BOUNDED RATIONALITY IN REPEATED GAMES AND MECHANISM DESIGN FOR AGENTS IN COMPUTATIONAL SETTINGS

by

Thomas C. O'Connell

A Dissertation
Submitted to the University at Albany, State University of New York
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

College of Arts and Sciences
Department of Computer Science
2000

To Ellen,

This thesis is as much a product of your work as it is mine.

Abstract

In Part I, we study bounded rationality in repeated two-person zero-sum games. First we investigate infinitely repeated games in which both players are restricted to pure strategies that can be executed on a finite automaton. In particular, we provide an upper bound on the number of states that Player 2 needs to defeat Player 1 when Player 1 is restricted to simple cycles of length m. Next we argue that the finite automaton approach to bounded rationality is not satisfactory. As an alternative, we propose limiting the number of strategies available to the players. We provide a thorough study of finitely repeatedly zero sum games in which Player 1 is restricted to mixing over a fixed number of pure strategies while Player 2 is unrestricted. We describe an optimal set of pure strategies for Player 1 and a method for describing these strategies such that any strategy from this set can be efficiently executed given its description. We develop upper and lower bounds on the value of these games and discuss how the value is related to the strategic entropy function defined by Neyman and Okada (1999). Finally, we show that an approximately optimal set can be produced in time which is linear in the size of the set. This set achieves a total expected payoff that is within an additive constant of the optimal.

In Part II, we investigate the problem of designing mechanisms to control collective decisions made by self-interested autonomous agents. In particular, we examine how results in the economics literature on mechanism design apply to collective decisions involving NP-hard optimization problems. We formalize the idea of polynomial time mechanism design and investigate both dominant strategy and Nash implementation for a multiagent version of MAXSAT. In particular, we show that there exists a polynomial time mechanism for multiagent MAXSAT that guarantees the outcome to be within a factor of $1/2$ of the optimal outcome. We also show that, in general, a $1/2$ approximation is the best approximation possible for dominant strategy, Nash, undominated Nash and subgame perfect implementation. Our analysis highlights some of the difficulties that arise in applying results from mechanism design to computational problems.

ACKNOWLEDGEMENTS

There are so many people that have helped me over the years that I am sure I will overlook someone. First, I would like to thank Professor Stearns. I cannot imagine having a better advisor both in terms of intellect and personality. I would also like to thank the members of my committee – Professors Berg, Haas, Kranich and Ravi. Professor Berg first introduced me to research, serving as the advisor for my master's thesis. I am grateful to him for giving me the freedom to follow my interests, even when those interests eventually took me away from his machine learning lab. I am also grateful to Professor Berg for allowing me access to that lab long after my official association with it had ended. My many hallway conversations with Professor Haas served as a source of encouragement for me to continue in the Ph.D. program on the numerous occasions that I considered going back to industry. Professor Kranich provided invaluable assistance to me as I tried to understand a great deal of economics research in a short period of time. Without his help, I simply would not have been able to complete Part II of this thesis. Professor Ravi served as a member of my committee prior to his sabbatical. I regret that he could not be here for my defense. I know no one as generous as Ravi. There is no doubt in my mind that without him I would never have completed my degree. There is nothing I could do that could repay him for the kindness he has shown me and my family.

Many of my teachers encouraged me over the years including Professor Richard McGovern of Marist College who first started me on this Ph.D. pursuit, Professors Steven Andrianoff and Dalton Hunkins of St. Bonaventure University, and Professors Don Wilken, Louisa Slowiacek and Paliath Narendran of SUNY–Albany. Professor Abraham Neyman of SUNY–Stony Brook provided many helpful and encouraging comments during my presentations at the International Game Theory Conference in 1998 and 1999.

I would like to thank the many students and staff members who made my stay in Albany bearable. If I tried to list them all, I would certainly omit someone important so I won't even try. However, I should single out my office mates Christian Harris,

George Becker and Haleh Najafzadeh for putting up with my mess over the years and Lance Nevard who actually took the time to proofread the first couple of chapters of this thesis which I am sure was no easy task. I would also like to thank Pat Keller for helping me deal with all the bureaucratic hurdles associated with the graduate program.

I am grateful for the financial support I received under National Science Foundation Grant CCR-97-34936 and the Faculty Research Award given to Professor Ravi by the University at Albany. I was also supported under a University Fellowship and I thank the computer science faculty for renewing that support every year.

Finally, I would like to thank my family for telling me (and this is an exact quote) "Quite honestly, we are all quite bored of saying 'Oh yeah, one of my relatives is getting a PhD' ... Get the degree!" My parents taught me to love to learn and to love games. Who knew all that game playing could amount to something? I would like to thank my son Brian who would probably have a lot more cool toys if I had a real job and who could care less about my degree. He constantly reminded me not to take my Ph.D. too seriously – after all my true purpose in life is to play basketball with him.

There are no words to adequately thank my wife for what she has endured. Without her I would have nothing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates bounded rationality in both repeated games and mechanism design using the tools of theoretical computer science. The idea of bounded rationality has its roots in the 1950's in the work of Simon (see Simon, 1982). Traditional game theoretic models assume that the players are perfectly rational. By perfectly rational, we mean each player plays its best possible action regardless of its ability to determine which action that is. In other words, traditional game theory ignores the procedural aspects of the decision making process[1]. In particular, even if computing the player's best action would require an enormous amount of time, perhaps even longer than one could expect any player to exist, it is assumed that the player somehow determines and plays its best action. Because of their dissatisfaction with this assumption, many researchers have studied games in which players are boundedly rational. A boundedly rational player has some limit on its ability to determine its best action in a game.

In Part I, we study bounded rationality in repeated games. In the study of repeated games, bounded rationality often takes the form of a limit on the player's ability to remember the history of play. One commonly used approach suggested by Aumann (1981) is to limit the pure strategies available to a player to strategies that can be executed on finite automata. Chapter 2 provides an introduction to repeated

[1]For a detailed discussion of bounded rationality in game theory see Rubinstein (1998).

two-person zero-sum games with finite automata. In Chapter 3, we show that when Player 1 is further restricted to finite automata that are simple cycles of size m, Player 2 needs at most $3m2^{m+1}$ states to "defeat" Player 1 in an infinitely repeated game.[2] In Chapter 4, as an alternative to the finite automaton approach, we suggest limiting the number of pure strategies that a player is allowed to use. We thoroughly investigate finitely repeated two-person zero-sum games in which one of the players is restricted in this way while the other is unrestricted. We provide upper and lower bounds on the value of these games and investigate the computational requirements of optimal play under these restrictions.

In Part II, we study bounded rationality in mechanism design. Rather than restricting the players to finite automata or to sets of a particular size as we do in Part I, we restrict the players to polynomial time computation on a general computing device. Chapter 5 introduces the mechanism design problem. Chapter 6 formalizes what we mean by polynomial time mechanisms and investigates polynomial time mechanism design when the social choice rule is hard to compute and, therefore, must be approximated. In particular, we provide polynomial time mechanisms for approximate social choice rules for a multiagent version of MAXSAT. Part II can be read independently of Part I.

The appendices provide short introductions to the concepts from game theory and computational complexity theory that are used throughout the main body of the thesis but which are not thoroughly explained there. Readers unfamiliar with game theory or computational complexity theory may wish to read the appropriate appendix before reading the rest of the thesis.

[2]The terms used in this Introduction will be defined in the subsequent chapters.

