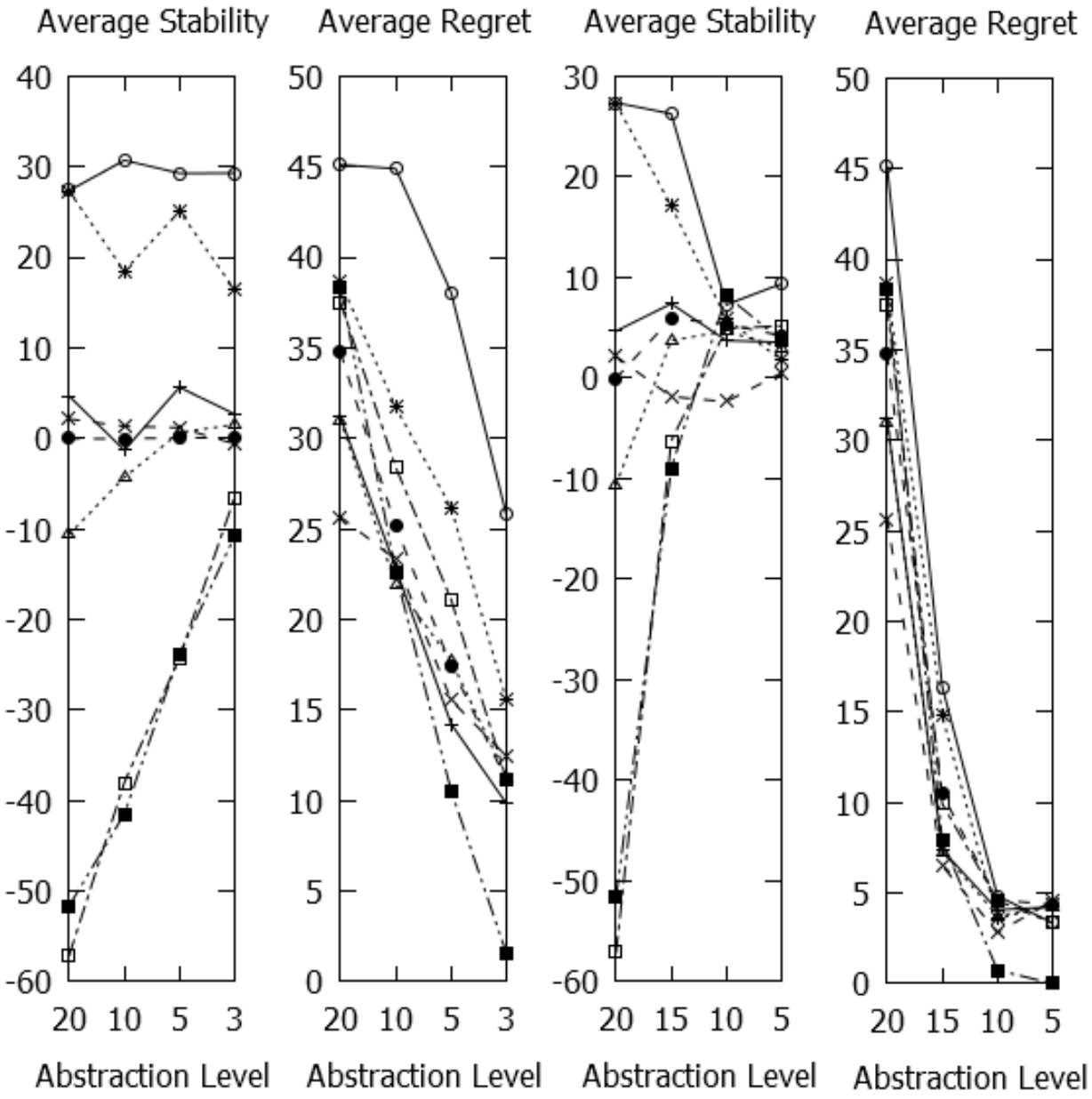


Figure 7.3: Key for Figures 7.4 and 7.5

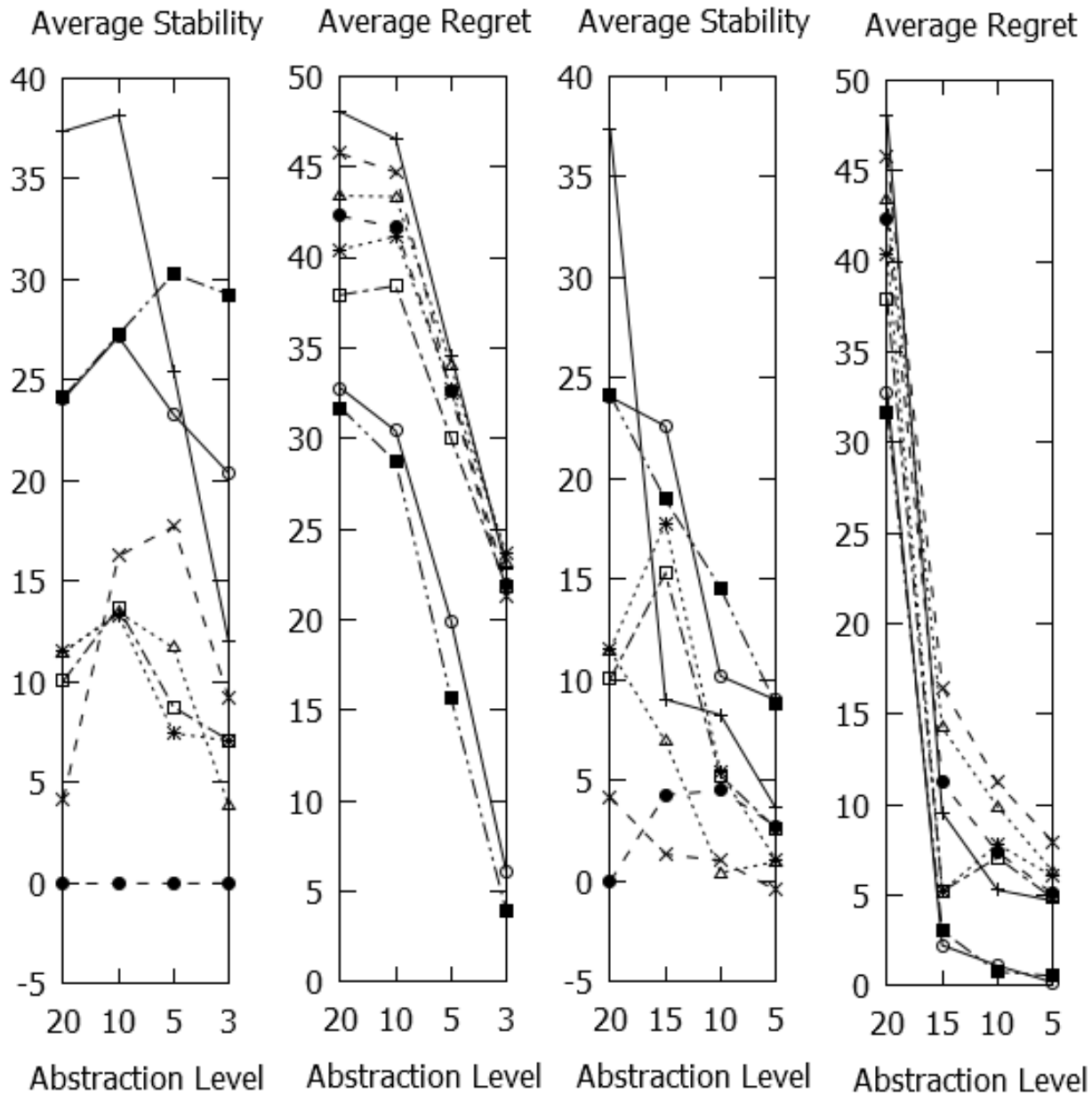
We now turn to analysis of the complete set of agents. The following results are from a fully symmetric tournament of 500 games for each class of games and 30 agents which include the best parameter settings for QRE, CH, QLK, and the other main solution concepts. We selected the best parameter settings for each of the QRE, CH, and QLK variants based on the results of the previous analysis to visualize in these results along with the other agents.