Part I

Bounded Rationality in Repeated Games

Chapter 2

An Introduction to Repeated Games with Finite Automata

There have been many studies of bounded rationality in repeated games (see Kalai, 1990 and Aumann, 1997 for surveys). A number of these studies restrict the pure strategies available to one or more of the players to strategies that can be executed by finite automata of a particular size where the size is the number of states in the automaton.

Definition 2.1 *A* **finite automaton**[1] *is a 6-tuple* $\langle Q, q_1, \beta, \Lambda, \delta(\cdot), g(\cdot)\rangle$ *where:*

Q *is a finite, nonempty set of states.*

q_1 *is the starting state.*

β *is a finite input alphabet.*

Λ *is a finite output alphabet.*

$\delta : Q \times \beta \to Q$ *is the transition function.*

$g : Q \to \Lambda$ *is the output function.*

[1] As we have defined it, this is usually called a Moore machine. See Hopcroft and Ullman (1979) and Moore (1956).

A pure strategy for a repeated game that can be executed on a finite automaton is called a **finite state strategy**. The size of a finite state strategy is the number of states in the smallest automaton that executes that strategy. We think of a finite automaton executing a pure strategy for Player 1 in a repeated game by associating the automaton's output alphabet with the actions available to Player 1 and its input alphabet with the actions available to Player 2. At each stage of a repeated game, the action chosen by Player 1 is the symbol specified by the output function for the current state of the automaton. The automaton changes state based on the actions chosen by Player 2. For example, suppose Player 1's action set is $\{a, b\}$ and Player 2's action set is $\{c, d\}$. Figure 2.1 shows a two state finite automaton $\langle \{q_1, q_2\}, q_1, A_2, A_1, \delta(\cdot), g(\cdot) \rangle$ that executes the pure strategy "Play a until Player 2 plays d and play b thereafter."

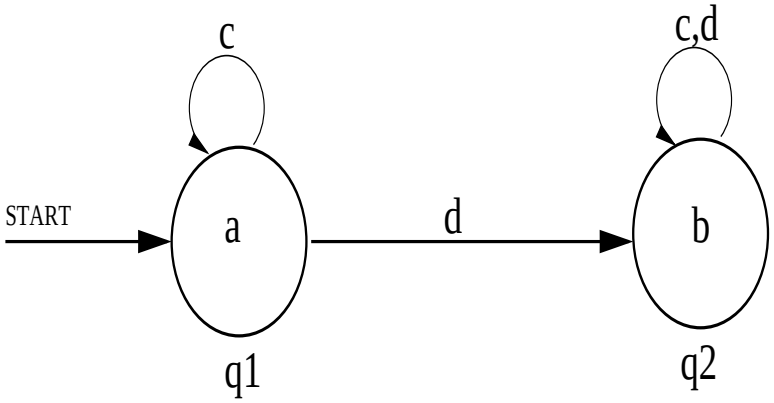

Figure 2.1: A finite automaton that executes a pure strategy for Player 1. The output associated with each state is shown inside the state.

Finite state strategies of size m partition the history of play into m equivalence classes where the strategy plays in exactly the same way following each history in an equivalence class. A finite state strategy, therefore, has a limited ability to react to the history of play. For example, "Play a if and only if the sequence of actions played by Player 2 so far has the form $c^k d^k$ for some nonnegative integer k" is not a finite state strategy (see Hopcroft and Ullman, 1979, Chapter 3).

There are two papers in the literature on repeated games with finite automata that are particularly pertinent to our discussion – Ben-Porath (1993) and Neyman and Okada (1999).

2.1 Infinitely Repeated Games

Ben-Porath (1993) considers infinitely repeated two-person zero-sum games in which the set of pure strategies available to Players 1 and 2 are restricted to finite state strategies of size m and $Q(m)$ respectively for some integer $m > 0$.

Definition 2.2 *A* **two-person zero-sum** *game is a 3-tuple* $G = \langle A_1, A_2, r \rangle$ *where:*

A_1 *is a finite set of actions for Player 1.*

A_2 *is a finite set of actions for Player 2.*

$r : A_1 \times A_2 \to \mathcal{R}$ *is a payoff function.*

Player 1 chooses his action so as to maximize the payoff.

Player 2 chooses his action so as to minimize the payoff.

By the Minimax Theorem (Proposition A.7) every finite two-person zero-sum game G has a value which is denoted by $V(G)$. Let s_1 and s_2 be finite state strategies for Players 1 and 2 respectively. Define the payoff for this pair of strategies in the infinite repetition of G by:

$$R(s_1, s_2) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} r(a_1^i, a_2^i) \tag{2.1}$$

where a_1^i and a_2^i are the actions chosen at stage i by Players 1 and 2 respectively assuming the players are playing strategies s_1 and s_2. Because s_1 and s_2 are finite state strategies, the sequence of payoffs that they generate is cyclic and, therefore, the limit in Equation 2.1 exists. Hence, $R(s_1, s_2)$ is well defined for all pairs of finite state strategies s_1 and s_2. Let $G^{\infty}_{m,Q(m)}$ denote the infinite repetition of G where Player 1 is restricted to finite state strategies of size m and Player 2 is restricted to finite state

strategies of size $Q(m)$. Since the payoff is well defined for each pair of finite state strategies, $G^{\infty}_{m,Q(m)}$ has a value which is denoted by $V(G^{\infty}_{m,Q(m)})$. In other words,

$$V(G^{\infty}_{m,Q(m)}) = \max_{\sigma \in \Delta(S_1^m)} \min_{\gamma \in \Delta(S_2^{Q(m)})} E_{\sigma,\rho}[R(s_1, s_2)]$$

where $E_{\sigma,\rho}[R(s_1, s_2)]$ denotes the expected value of the payoff given that Players 1 and 2 are playing mixed strategies σ and ρ respectively, S_1^m is the set of finite state strategies of size m for Player 1, $S_2^{Q(m)}$ is the set of finite state strategies of size $Q(m)$ for Player 2, and $\Delta(S)$ denotes the set of mixed strategies over a set of pure strategies S.

We say that Player 2 **defeats** Player 1 if the value of the repeated game is $\max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$. Ben-Porath (1993) proves the following two results which indicate that when Player 1 is restricted to finite state strategies of size m, the number of states that Player 2 needs to defeat Player 1 is at least exponential in m but no more than $|A_1|^m m^{m|A_2|+2}$.

Proposition 2.3 *If Player 1 is restricted to finite state strategies of size m and Player 2 is restricted to finite state strategies of size $Q(m)$ where $\lim_{m\to\infty} \frac{\log_2 Q(m)}{m} = 0$ then $\lim_{m\to\infty} V(G^{\infty}_{m,Q(m)}) = V(G)$.*

Proposition 2.4 *Let S_1^m be the set of finite state strategies of size m for Player 1. Player 2 has a finite state strategy s_2 of size $Q(m) = |A_1|^m m^{m|A_2|+2}$ such that, for all $s_1 \in S_1^m$, $R(s_1, s_2) \leq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$.*

In Ben-Porath's proof of Proposition 2.3, Player 1 only plays simple cycles so the result holds even when Player 1 is restricted to pure strategies that are simple cycles of size m.

Definition 2.5 *A finite automaton $C = \langle \{1, \ldots, m\}, 1, \beta, \Lambda, \delta(\cdot), g(\cdot) \rangle$ is a simple cycle if the following two conditions hold:*

1. *for all states $i, 1 \leq i < m$, and all symbols $b \in \beta$, $\delta(i, b) = i + 1$*

2. for all symbols $b \in \beta$, $\delta(m, b) = 1$

Figure 2.2 illustrates a simple cycle of size 3. There are no labels on the transitions since the transitions are taken regardless of the last symbol read. In Chapter 3, we

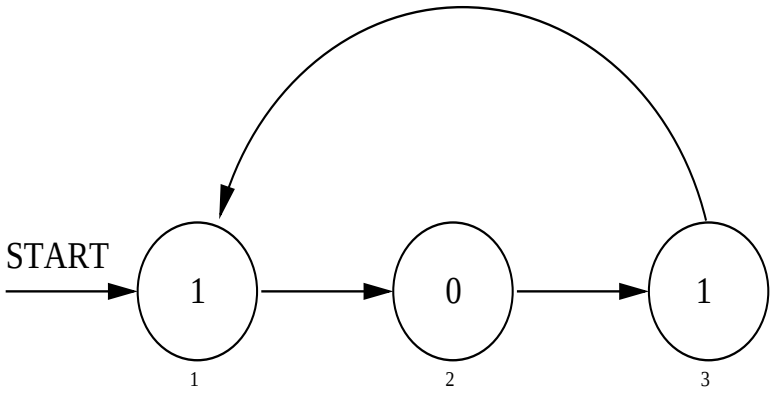

Figure 2.2: A simple cycle.

show that Player 2 needs at most $3m2^{m+1}$ states to defeat Player 1 when Player 1 is restricted to simple cycles of size m.

2.2 Finitely Repeated Games

Neyman and Okada (1999) consider N-stage two-person zero-sum games in which the pure strategies available to Player 1 are finite state strategies of size $m(N)$ while Player 2 is unrestricted. They define a function called strategic entropy to measure the uncertainty that a mixed strategy creates about the sequence of play. and demonstrate conditions under which a Player with bounded strategic entropy will necessarily be defeated by an unbounded opponent. They state, however, that strategic entropy should not be viewed as a measure of strategic complexity per se but rather as a useful "abstract informational feature" of repeated games with finite automata. They use strategic entropy to prove the following result, originally conjectured by Neyman (1997), regarding the relationship between the size of the strategies available to Player 1 in an N stage game and the value of the game as N goes to infinity.

Proposition 2.6 *In an N-stage game, suppose Player 1 is restricted to finite state strategies of size $m(N)$. If $m(N)\log_2 m(N) = o(N)$ then*

$$\lim_{N\to\infty} V(G^N_{m(N)}) \;=\; \max_{a_1\in A_1}\min_{a_2\in A_2} r(a_1,a_2)$$

where $V(G^N_{m(N)})$ is the value of the N-stage game given the above restriction on Player 1. ∎

In Chapter 4, rather than restricting Player 1 to finite state strategies, we consider finitely repeated games in which Player 1 is restricted to pure strategy sets of cardinality K for some integer $K \geq 2$. We provide upper and lower bounds on the value of these games and use these bounds to give an alternative proof of Proposition 2.6. We also provide $O(K)$ time algorithms to compute Player 1's optimal set of pure strategies when the stage game is 2×2 and an approximately optimal set when the stage game is bigger than 2×2. Providing such algorithms is important since there are two phases to the decision making process. In the first phase, the agent has to design his best strategy. In the second phase, the agent has to execute this strategy. Most of the work on bounded rationality in repeated games considers only the second phase. Papadimitriou (1992) argues that the first phase is equally important. We also relate our results to strategic entropy. There are other informational measures that one might use in place of strategic entropy (see Chapter 3 of Aumann and Maschler, 1995 which is a reprint of Stearns, 1967) and it is not clear why strategic entropy should be chosen over something else. Entropy, however, arises naturally in our analysis of repeated games with finite strategy sets. Our results show that entropy is closely tied to the value of repeated games with finite strategy sets. These results suggest that the power of strategic entropy as an informational measure relies on the close connection between the entropy of a mixed strategy and the number of pure strategies available to Player 1.

Chapter 3

Defeating Cycles

3.1 The Algorithm

In this chapter, we develop an upper bound on the number of states that Player 2 requires to defeat Player 1 when Player 1 is restricted to simple cycles of size m. Impose an ordering on Player 1's cycles and suppose Player 2 started his automaton by hypothesizing that Player 1 is playing cycle number 1. As soon as the sequence of actions deviates from Player 1's first cycle, Player 2 could change his hypothesis to "Player 1 is playing cycle number 2." This is essentially the approach used by Ben-Porath (1993) in the proof of Proposition 2.4. For example, suppose Player 2 knows that Player 1 is playing one of the two cycles in Figure 3.1. If Player 2 sees a 1 at the first stage he knows Player 1 is playing cycle number 1 and if he sees a 0 at the first stage he knows Player 1 is playing cycle number 2. Figure 3.2 shows an automaton for Player 2 to defeat Player 1 in this case. Notice that if we ignore the starting state of the two cycles in Figure 3.1, the two cycles are equivalent up to a renaming of states. Is it possible for Player 2 to make use of this property to reduce the number of states he needs to defeat Player 1? Suppose Player 2 could design a finite automaton to solve the following problem:

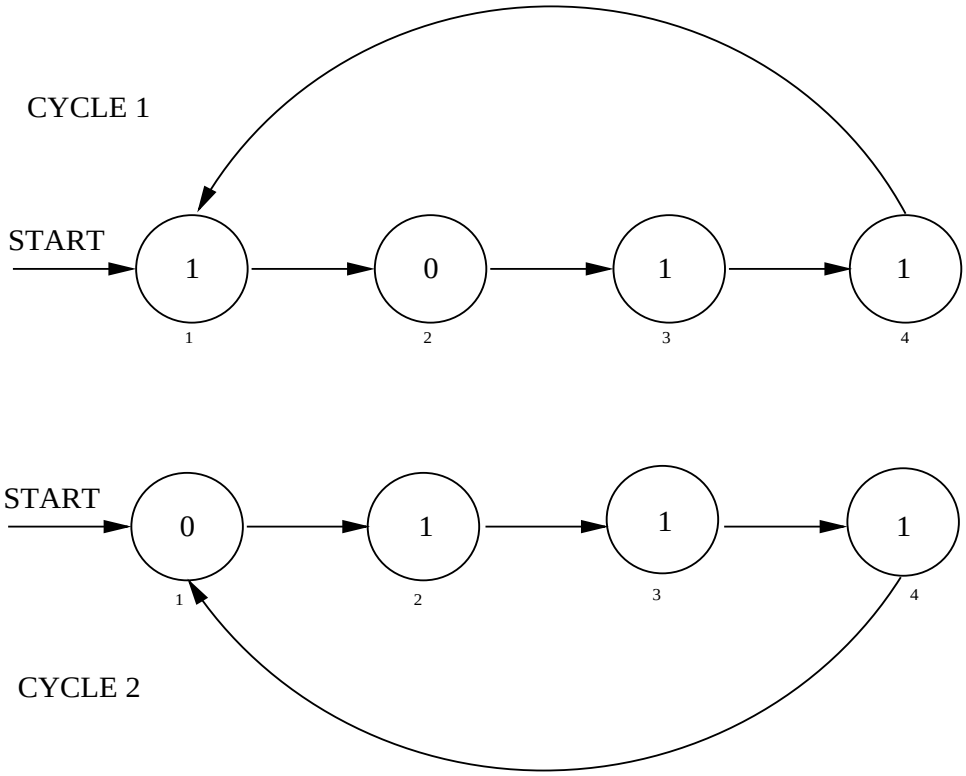

Figure 3.1: Two possible cycles for Player 1.