Figures 7.4 and 7.5 show the average stability and regret for the different solution methods (excluding the lines for all but the best QRE, CH, and QRE settings) when using the KMeans and TopN abstractions. We begin by noting some general trends in the results. Although we do not show the results for logical games all of the stability and regret values are very low compared to both general-sum and zero-sum games. There is also very little variation among each of the strategies. This indicates that these games are much easier to play well in, even when they are abstracted. In most cases, agents are more stable for cases without any abstraction, and less stable but converging to the same range as abstraction increases. Zero-sum games behave somewhat differently; the overall stability is worse, and



(a) TopN (b) TopN (c) K-Means (d) K-Means

Figure 7.4: General Sum Stability and Regression



(a) TopN (b) TopN (c) K-Means (d) K-Means

Figure 7.5: Zero Sum Stability and Regret

the stability of many of the agents actually improves (surprisingly) as abstraction increases.

There is no solution concept that is clearly the best across all of the settings tested. Instead, the best strategy depends heavily on both the class of games and the type of abstraction being used. The UR and Fair agents do poorly in almost all cases. The ENE and Social agents perform surprisingly well, particularly when using the TopN abstraction and in the general-sum and logical games. These agents focus on coordinating on high payoffs, so it is interesting that even these simple strategies are able to do so well even in cases with abstraction error. For the cases with KMeans abstraction and for zero-sum games, approaches close to equilibrium solvers perform better particularly MSNE and the QRE and QLK.

While this pattern of results fails to identify a clear winner among the common solution methods for how to select strategies in abstracted games, the results are still quite interesting. They clearly show that both the game class and abstraction method interact strongly with the applied solution concept, and simply finding a NE is not the best approach. There is also an opportunity to develop significantly improved solution concepts that account for these factors unlike current practices.

The final experiment we present directly compares the performance of the two different types of abstractions. In this experiment, players can use either abstraction, which leads to even more asymmetry in the abstractions players are using.

In Figure 7.6 we show the average payoffs for the different solution methods in a tournament that combines agents using both abstractions. We use only the best parameter settings for the QRE, CH, and QLK agents. This tournament was run on general-sum games, using the highest level of abstraction (from a game size of 20 down to a game size of 5). For each solution method we show the average payoffs achieved when using the two abstraction methods side by side. Interestingly, the TopN abstraction method outperforms KMeans in combination with *every one* of the solution methods, often quite substantially. The TopN abstraction plays a role in coordinating agents on high-payoff outcomes independent the solution algorithm.

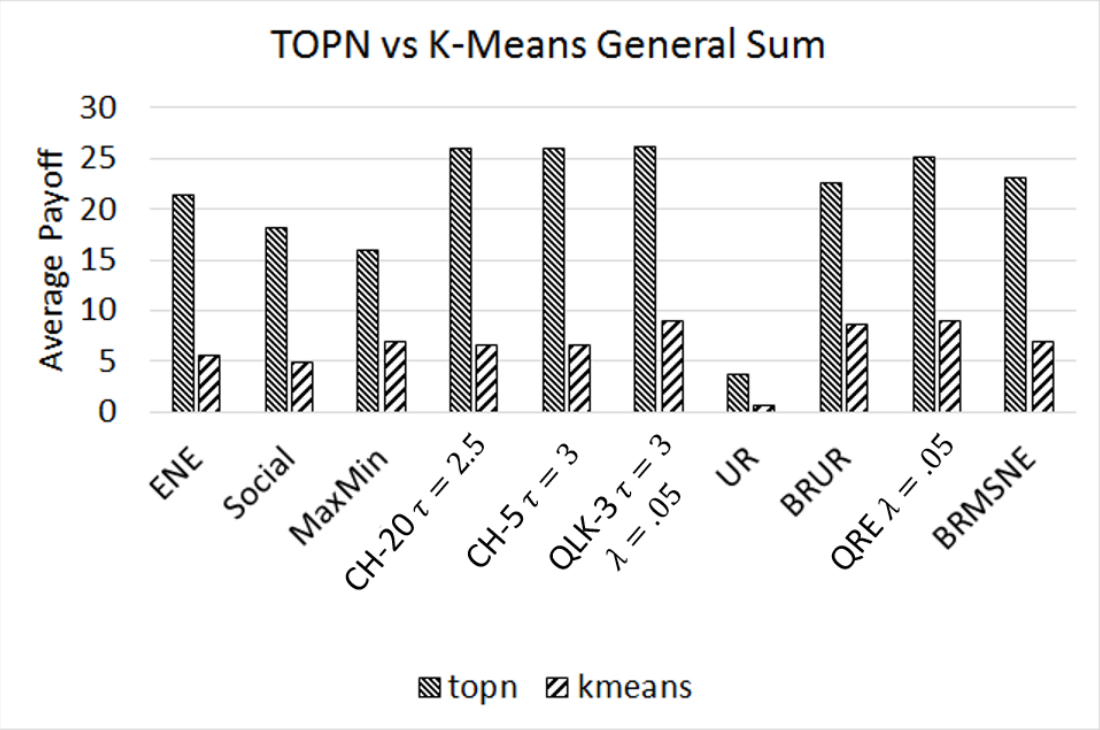


Figure 7.6: TopN vs Kmeans in General Sum - Abstraction Level 5. CH means Cognitive Hierarchy and QLK is Quantal Level-k.

7.2 Abstraction Conclusions

Our results demonstrate that using abstraction to solve games is a complex endeavor, and the type of abstraction, the solution methods used to analyze the abstracted games, and the class of games all have a strong influence on the results. Many of the strongest results using abstraction to analyze large games (e.g., Poker) have focused on zero-sum games. One of the most interesting observations from our results is that abstraction often works very differently in zero-sum games than it does general-sum games or the more structured logical games. In particular, solution methods based on finding Nash equilibrium seem to work much better in zero-sum games than they do in the other classes of games in our experiments. Another important observation from our experiments is that Nash equilibrium often does not perform well in cases where abstraction is used as part of the solution process.

It is still effective when the games are zero-sum, but in the other cases it was not robust to the introduction of error based on game abstraction.

We also found that the specific method used for generating abstractions has a strong impact on the results. One very interesting result was that TopN was sometimes able to *increase* the payoffs for the agents in comparison to the case without any abstraction. However, TopN is a symmetric abstraction when both players use it, while KMeans is asymmetric.

Some methods like the social agent performed much better when using the symmetric TopN abstraction than when using the asymmetric KMeans abstraction. This kind of interaction is very important to understand in greater depth if we are to make effective use of abstraction as part of game-theoretic analysis. Our model of abstraction meta-games provides a formal model for studying this type of interaction, and our simulations have resulted in several interesting observations that provoke many additional questions about the use of abstraction in solving games.

Chapter 8

Proposed Robust Solutions

From the experiments in Chapter 7 there were many indicators that strategies such as ENE, CH, and QLK had some of the best characteristics in the presence of abstraction. Both of the CH and QLK agents relied on the ideas from logit (Equation 2.3, assigning more probability towards actions that had higher expected utility. While utility is important, as shown in those earlier experiments it can be sensitive to noise from abstraction.

We have developed three new robust strategies, based on the ideas from CH and QLK (see Chapter 6) called Robust, Adversary, and Resilient that ar defined in Algorithms 1, 2, and 3 as well as Equations 8.1, 8.2, and 8.3 respectively. The intuitions for how these agents function are as follows:

- Robust - Opponent could play any strategy so try to minimize a perceived regret against opponent actions as input to the logit function then find a best response.
- Adversary - Play best response to an opponent who is applying more proportional weight to actions with higher payoffs as inputs to the logit function.
- Resilient - Given an assumed opponent strategy and an interpolation parameter, assign a score per each action interpolating between the expected payoff and a hypothetical nemesis, then use those values in logit to recompute expected payoffs and select the action with the highest value.

For simplicity, these agents are also Level-0, meaning that they do not perform an iterated best response nor noisy best response like Quantal Level-k. While somewhat limiting, it also provides the best opportunity to quickly gauge their behaviors without the

Table 8.1: Example Game

	D	E	F
A	2,2	5,4	7,2
B	0,5	4,0	3,5
C	1,2	2,3	2,3

impact of best response dynamics. It is also important to note that to keep the models as simple as possible, each agent will assume initial uniform random strategies unless otherwise specified. Because each uses a variation of logit (Equation 2.3), λ has to be given as an input parameter. When $\lambda = 0$ then Robust and Adversary will essentially be performing as a BRUR agent since the action values inside the function would be zero. The agents appear to have a less obvious behavior as λ is increased compared to logit. Resilient also requires an additional parameter α. This value is meant to represent how much to lean into the agent's own utility, or a hypothetical Nemesis out to get Resilient. Because of this, Resilent has many moving parts and identifying strong values for λ and α is quite a challenge. This is also why these agents were kept as Level-0 because adding τ and k would have greatly increased the possible test space.