Problem 3.1 *Let C be a simple cycle of size m and let w be an infinite string. If w could have been generated by cycle C, determine which state w started in and play the best response to w for the remaining stages. If w could not have been generated by C then reject the string and exit.*

For each cycle C of size m, there could be as many as m^2 different cycles which are equivalent to C up to a renaming of states and ignoring the starting state. An $O(m)$ state solution to Problem 3.1 could reduce the number of states that Player 2 requires by a factor of m.

If Player 2 did not have to worry about creating a small finite automaton to solve Problem 3.1, he could use the following algorithm which is stated rather informally:

Algorithm 3.1.

1. Start by hypothesizing that the string started in state 1.

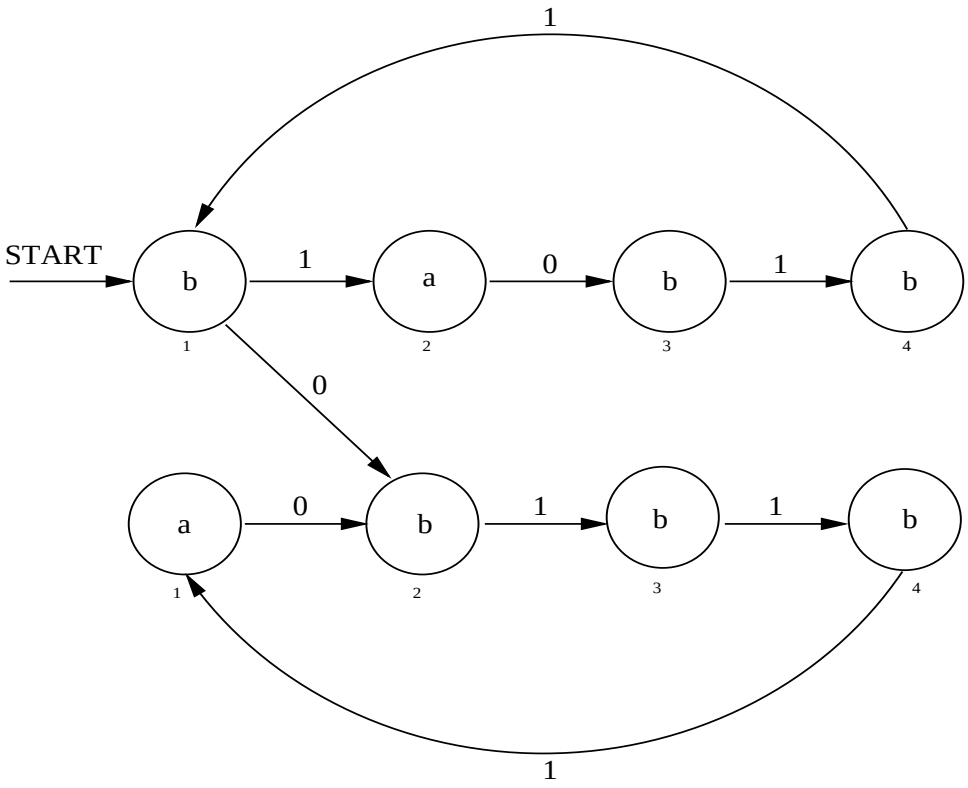

Figure 3.2: An automaton for Player 2 to defeat the cycles in Figure 3.1 assuming a and b are Player 2's best responses to actions 1 and 0 respectively.

2. Play the best response to the currently hypothesized state.

3. If at some point the string is not consistent with the hypothesized starting state, change the hypothesis to the next starting state that is consistent with the string seen so far and continue.

4. If the string seen so far is not consistent with any starting state, reject the string and exit.

We assume the cycle is given in its minimized form.[1] In other words, no two states are equivalent, i.e., no two states generate exactly the same strings. Hence, for any string generated by the cycle, after a finite number of iterations the above algorithm will hypothesize the correct starting state and, therefore, Player 2 will play a best response to Player 1 from some finite stage onward.

We use this informal algorithm as a guide in designing Player 2's automaton. This automaton will spend a number of states determining Player 1's starting state and then transition into the appropriate state in the best response automaton. If at any point the string is found to be inconsistent with Player 1's cycle, then Player 2's automaton will enter the REJECT state.

Suppose we create an automaton M for Player 2 with states $q_1, \ldots, q_m$ as follows where M being in state q_i corresponds to hypothesizing that Player 1 is in state i. M starts in state q_1. If the next action played is consistent with starting in state 1, M moves to state q_2. If the next action is not consistent with starting in state 1, M jumps to the next state that is consistent with the string it has seen so far. This corresponds to updating the hypothesis. If the hypothesis has changed from "started in state 1" to "started in state 2", M jumps to state q_3 since M has already seen one action. For example, consider the simple cycle in Figure 3.3. Suppose Player 2's best response to actions 1 and 0 are a and b respectively. M starts in state q_1. If Player 1 plays 1 at the first stage, M moves to state q_2 since this action is consistent

[1] If not, minimize it using the algorithm given in Hopcroft and Ullman (1979).

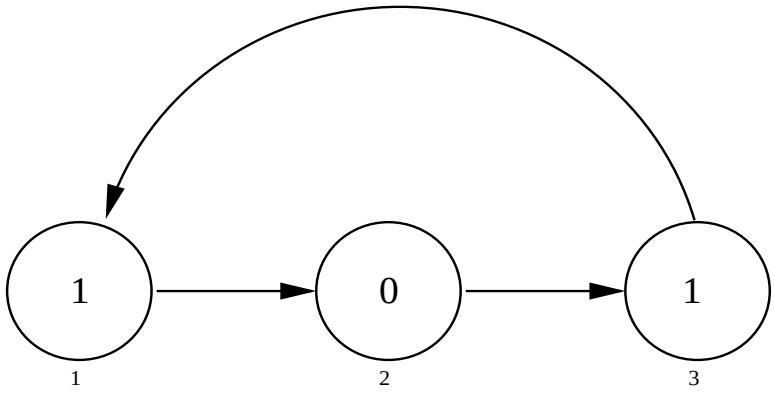

Figure 3.3: A simple cycle with an unspecified starting state.

with Player 1 starting in state 1. If Player 1 plays 0 at the first stage, then he cannot have started in state 1. The first state in the cycle that Player 1 could have started in is state 2. Player 2's hypothesis should, therefore, change to "Player 1 started in state 2". Since M has already seen one action, this corresponds to hypothesizing that Player 1 is currently in state 3. Therefore, M moves to state q_3. This is shown in Figure 3.4.

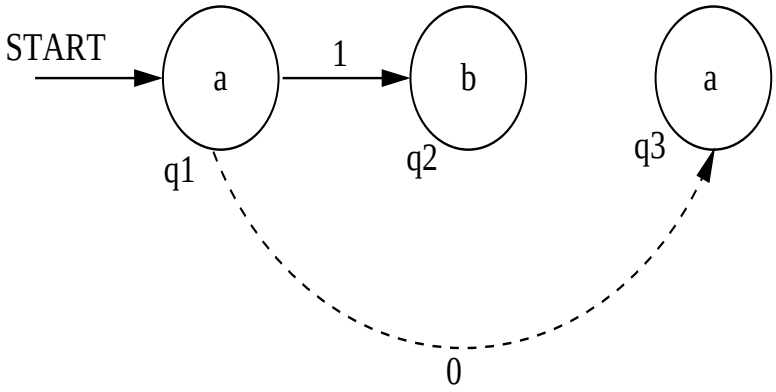

Figure 3.4: A partial construction of an automaton to defeat the simple cycle which appears in Figure 3.3. A dashed line indicates a change in hypothesis.

Suppose the hypothesis changed from "started in state 1" to "started in state 3". In that case, M should jump to state q_1 because we have already seen one action. Consider how Player 2 should construct M to defeat the simple cycle given in Figure 3.5. This is the same cycle as that in Figure 3.3 except that the states are

renumbered. Again M starts in state q_1 and moves to state q_2 if Player 1 plays a

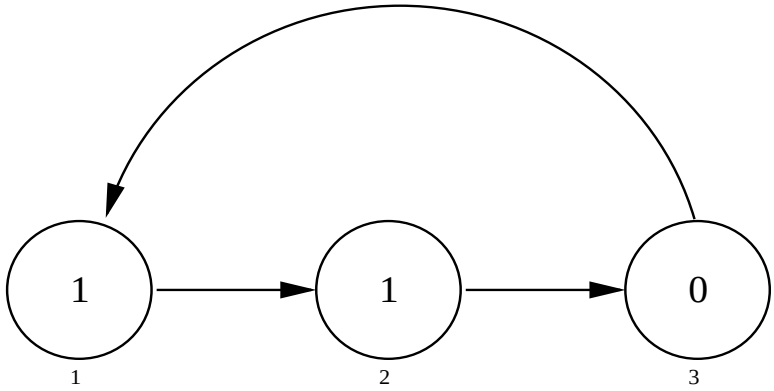

Figure 3.5: A simple cycle with an unspecified starting state.

1 at the first stage. If Player 1 plays a 0 at the first stage, the hypothesis should change to "Player 1 started in state 3". Since M has already seen one action this corresponds to hypothesizing that Player 1 is currently in state 1. Hence, M should move to state q_1. This creates a self loop at state q_1 which is shown in Figure 3.6. Recall, however, that Player 2 is required to reject any string that is not generated

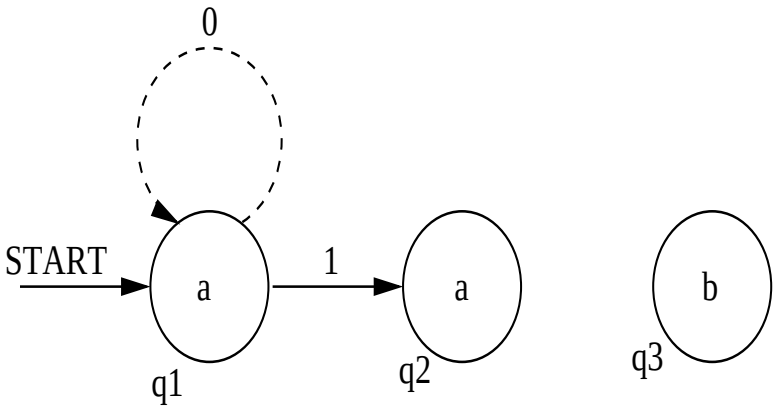

Figure 3.6: An automaton in which a change in hypothesis causes a suboptimal cycle.

by the cycle. The automaton in Figure 3.6 does not reject a string consisting entirely of 0's even though it could not have been generated by Player 1's cycle. M needs to be constructed carefully to avoid creating loops prematurely. This can be done by using $3m$ states rather than m states. M being in state q_{tm+i} where $t = 0, 1$, or 2,

corresponds to the hypothesis that the cycle is in state i.[2] The complete algorithm for constructing M is given in Algorithm 3.2. To simplify the notation, define $C(i,j)$ to be the string that is output when automaton C is started in state i and moves to state j. $C(i,i)$ is defined to be the empty string ϵ. At the beginning of the algorithm, the m-state cycle given as input is unfolded into an equivalent $3m$-state automaton in which there is a simple prefix of length $2m$ followed by the original cycle of length m. This equivalent automaton is assumed to start in one of the first m states. Figure 3.7 shows the automaton equivalent to the simple cycle of Figure 3.5.

Algorithm 3.2: CycleKiller(C').

/* Creates an automaton $M = \langle \{q_1, \ldots, q_{3m}, \text{REJECT}\}, q_1, A_1, A_2, \Delta(\cdot), \gamma(\cdot)\rangle)$ */

/* $\Delta(\cdot)$ is the transition function for M. */

/* $\gamma(\cdot)$ is the output function for M. */

1. Convert cycle C' into the equivalent $3m$ state automaton $C = \langle \{1, \ldots, 3m\}, 1, A_2, A_1, \delta(\cdot), g(\cdot)\rangle$ as described in the text above. The states in C are ordered from 1 to $3m$ where, for all $a \in A_2$, $\delta(1,a) = 2, \ldots,$ $\delta(3m-1,a) = 3m$ and $\delta(3m,a) = 2m+1$.
2. Create the $3m+1$ states for M $(q_1, ..., q_m, q_{m+1}, ..., q_{3m}, \text{REJECT})$.

/* The following loop creates the transitions out of each state q_i in M. */

3. For $i = 1$ to $3m$
4. Set $\gamma(q_i)$ to Player 2's best response to $g(i)$.
5. Let x_i be the string on the shortest path from q_1 to q_i in M where $x_1 = \epsilon$.
6. For each action a in the action set of Player 1
7. Let k be the lowest numbered state if any in C such that $1 \leq k \leq m$ and $C(k,j) = x_i a$ for some j in C.
8. If no such k exists then $\Delta(q_i, a) = \text{REJECT}$
9. else $\Delta(q_i, a) = q_j$
10. Return $M = \langle \{q_1, \ldots, q_{3m}, \text{REJECT}\}, q_1, A_1, A_2, \Delta(\cdot), \gamma(\cdot)\rangle$

[2] M being in state tm for $t = 1$ or 2 corresponds to the hypothesis that the cycle is in state m.

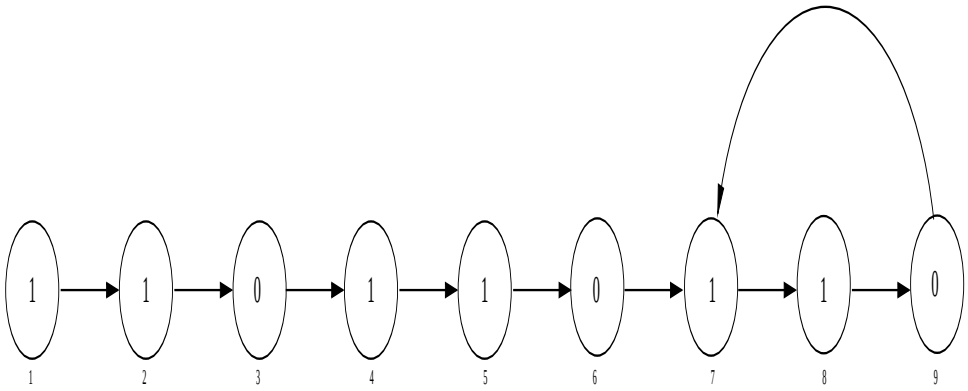

Figure 3.7: The 9-state automaton equivalent to the simple cycle from Figure 3.5.