Algorithm 1 Robust Strategy

Assume opponent is trying to punish current player

Compute best response each my opponent action save my resulting payoffs store maximum per action m_a

Identify the maximum overall payoff $M = \max(m_a)$

Compute a regret values per action $r_i = M - m_a$

Replace $EU_i a$ in the logit equation with r_a to compute the probability for distribution S

Identify Pure Strategy Best Response to S

$$S_i = \text{BR}(S_{-i}, G) \text{ s.t. } S_{-ia} = \frac{\exp(\lambda r_a)}{\sum_k \exp(\lambda r_k)} \ , \ r_a = M - EU_{ia} \ , \ M = \max_{a \in k}(EU_{ia}) \tag{8.1}$$

Algorithm 2 Adversarial Strategy

Assume you are using uniform random, compute your expected payoff for each opponent action EU_{-ia}

Use EU_{-ia} as the input for the logit function to compute opponent strategy S

Identify Pure Strategy Best Response to S

$$S_i = \text{BR}(S_{-i}, G) \ \text{ s.t. } \ S_{-ia} = \frac{\exp(\lambda EU_{-ia})}{\sum_k \exp(\lambda EU_{-ik})} \tag{8.2}$$

Algorithm 3 Resilient Strategy

Given an opponent strategy (assume uniform random if none is given)

Given α parameter between 0 and 1 where 1 leans to expected payoffs and 0 to the hypothetical nemesis

Find the highest payoff value in the game called p_{max}

Compute $v1$ for each action as the expected payoff

Compute $v2$ as $p_{max} - v1$ per action

Assign $v3$ for each action as $\alpha * v1 + (1 - \alpha) * v2$

Use $v3$ in place of EU_i as the input for the logit function to compute strategy S_i

Calculate the expected payoffs per action using S_i and the given opponent strategy S_{-i}, then compute the Quantal Best Response

Table 8.2: Robust Applied to Table 8.1

	D	E	F
m_i	2	5	7
r_i	5	2	0

Table 8.3: Adversary Applied to Table 8.1

	D	E	F
EU_i	3	$2.\bar{3}$	$3.\bar{3}$

$$
\begin{aligned}
&S_i = \text{QBR}(S_i, S_{-i}, G, \lambda) \text{ s.t.} \\
&\quad S_{ia} = \frac{\exp(\lambda v3_a)}{\sum_k \exp(\lambda v3_k)}, \\
&\quad v3_a = v1_a \times \alpha + v2_a \times (1-\alpha), \\
&\quad v1_a = EU_{ia}\ ,\ v2_a = p_{max} - v1_a, \\
&\quad p_{max} = \max(U_i)
\end{aligned}
\tag{8.3}
$$

Applying the Robust Strategy (Algorithm 1) as Row Player in the game given in Table 8.1 gives the results in Table 8.2. Applying r_i to logit with $\lambda = 0.5$ gives the opponent strategy of $S = (0.766, 0.171, 0.063)$ then the pure strategy best response for the Row Player is to choose action A. Applying the Adversary Strategy (Algorithm 2) as Row Player in the game given in Table 8.1 gives the results in Table 8.3. Applying EU_i to logit with $\lambda = 0.5$ gives the opponent strategy of $S = (0.345, 0.247, 0.408)$ then the pure strategy best response for the Row Player is to choose action B.

Note that these three agents are part of a larger family of novel strategies for solving games based on the logit function and QLK that the author is proposing. There exist many more variations on this approach with other ways of calculating the action values that could lead to stronger play under uncertainty. They also coincide with the aligning to Definition 3.1 yet the author is aware that there are domain areas that require other types

of robustness to which these new agents may not match. Solving games using Resilient uses a process similar to the other Robust and Adversary only using Algorithm 3 instead.

Chapter 9

Robust Agent Experiments

The experiments in Chapter 7 evaluated the performance of common strategies in the presence of abstraction. Yet, the manner of the noise injected into the simulation showed more an impact of the abstraction on the agents, rather than the ability of the agents to show the resilience of their strategies. The following set of experiments adds more controlled noise, instead of abstraction, and tests the new and promising class of robust agents developed in Section 3.3.

9.1 Experiment Setup

We evaluate these new robust family of agents in tournaments, similar to those of Chapter 7 with a meta-game approach (examples of which can be found in Tables 3.1 and 3.2) with the formal model in Section 5.1. The games were kept small, with few actions (3x3), to ensure that drastic changes between strategies would be more apparent. The process of abstraction added too much uncertainty into the games to clearly separate the impacts resulting from abstraction and what was the based on the performance of the agents. Therefore the following initial exploratory experiments were evaluated without abstraction.

Instead, we introduced random noise in the asymmetric games, for the creation of the meta-games, that we gave to agents in the tournament. Then, instead of shrinking the size of the meta-games, we raised the amount of noise with each new tournament. This random noise was seeded for repeatability. The tournament generated new two versions of each game by altering every outcome given a noise parameter. To alter an outcome, it makes

a call to `nextGaussian()` and multiplies the result by the noise parameter. That creates a change amount that is used to alter one payoff, then the other by the negation of that change. This also ensures that zero-sum games remain zero-sum. The process also verifies that payoffs are still capped within the valid payoff range (for this experiment between -100,100 for zero sum and 0,100 for general) so it is not apparent that the game has been changed.

9.2 Baseline Agents

To evaluate the new Robust, Adversary, and Resilient agents, it became necessary to identify baseline agents to test against. The baseline agents were chosen to demonstrate the characteristics of 'robustness' from Definition 3.1. As such, Uniform Random, Epsilon Nash Equilibrium, MaxMin, and MinMax (also called Punisher) were selected. Additionally, another agent was created to generate the absolute worst called Nemesis.

Baseline Agents:

- Uniform Random never changes its strategy, no matter the uncertainty, and therefore sets a level that all other agents should do at least as well against. When a smart agent performs worse than UR (on average), it is an indicator that the agent is being strongly impacted by the uncertainty or that something else is wrong.

- ENE has no expectation to do well against non-ENE opponents. Instead it is meant to demonstrate a level of performance comparison, meaning an agent's expected utility relative to the ENE. It will also be an agent that is greatly impacted by noise and obfuscation, even against another ENE opponent as the changes created by noise are asymmetric.

- The MaxMin baseline agent is a worst-case expectant strategy that finds the minimum payoff for each action then selects the largest to guarantee at least that amount. When games are obfuscated this can cause the agent to select an action that it otherwise

would not have by changing what are the apparent minimums per action. Depending on the amount of noise, this might lead to minor, or major, changes in decided actions.

- Punish (aka MinMax) is an agent that cares not for its own payoff but rather tries to limit the payoff of the opponent. When the game is zero sum then Punish is equivalent to MaxMin. This agent is meant to demonstrate the resiliency of the agent in general sum games and how well it can perform when the opponent is actively trying to work against it. This agent can perform worse than Uniform Random because it does not consider its own payoff.

- Nemesis executes after the all other strategies have been computed and then finds the strategy that minimizes the performance of the tested agent knowing the full game and the selected strategy. It is the worst-case agent that represents an adversary with maximum capabilities (beyond those of a realistic opponent in some cases).

Results from several experiments are plotted in Figures 9.1 and 9.2. These histograms show a given agent's average performance against the four given baseline agents. A few behaviors to note, the Punisher agent earned very low payoffs because it is only interesting in lowering the opponent utility. Also the ENE agent performs the well in General Sum games when it competes against itself. This should come as no surprise given that the equilibrium strategy maximizes its own payoff assuming the opponent is also trying to maximize theirs. The trouble for this agent though is that the assumption that it makes, does not always hold, and its performance suffers when the opponent is not playing according to the equilibrium strategy. This behavior is shown in its performance against the other three baseline agents. Additionally, in Zero Sum games (Figure 9.2) the ENE versus ENE resulted in a near zero strategy and a poor showing against the MaxMin and Punisher agents.

The newly developed Robust, Adversary, and Resilient agents had some of the best overall showings usually on par or better performance compared to the baseline performances. Robust in particular had much higher average utility against the Punisher agent

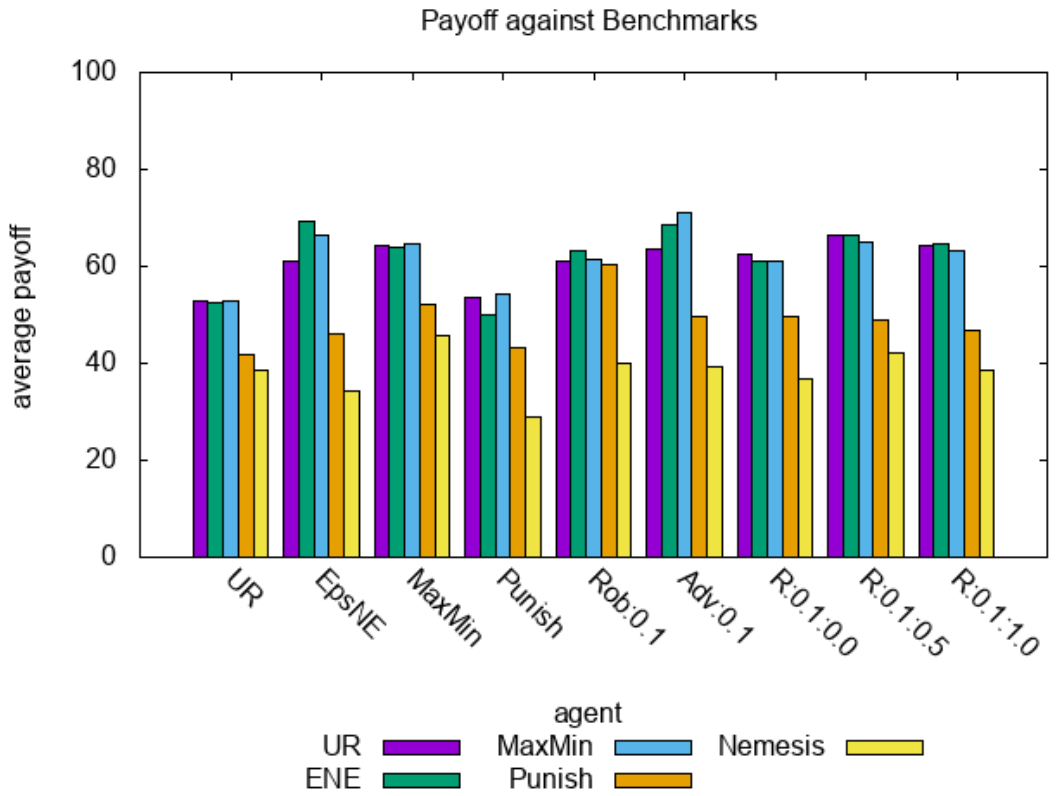


Figure 9.1: Robustness Testing in General Sum Games

than did any of the baselines. Similarly, the Adversary agent was able to perform well above average when playing against the MaxMin baseline. Then in Zero Sum Games the Robust and Adversary agents showed clearly higher performances than the standard agents particularly the Robust agent against the MaxMin/Punisher strategy.

Without uncertainty, the majority of agents earned near-zero overall payoffs against ENE and MaxMin/Punisher while performing quite positively against UR and overwhelmingly negative against their Nemesis; all of which was to be expected. Some of the new tested agents like Adversary $\lambda = 0.1$ and Resilient $\lambda = 0.1$ $\alpha = 0$ had lower performances overall in zero sum but improved performance in general sum. The rest of the tests results are in general sum. The improved behavior of the newly developed robust agents becomes evident when noise is introduced into the tournaments as demonstrated in Figures 9.3,

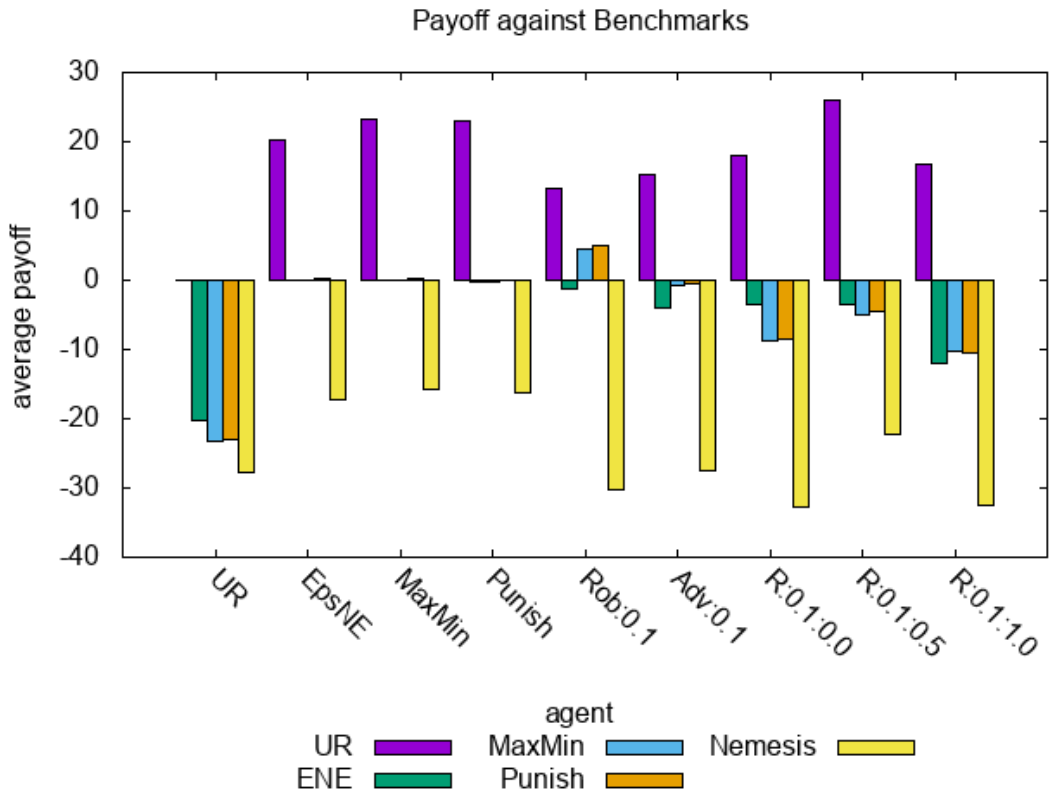


Figure 9.2: Robustness Testing in Zero Sum Games

9.3, 9.4, and 9.5.

When running these tournaments, the level of possible noise started at none for a baseline measure, then increased with each repeated tournament. At the end of each tournament, the results of the expected payoffs are evaluated on the original un-obfuscated version of the game. This demonstrates the impacts of the noise on the agent's solution quality. The MaxMin agent is expected to perform well against Nemesis when there is no uncertainty, because it expects a worst-case scenario and selects the action which performs best against it. When there is uncertainty in the game MaxMin still has the possibility to do well although it my not longer find the optimal MaxMin strategy.

Figure 9.3 shows how against a UR opponent with increasing uncertainty all agents appeared to have an overall decline in average expected payoff while strategies like Re-