The automaton created for the cycle in Figure 3.5 is shown in Figure 3.8. Notice

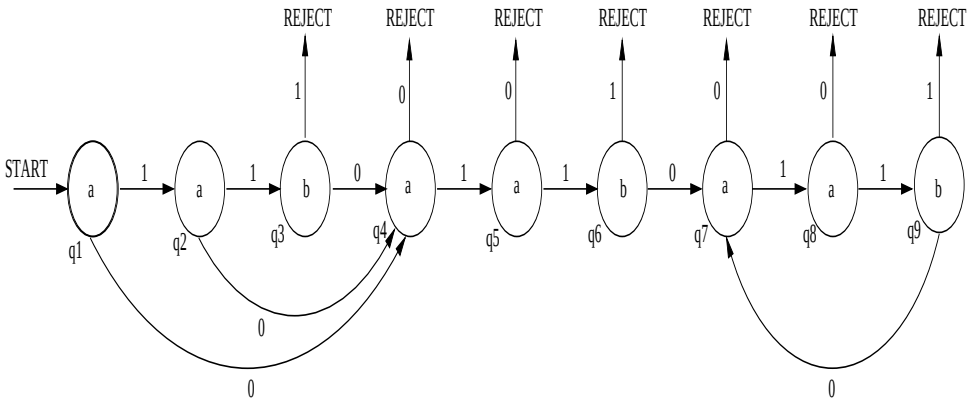

Figure 3.8: The automaton created by Algorithm 3.2 for the cycle in Figure 3.5.

that instead of having a self loop at state q_1, M moves to state q_4 when Player 1 plays 0 at the first stage. Algorithm 3.2 avoids creating a loop until M has seen enough of the string that it can be sure the loop will not trap any string which is not generated by C.

3.2 Proof of Correctness

By using a series of lemmas, we show that the above algorithm creates an automaton that plays the best response to Player 1's cycle after a finite number of stages. Since Step 1 of Algorithm 3.2, converts Player 1's cycle into an equivalent automaton C, it

suffices to show that the algorithm creates an automaton that plays the best response to automaton C after a finite number of stages. As in Algorithm 3.2, let x_i be the shortest string such that $\Delta^*(q_1, x_i) = q_i$.[3] We say that a string x is **consistent with a state** j in automaton $C = \langle \{1, \ldots, 3m\}, 1, A_2, A_1, \delta, g(\cdot) \rangle$ if there is some state k of C such that $1 \leq k \leq m$ and $C(k, j) = x$. The proof proceeds as follows:

1. Lemma 3.2 shows that x_i is consistent with state i for all i.

2. Lemma's 3.3 and 3.4 imply that any string reaching a state beyond q_{2m} can be consistent with at most one state in C.

3. Lemma's 3.5 and 3.7 imply that any string consistent with C eventually reaches a state beyond q_{2m} in M and any string that is not consistent with C is either rejected before state q_{2m} or reaches a state beyond q_{2m} in M.

These results together imply that once a string reaches a state beyond q_{2m}, M has uniquely determined the state that C is currently in and, therefore, can begin to play the best response automaton for C. Any inconsistency encountered after state q_{2m} indicates that Player 1 is not playing automaton C so the string can be rejected.

First, we show that x_i is always consistent with state i in C.

Lemma 3.2 *For $1 \leq i \leq 3m$, x_i is consistent with state i.*

Proof.

Case 1: $i = 1$.

$x_1 = \epsilon$ so since $C(1, 1) = \epsilon$, x_1 is consistent with 1.

Case 2: $i > 1$.

Since x_i is the shortest string x such that $\Delta^*(q_1, x) = q_i$, $x_i = x_p a$ for some p such that $\Delta(q_p, a) = q_i$. By Step 7 of Algorithm 3.2, there must be a k such that $C(k, i) = x_p a$. Hence, $x_i = x_p a$ is consistent with state i. ∎

[3] $\Delta^*(q, x)$ is defined to be the state reached by an automaton with transition function Δ when starting in state q and reading string x.

Lemma 3.3 *If $2m \leq i \leq 3m$ then i is the only state that x_i is consistent with.*

Proof. Since no two starting states in C are equivalent, for any string w such that $|w| \geq m$, w can be consistent with at most one state in C. Since, by Lemma 3.2, x_i is consistent with i, it suffices to show that $|x_i| \geq m$ when $i \geq 2m$.

Since x_i is consistent with i, there is a k, $1 \leq k \leq m$ such that $C(k, i) = x_i$. Suppose starting at k in C, x_i crosses the edge from state $3m$ to state $2m+1$. It must be the case that $|x_i| \geq m$ since $k \leq m$ and $\delta(t) = t+1$ for all t, $1 \leq t < 3m$. If x_i never crosses the edge from state $3m$ to state $2m+1$, then $k = i - |x_i|$. Since $k \leq m$, $|x_i| \geq i - m \geq 2m - m = m$. Therefore, i is the only state that x_i is consistent with. ∎

Next we show that any string that reaches state q_i in M contains x_i as a suffix. This combined with Lemma 3.3 implies that any string that reaches a state q_i with $i \geq 2m$ is either consistent with state i or was not generated by C.

Lemma 3.4 *For all i, $1 \leq i \leq 3m$, if y is a string such that $\Delta^*(q_1, y) = q_i$ then $y = wx_i$ for some finite string w.*

Proof. By induction on $|y|$.

Basis: $|y| = 0$.

Trivially true since $x_1 = \epsilon$.

Induction Hypothesis: Assume, for all y such that $0 \leq |y| \leq l$, $\Delta^*(q_1, y) = q_i$ implies $y = wx_i$ for some finite string w.

Let y be a string of length $l+1$ such that $\Delta^*(q_1, y) = q_i$. Then there exists a string z, an action a, and a number t, $1 \leq t \leq 3m$, such that $y = za$, $\Delta^*(q_1, z) = q_t$ and $\Delta(q_t, a) = q_i$. Since $|z| = l$, by the induction hypothesis, $z = ux_t$ for some finite string u. Furthermore, since $\Delta(q_t, a) = q_i$, there exists a $k_1, 1 \leq k_1 \leq m$ such that $C(k_1, i) = x_t a$. By Lemma 3.2, there must be a $k_2, 1 \leq k_2 \leq m$ such that $C(k_2, i) = x_i$. Since $\Delta^*(q_1, x_t a) = q_i$ and since x_i is the shortest string reaching q_i

from q_1, $|x_t a| \geq |x_i|$. Observe that if two strings reach the same state in C then one must be a suffix of the other no matter which state each of them started in. Hence, x_i must be a suffix of $x_t a$. Therefore, $y = za = ux_t a = uvx_i$ for some finite string uv. ∎

Lemma 3.5 *No string y that is generated by C is rejected by M.*

Proof. Suppose y is rejected at state q_p for some $p, 1 \leq p \leq 3m$. Then $y = za$ where $\Delta^*(q_1, z) = q_p$ and $\Delta(q_p, a) = REJECT$. By Lemma 3.4, $z = wx_p$ for some finite string w. Since $x_p a$ is a substring of y, it must be consistent with some starting state in C, i.e. there exists a $k, 1 \leq k \leq m$ such that $C(k, j) = x_p a$ for some j in C. But then $\Delta(q_p, a)$ is set to $q_j \neq REJECT$ in Step 9 so y can't possibly be rejected at state q_p. Therefore, y is not rejected by M. ∎

Lemma 3.6 *If $i \leq 2m$ and k and j are such that $1 \leq k \leq m$ and $C(k, j) = x_i$ then $k = j - |x_i|$.*

Proof. If, starting at k in C, x_i does not traverse the edge from state $3m$ to state $2m+1$ on its route to state j then $k = j - |x_i|$ since $\delta(t) = t+1$ for all t, $1 \leq t < 3m$. Since $\delta(t) = t + 1$ for all t, $1 \leq t < 3m$, if x_i did cross this edge then $|x_i| > 2m$. It suffices to show, then, that $|x_i| \leq i$. We can prove this by induction on i.

Basis: $i = 1$. $|x_1| = 0$.

Induction Hypothesis: Assume $|x_i| \leq i$ for all i, $1 \leq i \leq t$ for some t.