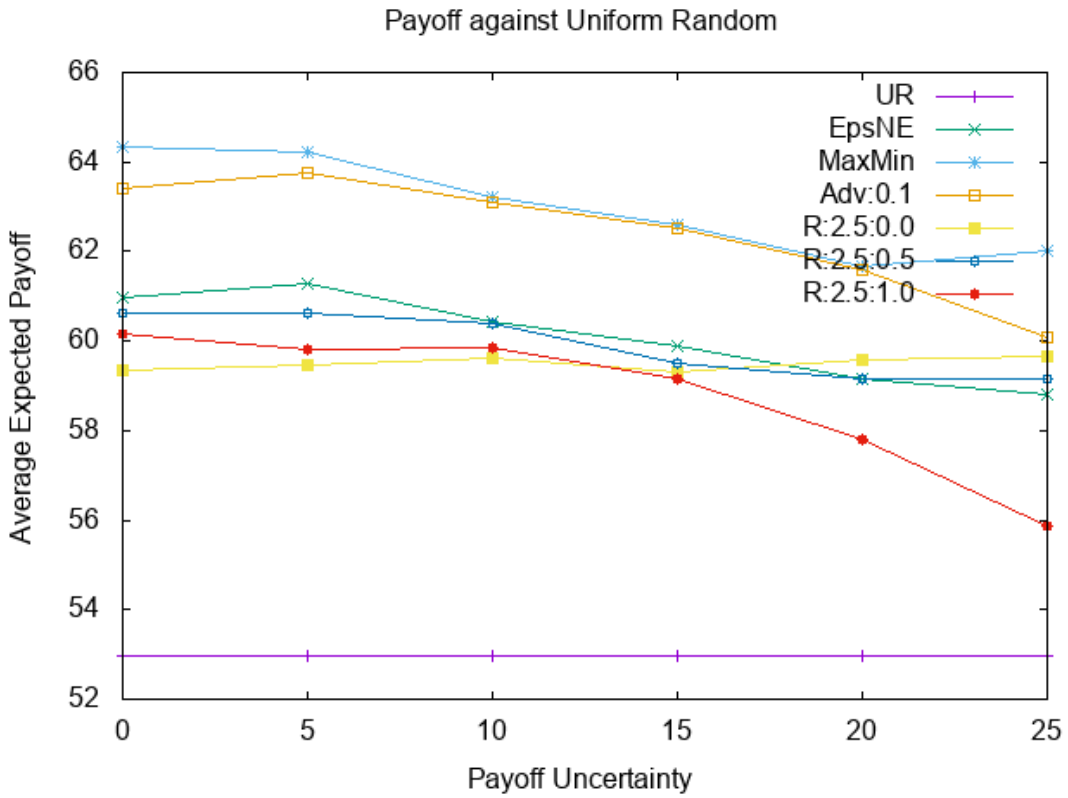


Figure 9.3: Payoff against UR with increasing uncertainty

silient:2.5:1.0 ($\lambda = 2.5$ and$\alpha = 1$) did not do as well as hoped but still performed better than UR against UR. The issue with an agent like Resilient is that there are a lot of moving parts and finding optimal (or even good) parameter settings would take a very long time. Should the agent use the iterated best response dynamics, like in QLK, it would be expected to perform better. Amazingly though, the Adversary strategy was able to earn payoffs similar to MaxMin even at higher levels of uncertainty

Meanwhile the same tournaments, but against ENE, tells a similar story represented in Figure 9.4. These results show how the noise impacts ENE's choices even having UR earn increasing payoffs although its strategy never changed no matter the uncertainty. With some uncertainty the Adversary:0.1 agent ends up usually out-performing ENE against ENE, because of how sensitive ENE is to noise.

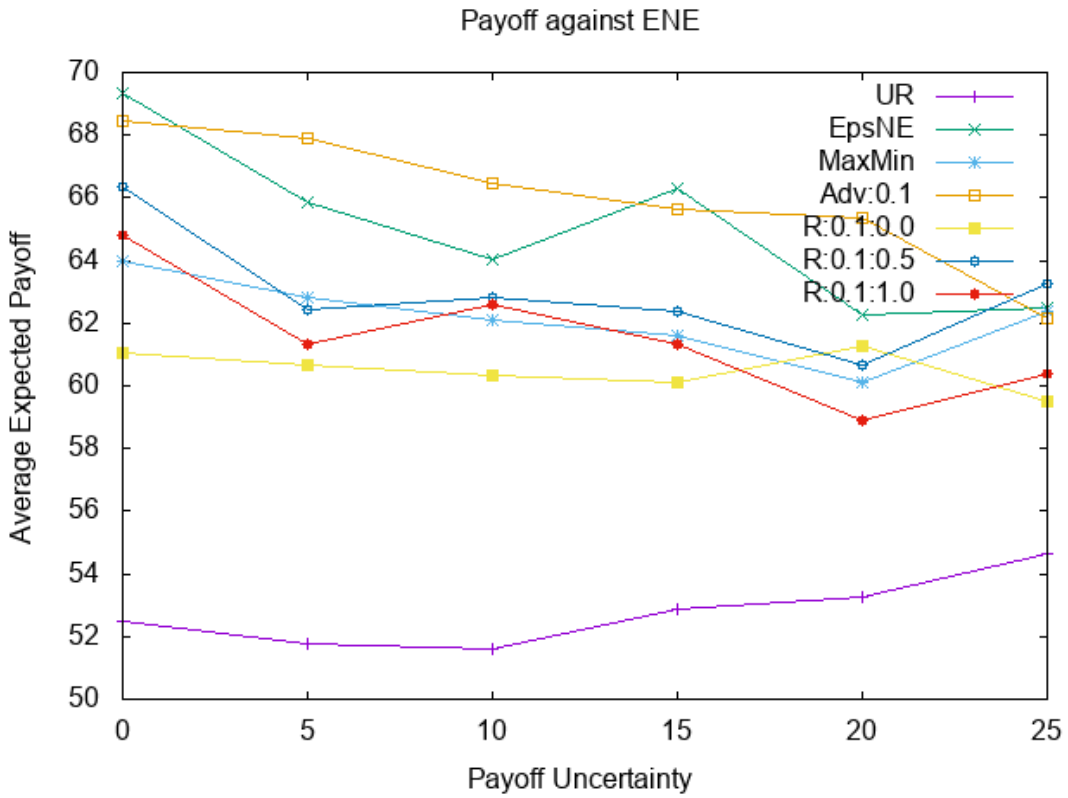


Figure 9.4: Payoff against ENE with increasing uncertainty

Against the Nemesis (Figure 9.5), all agents earned average payoffs in the 30's and 40's compared to the 60's and 70's of the earlier tournaments. Meanwhile the behavior also changes with MaxMin, UR, and R:0.1:0.5 hovering around each other with the largest amount of uncertainty added. Even an agent like Adv:01 and R:0.1:1.0 perform similar to UR agaisnt their worst-case Nemesis when there is no or little uncertainty in the tournament. Agents like ENE that are highly sensitive are are at the bottom of these sets of agents in the tournament.

To demonstrate the impact of playing against an equilibrium agent and the Nemesis, Figure 9.6 shows what happens to a strategy's expected utility in a tournament against a mix between ENE and Nemesis by linearly interpolating the two pure strategies to develop mixed strategies. Agents like Punish and UR do very poorly (as expected) because they

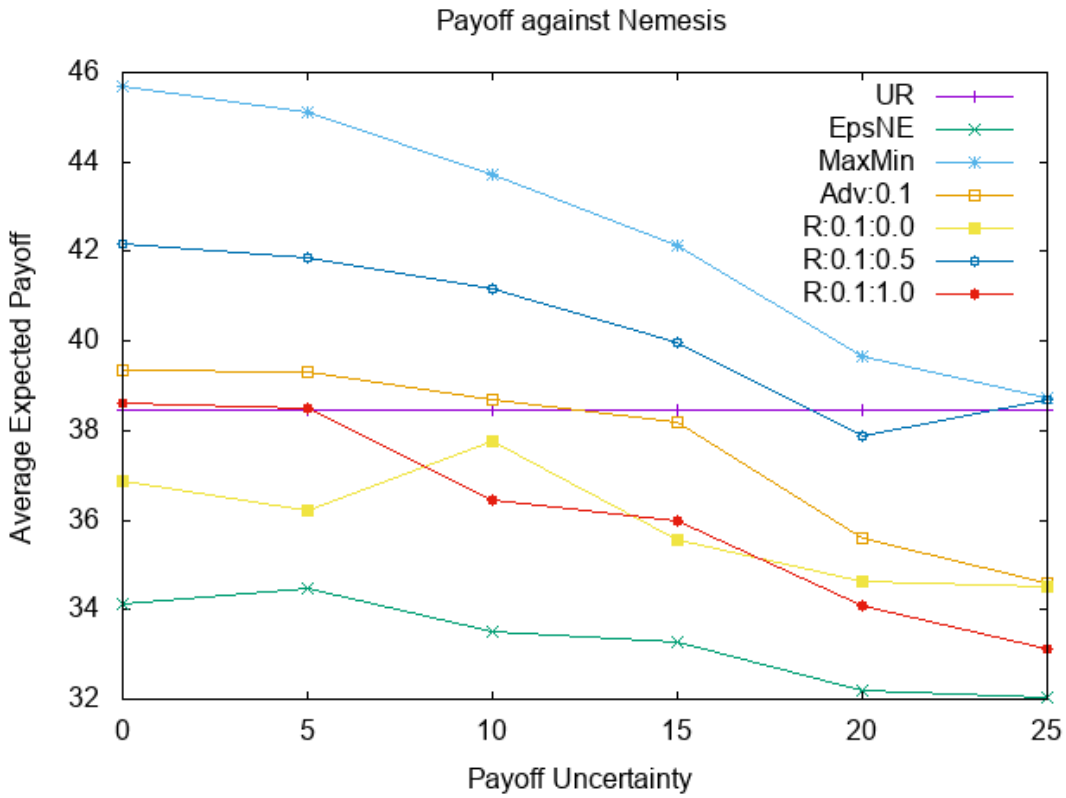


Figure 9.5: Payoff against Nemesis with increasing uncertainty

do not evaluate their own payoffs. Also expected is the high value for ENE against ENE followed by the steepest decline as the strategy tends to pure Nemesis.

Because Figure 9.6 has a lot going on it can be difficult to decipher. A simpler version is presented in Figure 9.7 that only evaluates the performance of the Adversary agent. Notice that on the left side, when playing against ENE, Adversary earns near ENE level payoff. With a mixture of the Nemesis though ENE actually begins to receive less expected utility than Adversary until at the very right where it earns even less than Uniform Random against Nemesis. Meanwhile Adversary against Nemesis did better than Uniform Random which is saying a lot. The figure demonstrates how robust to the worst-case Adversary is while still performing near equilibrium levels against the equilibrium agent.

To truly understand how the other new agents, Robust and Resilient, would behave

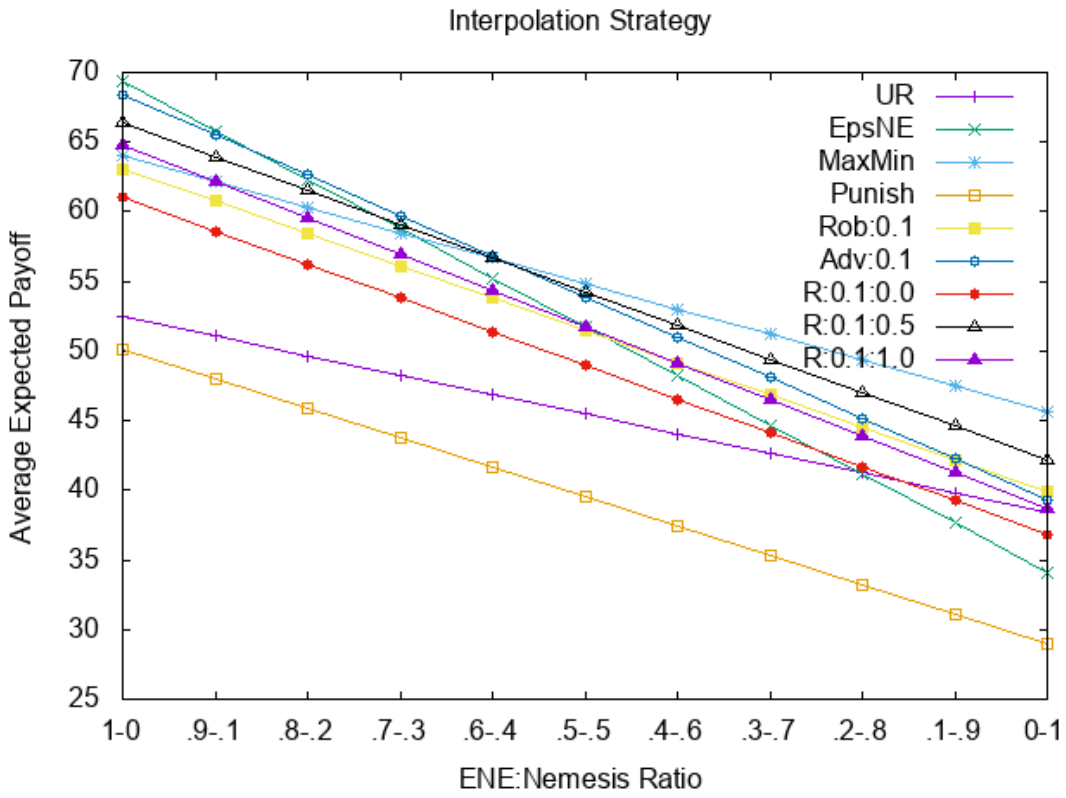


Figure 9.6: Response to interpolation between ENE and Nemesis

requires a lot more testing for many values of λ and α. Still, that may be moot since perhaps the biggest performance boost could come in the raising of the agents from Level-0 to Level-k. In any event, further testing is needed to identify good parameter settings for all three agents.

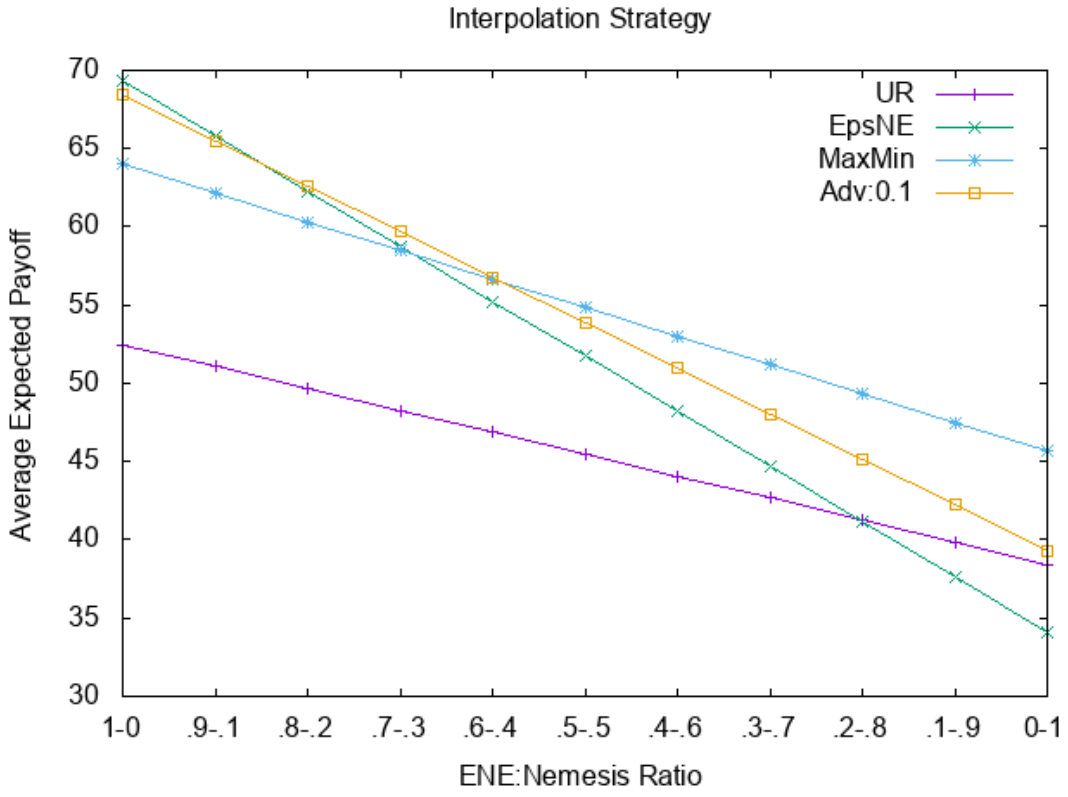


Figure 9.7: Response to interpolation between ENE and Nemesis

Chapter 10

Conclusions and Future Work

This dissertation set out to answer the questions in Section 1.3 regarding the use of abstraction in games and the selection of robust strategies for playing under uncertainty (Q2). We classified two different types of abstraction (action merging and action elimination) and developed two abstraction algorithms (K-means and TopN). We created a formal model (meta-games) for evaluating game-playing agents in the presence of uncertainty including abstraction (Q1). We then evaluated interactions between abstraction and strategy selection (Q3). Then, additionally we defined robustness and then developed new agents focused on robustness (Q4). Then we demonstrated the behavior of these agents under uncertainty by isolating the noise in games by using obfuscation instead of abstraction (Q4).

Detailed results of the experiments are given in Chapters 7 and 9. There are of course many more experiments that can be run to identify how the new robust family of agents behave with abstraction, yet before that can be accomplished it requires further exploration of λ and α values as they are sensitive to the payoff range and possibly the size of the game. Finding optimal values for one type of game may likely will not be optimal for another type of game. One aspect of further research could be finding a method for determining initial parameter settings or finding an interval that contains the optimal value in the presence of uncertainty.

A distinct nuance in the tournaments that the reader may have missed is the agents were unaware that there was noise in the game. One could imagine an agent that knows that a game has some uncertainty and how much each action could be changed by, but not which actions were changed. Consider a real-life example of an Emergency Response Coordinator modeling resource allocation immediately after a disaster as a game. The coordinator might

have preliminary intelligence with some amount of uncertainty as reporting changes. This agent would still want to develop a robust strategic response while being aware that there is noise in the data.

Another aspect of further research could relate to the very nature of the uncertainty itself. The meta-game model has so far used structured obfuscation through abstraction or random noise but consider controlled noise by a nemesis. An approach to doing this has been attempted by author in a Stackelberg Security Game scenario using quadratic programming with inconclusive results so far.