It must be the case that $x_{t+1} = x_i a$ for some i, $1 \leq i \leq t$, and some action a. Therefore, by the induction hypothesis, $|x_{t+1}| = |x_i| + 1 \leq i + 1 \leq t + 1$.

Since $|x_i| \leq i \leq 2m$, x_i does not traverse the edge from state $3m$ to state $2m + 1$ and, therefore, $k = j - |x_i|$. ∎

Lemma 3.7 *If $1 \leq i \leq 2m$ and $\Delta(q_i, a) = q_t$ then $t > i$.*

Proof.

Case 1: $i = 1$

$\Delta(q_i, a) = q_t$ implies $C(k,t) = a$ for some $k, 1 \leq k \leq m$. Since state 1 has no incoming edges in C, $t = \delta(k) > 1$.

Case 2: $1 < i \leq 2m$

Let k_i be the least $k, 1 \leq k \leq m$, such that $C(k,j) = x_i$ for some state j of C. Since $\Delta^*(q_1, x_i) = q_i$ and $i > 1$, it must be the case that $x_i = x_p b$ for some p and b such that $\Delta(q_p, b) = q_i$. Since $\Delta(q_p, b) = q_i$ and k_i is the least k such that $C(k,j) = x_i$ for some j, by Step 7 of Algorithm 3.2, $C(k_i, i) = x_i$. This and Lemma 3.6 imply that $k_i = i - |x_i|$.

Suppose $\Delta(q_i, a) = q_t$ where $t \leq i$. Then, by Step 7 of Algorithm 3.2, $C(k,t) = x_i a$ for some $k, 1 \leq k \leq m$ which implies $C(k, t-1) = x_i$. By Lemma 3.6, $k = t-1-|x_i| < i - |x_i| = k_i$ contradicting k_i being the least state k such that $C(k,j) = x_i$ for some state j. Therefore, $t > i$. ∎

Theorem 3.8 *Let M be the automaton created by Algorithm 3.2 given cycle C as input. For any infinite string y generated by C, M plays a best response to y from some stage onward. For any infinite cyclic string y which is not generated by C, M rejects y after a finite number of steps.*

Proof. First suppose y is an infinite string generated by C. Lemma's 3.5 and 3.7 imply that, for any string y consistent with C, M will eventually reach a state q_i with $i \geq 2m$. Therefore, $y = wz$ for some finite string w and infinite string z such that $\Delta^*(q_1, w) = q_i$ for some $i \geq 2m$. By Lemmas 3.3 and 3.4, i is the only state that x_i is consistent with and w has x_i as a suffix. Therefore, i must also be the only state that w is consistent with. Hence, once M reaches a state q_i with $i \geq 2m$, y will be consistent with the current state of M from that point onward. Since, starting at any state q_i of M, M plays the best response to any string generated by C starting at state i, M plays a best response to y from some finite stage onward.

Now suppose y is an infinite cyclic string which is not generated by C. Lemma 3.7 implies that either y is rejected before state q_{2m+1} or y reaches some state q_j for

$j \geq 2m+1$. Suppose y not is rejected before state q_{2m+1}. Then, $y = uv$ where $\Delta^*(q_1, u) = q_j$ for some $j \geq 2m+1$. Since C is equivalent to a cycle, no infinite cyclic string that is not generated by C can have an infinite suffix that is generated by C. Therefore, v is not consistent with C. When $i \geq 2m$, i is the only state that x_i is consistent with. Therefore, for each $i \geq 2m$ and for each action $a \neq g(i)$, $x_i a$ cannot be consistent with any state in C so $\Delta(q_i, a) = \text{REJECT}$. At some point v must contain an action a such that $a \neq g(i)$ when M is in state q_i. Since v is the part of y that is seen after a state beyond q_{2m} is reached, y will be rejected after a finite number of stages. ∎

Theorem 3.9 *Let $\mathcal{C}_m$ be the set of simple cycles of size at most m for Player 1. Player 2 needs at most $3m|\mathcal{C}_m| \leq 3m2^{m+1}$ states to defeat Player 1 when Player 1's set of pure strategies is $\mathcal{C}_m$.*

Proof. Use Algorithm 3.2 to create an automaton M_j for each of Player 1's cycles $C_j \in \mathcal{C}_m$ but, rather than going to the REJECT state, M_j will go to the starting state of the next automaton in the list M_{j+1}. By Theorem 3.8, any time M_j sees a cyclic string that is not generated by C_j, it will eventually reject that string and move to the next automaton. After a finite number of such rejections, it will move to the correct automaton and stay there playing the best response to Player 1's cycle. Since Player 2 makes a finite number of suboptimal plays, from some point onward he will be playing his best response to Player 1 and, therefore, he defeats Player 1. Since Player 1 has at most 2^{m+1} cycles of length at most m and since Player 2 uses at most $3m$ states in each M_j, Player 2 uses at most $3m|\mathcal{C}_m| \leq 3m2^{m+1}$ states. ∎

Chapter 4

Repeated Games with Finite Strategy Sets

4.1 Introduction

Although restricting players to finite state strategies of some limited size is currently the most common approach to modeling bounded rationality in repeated games, Stearns (1989) argued that the state counting measure of strategic complexity is not satisfactory since the number of states required by a finite automaton to execute a strategy is not an accurate measure of the amount of memory required by a more general computing device to execute the same strategy. He illustrated this inaccuracy with the following example of three strategies for playing matching pennies.

Example 4.1 *Let n be the stage number, a_n be the action chosen by Player 1 at stage n and b_n be the action chosen by Player 2 at stage n.*

Strategy A: *Play $a_n = H$ if $n \equiv 0 \pmod{10^6}$ and $a_n = T$ otherwise.*

Strategy B: *Play $a_n = H$ if $n \leq 10^6$ and $a_n = b_{n-10^6}$ otherwise.*

Strategy C: *Let R be a table of 10^6 random bits. Play $a_n = R[n \bmod 10^6]$.*

Table 4.1: State and memory requirements for the strategies given in Example 4.1.

Strategy	Number of States	Approximate Memory
A	10^6	$6 \log_2 10$ bits
B	$2^{(10^6)}$	10^6 bits
C	10^6	10^6 bits

Table 4.1 shows the number of states and the amount of memory required to execute each of these three strategies. Strategies A and C appear to be equally complex if we look at the state count but strategy A can be implemented using exponentially less memory. Strategy B is exponentially more complex than Strategy C according to the state count yet it can be implemented using approximately the same amount of memory. Thus the relative complexity of strategies depends on the model of computation chosen to evaluate the strategies. Theoretical results can vary qualitatively based on the computational model chosen. For example, as we saw in Chapter 2, Ben-Porath (1993) showed that if Player 1 is allowed to mix over all m-state finite automata then, unless Player 2 has an exponentially larger automaton, the value of the repeated game converges to the value of the stage game. However, if we look at the proof of this result in terms of memory requirements, the proof has Player 1 mixing over finite automata that are simple cycles. These automata require m bits of memory to implement on a general computing device. Since a finite automaton of size 2^m might require as little as m bits of memory to implement, Ben-Porath's proof shows that Player 2 needs $\Omega(m)$ bits of memory to defeat Player 1. In fact, Stearns (1989) shows that $O(m)$ bits is all that Player 2 needs to defeat Player 1 if Player 1 is restricted to an m-bit memory. Therefore, according to the state counting measure, Player 2 needs exponentially more resources to defeat Player 1 while according to the memory measure Player 2 needs only linearly more resources.