Yet another extension to the family of robust agents developed in this research relates to the Resilient agent. It assumes that the opponent strategy is given, or otherwise uniform random, but it could instead incorporate learnings from the QLK agent, and perform an iterated Quantal Best Response instead of its current simplistic approach. Of course, this adds another parameter to define how many levels to iterate and there may not be a guarantee of convergence, yet this appears as a natural next step to hopefully better the agent.

The major elephant in the room is abstraction, and the many other possible ways to shrink a game for strategy feasibility purposes that the author has not evaluated. Even with the use of machine learning to help reduce search spaces, the act of learning still requires a lot of time and memory, whereas abstraction is designed to be a fast way to shrink a game for tractability. The trade off is noise that is introduced which leads back to the need for strategies that are robust and the importance of this work. There may be ways to combine the strengths of both abstraction and learning. The future appears bright abstraction in games and robust strategies under uncertainty.

References

[1] Michele Aghassi and Dimitris Bertsimas. Robust game theory. *Mathematical Programming*, 107(1):231–273, 2006.

[2] Luca De Alfaro. Three-valued abstractions of games: Uncertainty, but with precision. In *Proc. 19th Annu. IEEE Symp. Found. Comput. Sci. Log. Comput. Sci. 2004.*, pages 345–354, 2004.

[3] Suzanne H. Alonzo and Barry Sinervo. Mate choice games, context-dependent good genes, and genetic cycles in the side-blotched lizard, Uta stansburiana. *Behav. Ecol. Sociobiol.*, 49(2-3):176–186, 2001.

[4] Tansu Alpcan and Tamer Basar. *Network security: A decision and game-theoretic approach*. Cambridge University Press, 2010.

[5] Nolan Bard, Michael Johanson, and Michael Bowling. Asymmetric abstractions for adversarial settings. In *Proc. 2014 Int. . . .*, pages 501–508. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[6] Steven Borowiec and Tracey Lien. AlphaGo beats human Go champ in milestone for artificial intelligence. *Los Angeles Times*, Mar 2016.

[7] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold ' em poker is solved. *Science (80-.).*, 347(6218):145–149, Jan 2015.

[8] Jennings Brown. Go Champion Retires After Realizing AI Is 'an Entity That Cannot Be Defeated', 2019.

[9] Noam Brown, Sam Ganzfried, and Tuomas Sandholm. Hierarchical Abstraction, Distributed Equilibrium Computation, and Post-Processing, with Application to a Champion No-Limit Texas Hold'em Agent. In *Int. Conf. Auton. Agents Multiagent Syst.*,

pages 7–15, Istanbul, Turkey, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[10] Matt Burgess. Google's DeepMind wins historic Go contest 4-1. *Wired UK*, Mar 2016.

[11] Shay Bushinsky. Deus Ex Machina-A Higher Creative Species in the Game of Chess. *AI Mag.*, 30(3):63–70, 2009.

[12] Sam Byford. Google's alphago ai beats lee se-dol again to win go series 4-1. `http://www.theverge.com/2016/3/15/11213518/alphago-deepmind-go-match-5-result`, 2016. Accessed: 2017-03-01.

[13] Sam Byford. AlphaGo retires from competitive Go after defeating world number one 3-0 - The Verge. *The Verge*, 2017.

[14] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu. Deep Blue. *Artif. Intell.*, 134(1-2):57–83, 2002.

[15] Shih-Fen Cheng and Michael P. Wellman. Iterated Weaker-than-Weak Dominance. In *Proc. Twent. Int. Jt. Conf. Artif. Intell. IJCAI-07*, pages 1233–1238, 2007.

[16] Vincent Conitzer and Tuomas Sandholm. A technique for reducing normal-form games to compute a Nash equilibrium. In *Proc. fifth Int. Jt. Conf. Auton. agents multiagent Syst. - AAMAS '06*, pages 537–544, New York, New York, USA, 2006. ACM Press.

[17] Neil Connor. Google AlphaGo 'can't beat me' says China Go grandmaster. *Telegraph*, Mar 2016.

[18] Emily Conover. Winning against a computer isn't in the cards for poker pros. *Sci. News*, Mar 2017.

[19] Rémi Coulom. Crazy stone. `https://www.remi-coulom.fr/CrazyStone/`, 2016. Accessed: 2017-03-01.

[20] Giovanni Paolo Crespi, Davide Radi, and Matteo Rocca. Robust games: theory and application to a cournot duopoly model. *Decisions in Economics and Finance*, 40(1):177–198, 2017.

[21] Pedro Dal Bó and Guillaume R Fréchette. The evolution of cooperation in infinitely repeated games: Experimental evidence. *American Economic Review*, 101(1):411–29, 2011.

[22] DeepMind. Deepmindyoutube. `https://www.youtube.com/channel/UCP7jMXSY2xbc3KCAE0MHQ-A/videos`, 2017. Accessed: 2017-03-01.

[23] Nathan Ensmenger. Is chess the drosophila of artificial intelligence? A social history of an algorithm. *Soc. Stud. Sci.*, 42(1):5–30, 2012.

[24] Feng-Hsiung Hsu. IBM's Deep Blue Chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.

[25] David B. Fogel, Timothy J. Hays, Sarah L. Hahn, and James Quon. A self-learning evolutionary chess program. *Proc. IEEE*, 92(12):1947–1954, 2004.

[26] Sam Ganzfried. My Reflections on the First Man vs. Machine No-Limit Texas Hold 'em Competition, 2015.

[27] Sam Ganzfried and Tuomas Sandholm. Safe opponent exploitation. *ACM Transactions on Economics and Computation (TEAC)*, 3(2):1–28, 2015.

[28] Sam Ganzfried, Tuomas Sandholm, and Kevin Waugh. Strategy purification and thresholding: Effective non-equilibrium approaches for playing large games. In *Proc. 11th Int. Conf. Auton. Agents Multiagent Syst. AAMAS '12*, volume 2, pages 871–878, 2012.

[29] Sylvian Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions. *Commun. ACM*, pages 106–113, 2012.

[30] Elizabeth Gibney. Go players react to computer defeat. *Nature*, Jan 2016.

[31] Elizabeth Gibney. Google AI algorithm masters ancient game of Go. *Nature*, 529(7587):445–446, Jan 2016.

[32] Elizabeth Gibney. Google reveals secret test of AI bot to beat top Go players. *Nature*, 541(7636):142–142, Jan 2017.

[33] Elizabeth Gibney. How rival bots battled their way to poker supremacy. *Nature*, 543(7644):160–161, 2017.

[34] Andrew Gilpin and Tuomas Sandholm. Expectation-Based Versus Potential-Aware Automated Abstraction in Imperfect Information Games: An Experimental Comparison Using Poker. In *Proc. Twenty-Third AAAI Conf. Artif. Intell.*, pages 1454–1457, 2008.

[35] Fernand Gobet and Philippe Chassy. Expertise and intuition: A tale of three theories. *Minds Mach.*, 19(2):151–180, 2009.

[36] Johannes Heinrich and David Silver. Smooth UCT search in computer poker. In *IJCAI Int. Jt. Conf. Artif. Intell.*, pages 554–560, 2015.

[37] Sean D Holcomb, William K Porter, Shaun V Ault, Guifen Mao, and Jin Wang. Overview on deepmind and its alphago zero ai. In *Proceedings of the 2018 international conference on big data and education*, pages 67–71, 2018.

[38] Constance Holden. Draw in Bahrain. *Science (80-.).*, 298(5595):959, 2002.

[39] Jeremy Hsu. Poker Pros School Computer on No-Limit Texas Hold'em. *IEEE Spectr.*, May 2015.

[40] Jeremy Hsu. AI Decisively Defeats Human Poker Players - IEEE Spectrum. *IEEE Spectr.*, Jan 2017.

[41] Shih-Chieh Huang and Martin Müller. Investigating the Limits of Monte-Carlo Tree Search Methods in Computer Go. In *Comput. Games*, volume 4630, pages 39–48. Springer, Cham, 2014.

[42] Zheping Huang. Google's AlphaGo AI secretively won more than 50 straight games against the world's top Go players. *Quartz Media*, Jan 2017.

[43] Zheping Huang. The world's best Go player says he still has "one last move" to defeat Google's AlphaGo AI. *Quartz Media*, pages 1–3, Jan 2017.

[44] Peter J. Huber and Elvezio M. Ronchetti. *Robust Statistics*. Wiley, Hoboken, NJ, 2009.

[45] IBM. Icons of progress - deep blue. `http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`, 2016. Accessed: 2017-03-01.

[46] Jing. Crazy stone computer go program defeats ishida yoshio 9 dan with 4 stones. `https://gogameguru.com/crazy-stone-computer-go-ishida-yoshio-4-stones/`, 2013. Accessed: 2017-03-01.

[47] Michael Johanson, Nolan Bard, Neil Burch, and Michael Bowling. Finding Optimal Abstract Strategies in Extensive-Form Games. In *AAAI'12 Proc. Twenty-Sixth AAAI Conf. Artif. Intell.*, pages 1371–1379. AAAI Press, 2012.

[48] Michael Johanson and Neil Burch. Evaluating state-space abstractions in extensive-form games. In *Proc. 12th Int. Conf. Auton. Agents Multiagent Syst.*, pages 6–10, 2013.

[49] Garry Kasparov. The Chess Master and the Computer. *New York Rev. Books*, 57(2):1–6, 2010.

[50] Jonathan Katz. Bridging Game Theory and Cryptography: Recent Results and Future Directions. In *Theory Cryptogr.*, volume 4948, pages 251–272. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[51] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing Optimal Randomized Resource Allocations for Massive Security Games. In *Proc. 8th Int. Conf. Auton. Agents Multiagent Syst.*, volume 1, pages 689–696, Budapest, Hungary, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[52] Christopher Kiekintveld, Jason Miller, Patrick R. Jordan, Lee F. Callender, and Michael P. Wellman. Forecasting market prices in a supply chain game. *Electron. Commer. Res. Appl.*, 8(2):63–77, 2009.

[53] Christopher Kiekintveld and Michael P. Wellman. Selecting strategies using empirical game models: an experimental analysis of meta-strategies. In *Proc. 7th Int. Jt. Conf. Auton. agents multiagent Syst. AAMAS '08*, volume 2, pages 1095–1101, 2008.

[54] Richard Klíma, Christopher Kiekintveld, and Viliam Lisý. Online Learning Methods for Border Patrol Resource Allocation. In *Conf. Decis. Game Theory Secur.*, pages 340–349. Springer International Publishing, 2014.

[55] Ben Kloester. Can alphago defeat lee sedol? `https://gogameguru.com/can-alphago-defeat-lee-sedol/`, 2016. Accessed: 2017-03-01.

[56] Alan Levinovitz. The Mystery of Go, the Ancient Game That Computers Still Can't Win. *Wired*, pages 1–12, 2014.

[57] Kevin Leyton-Brown and Yoav Shoham. *Essentials of game theory*, volume 2. Morgan & Claypool Publishers, 2008.

[58] Jiawei Li and Graham Kendall. The effect of memory size on the evolutionary stability of strategies in iterated prisoner's dilemma. *IEEE Trans. Evol. Comput.*, 18(6):819–826, 2013.

[59] Richard D. McKelvey, Andrew M. McLennan, and Theodore L. Turocy. Gambit: Software tools for game theory, version 13.1.2. `http://www.gambit-project.org/`, 2014. Accessed: 2014-08-01.

[60] Richard D. McKelvey and Thomas R. Palfrey. Quantal Response Equilibria for Normal Form Games. *Games Econ. Behav.*, 10(1):6–38, Jul 1995.

[61] Merriam-Webster. Robust, 2017.

[62] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker. *Science*, 356(6337):508–513, 2017.

[63] David Z. Morris. Google's AlphaGo AI Runs the Table on Asia's Go Champs. *Fortune/Tech*, Jan 2017.

[64] Eugene Nudelman, Jennifer Wortman, Yoav Shoham, and Kevin Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS*, volume 4, pages 880–887, 2004.

[65] David Ormerod. Alphago defeats lee sedol 4–1 in google deepmind challenge match. `https://gogameguru.com/alphago-defeats-lee-sedol-4-1/`, 2016. Accessed: 2017-03-01.

[66] Ben Popper. AI has definitively bested humans at poker. *The Verge*, pages 1–3, Jan 2017.

[67] Ben Popper. This AI will battle poker pros for $200,000 in prizes. *The Verge*, Jan 2017.

[68] Matthew Rabin. Incorporating Fairness into Game Theory and Economics. *Am. Econ. Rev.*, 83(5):1281–1302, 1993.

[69] Walid Saad, Zhu Han, H. Vincent Poor, and Tamer Basar. Game-Theoretic Methods for the Smart Grid: An Overview of Microgrid Systems, Demand-Side Management, and Smart Grid Communications. *IEEE Signal Process. Mag.*, 29(5):86–105, 2012.

[70] Tuomas Sandholm. The state of solving large incomplete-information games, and application to poker. *AI Mag.*, 31(4):13–32, 2010.

[71] Tuomas Sandholm. Abstraction for solving large incomplete-information games. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[72] Tuomas Sandholm and Satinder Singh. Lossy stochastic game abstraction with bounds. In *Proc. 13th ACM Conf. Electron. Commer. - EC*, volume 1, pages 880–897, New York, New York, USA, 2012. ACM Press.

[73] Jonathan Schaeffer. The games computers (and people) play. *Adv. Comput.*, 52(C):189–266, 2000.

[74] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[75] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK the world man-machine checkers champion. *AI Mag.*, 17(1):21–29, 1996.

[76] David Schnizlein, Michael Bowling, and Duane Szafron. Probabilistic State Translation in Extensive Games with Large Action Sets. In *Proc. 21st Int. jont Conf. Artifical Intell. IJCAI'09.*, pages 278–284, 2009.

[77] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, February 2009.

[78] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[79] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[80] Smithsonian. Computers & business machines. `http://americanhistory.si.edu/collections/subjects/computers-business-machines`, 2016. Accessed: 2017-03-01.

[81] Yoon Sung-won. Lee Se-dol shows AlphaGo beatable. *The Korea Times*, Mar 2016.

[82] Moshe Tennenholtz. Competitive safety analysis: Robust decision-making in multi-agent systems. *Journal of Artificial Intelligence Research*, 17:363–378, 2002.

[83] H. Jaap Van den Herik, Jos W.H.M. Uiterwijk, and Jack Van Rijswijck. Games solved: Now and in the future. *Artif. Intell.*, 134(1-2):277–311, 2002.

[84] Kevin Waugh, Nolan Bard, and Michael Bowling. Strategy grafting in extensive games. In *Adv. Neural Inf. Process. Syst. Found. NIPS'2009*, pages 1–9, 2009.

[85] Kevin Waugh, David Schnizlein, Michael Bowling, and Duane Szafron. Abstraction Pathologies in Extensive Games. In *Proc. 8th Int. Conf. Auton. Agents Multiagent*

Syst., volume 2, pages 781–788, Budapest, Hungary, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[86] James R. Wright and Kevin Leyton-Brown. Predicting Human Behavior in Unrepeated, Simultaneous-Move Games. *arXiv.org*, pages 1–48, 2014.

Curriculum Vitae

Oscar Samuel Veliz graduated from The University of Texas at El Paso (UTEP) with a Bachelor of Science in Computer Science in May 2011 and a Doctorate of Philosophy in Computer Science in August 2021. While working on his bachelor's degree he earned a UTEP Presidential Scholarship, the Alpha Phi Omega Social Fraternity Outstanding Engineering Student Award, a Top Ten Engineering Student Award, and was the College of Engineering Banner Bearer during his commencement. He was also a member of Mortar Board, the Association for Computing Machinery (ACM), served on the Engineering Student Leadership Council, and was Webmaster for The Society of Mexican American Engineers and Scientists (MAES) and the Society of Hispanic Professional Engineers (SHPE). While completing his undergrad he also had two on-campus jobs as a Peer Career Adviser with the University Career Center and as the Senior Microsoft Student Partner for UTEP with Microsoft. Somehow, he also found time to do a summer internship with ExxonMobil working on upstream research.

After graduating Oscar did another internship this time with Hewlett-Packard before returning to UTEP for graduate school. He joined the prestigious Intelligent Agent Strategic and Reasoning Lab with Dr. Christopher Kiekintveld and has worked as both a Teaching Assistant and Research Assistant. He has presented his research findings at International Joint Conference on Artificial Intelligence and at the AAAI Conference on Artificial Intelligence. Oscar has also taught CS1320 at UTEP and has been a long-time volunteer with Microsoft Philanthropies Technology Education and Literacy in Schools (TEALS) program, teaching CS at four local high schools. You will find lots of Numerical Analysis lessons on his YouTube channel with over 5,000 subscribers. In Fall 2020 Oscar joined Carnegie Mellon University as a Post-Doctoral Teaching Fellow in Information Systems.