Focusing solely on the amount of memory that a program uses in executing a strategy ignores the fact that the program may require an enormous amount of time to determine the action to be taken at each stage. Computation time must be considered a resource in any measure of strategic complexity. It is this time consideration that

makes the finite automaton attractive because the time that a finite automaton takes at each stage is considered to be independent of both the length of the game and the amount of resources available to the player. A more realistic measure of complexity would take into account both the total amount of memory used and the amount of time used at each stage. Unfortunately, additional difficulties arise in defining a general computational model with constant time and memory restrictions since different computational models (random access machines, linear bounded automata, etc.) will yield different results and it is not clear which model is most appropriate.

In this chapter, rather than restricting Player 1 to a particular level of resources on a particular type of machine, we simply limit the number of pure strategies that Player 1 is allowed to use. We first determine Player 1's best set of strategies given an unbounded opponent and then investigate the type and amount of computational resources required to produce an optimal set and to execute the strategies in this set. In Section 4.2, we show that, for any $K \geq 1$, Player 1 has an optimal set of K pure strategies which ignore Player 2's actions and therefore can be represented by a tree with K leaves. We also provide an algorithm to produce this set when the stage game is 2×2. The algorithm runs in time which is linear in the size of the set. In Section 4.3, we explain how a machine, given a description of a strategy from an optimal set, can execute the strategy using only a constant amount of time at each stage and a constant amount of memory beyond that required to hold the strategy's description. We also show that the length of this description is within a constant factor of optimal. Therefore, Player 1 can play nearly optimally given a limited memory while using only a constant amount of time at each stage. In Section 4.4, we provide upper and lower bounds on the value of the repeated game when Player 1 is limited to a finite number of strategies. The entropy of the optimal mixed strategy associated with Player 1's finite set of strategies figures prominently in the development of these bounds. As a result, these bounds provide insight into how strategic entropy, as defined by Neyman and Okada (1999), comes to play a role in bounding the value of repeated games in which one of the players is computationally restricted. Finally, in Section 4.5, we use these bounds to show how an approximately

optimal set can be computed in time which is linear in the size of the set.

We will use the following notation in the remainder of this chapter. A_i denotes the set of pure actions in the stage game available to Player i. S and T denote the set of pure strategies in the repeated game available to Players 1 and 2 respectively. $\Delta(S)$ is the set of mixed strategies over pure strategy set S. $|S|$ is the number of elements in set S. When Players 1 and 2 choose actions a_1 and a_2 respectively, the payoff in the stage game is denoted $r(a_1, a_2)$. $V(G)$ is the value of the stage game. $V(G^N)$ is the value of the N-stage repetition of G, i.e. $V(G^N) = \max_{\alpha \in \Delta(S)} \min_{t \in T} E_{\alpha,t} \left[\frac{1}{N} \sum_{n=1}^{N} r\left(a_1^n, a_2^n\right) \right]$ where a_i^n is the action chosen by Player i at stage n and $E_{\alpha,t}$ is the expected value given strategies α and t. We will also sometimes refer to the **optimal total expected payoff** for an N-stage game which is defined to be $\max_{\alpha \in \Delta(S)} \min_{t \in T} E_{\alpha,t} \left[\sum_{n=1}^{N} r\left(a_1^n, a_2^n\right) \right]$. G_K^N represents the N-stage repetition of G when Player 1 is restricted to mixing over K pure strategies. The adjoint of matrix M is denoted adj M while det M denotes M's determinant.

4.2 Representing an Optimal Set

Consider a finitely repeated two-person zero-sum game, G_K^N, in which Player 1 is restricted to picking a set of K pure strategies and then mixing over these strategies. Player 2 is informed of the set prior to the start of the game so there is no point in Player 1 mixing over the choice of sets. Player 2 is not restricted in any way. In this section, we show that Player 1 can play optimally by choosing a set of pure strategies that ignore Player 2's actions. We call a pure strategy that ignores the opponent's actions **oblivious**. Player 1's optimal set of K oblivious strategies can be represented by a tree with K leaves. The simple structure of this tree enables Player 1 to compute the set along with the corresponding optimal mixed strategy using $O(K)$ time when the stage game is 2×2.

Definition 4.2 *A set of pure strategies S is an* **optimal set** *for Player 1 in G_K^N if $|S| \leq K$ and there is a mixed strategy $\sigma \in \Delta(S)$ such that, for all sets of pure*

strategies X *with* $|X| \leq K$,

$$\min_{\tau \in \Delta(T)} E_{\sigma,\tau}\left[\frac{1}{N}\sum_{n=1}^{N} r\left(a_1^n, a_2^n\right)\right] \geq \max_{\gamma \in \Delta(X)} \min_{\tau \in \Delta(T)} E_{\gamma,\tau}\left[\frac{1}{N}\sum_{n=1}^{N} r\left(a_1^n, a_2^n\right)\right]$$

Suppose, for an N-stage game, Player 1 chooses a set of K oblivious pure strategies. Each strategy in the set can be described by a string of length N specifying the action to be taken at each stage. We can describe the entire set of strategies using a tree of depth N with K leaves where each leaf represents one of the strategies from the set. The labels on the edges of the path from the root to a leaf indicate the action chosen at that stage by the corresponding strategy. For example, suppose $A_1 = \{L, R\}$. The tree in Figure 4.1 describes the set $\{LLLL, LLLR, LLRL, LRLL, LRRL, RLLL, RLRL, RRLL\}$.

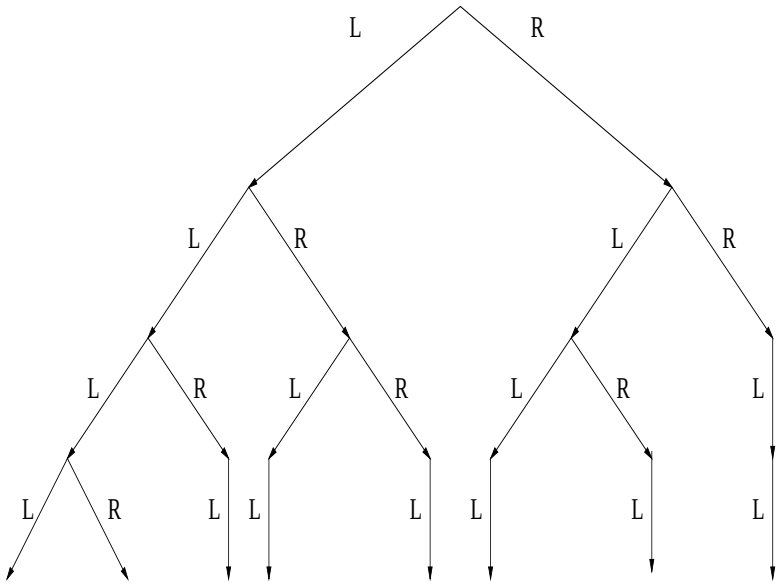

Figure 4.1: The tree representation of an eight strategy set for a 4-stage game. The set is $\{LLLL, LLLR, LLRL, LRLL, LRRL, RLLL, RLRL, RRLL\}$.

We call a node of outdegree greater than one a **fork**. Suppose, in the tree representation of Player 1's set of pure strategies, there is no path containing a node of outdegree one followed by a fork. Suppose further that every edge from the last fork to the leaf in any path is labeled with one of Player 1's stage game maxmin actions. In representing this set, we don't need to specify the actions after the last fork in a

path since it is understood that a maxmin action is played from that stage onward. Under these assumptions, every internal node in the tree will be a fork. We say that a set of pure strategies has a **simple tree representation** if it can be represented by a tree in which every internal node is a fork and it is understood that once a leaf is reached the strategy plays one of Player 1's stage game maxmin actions from that stage onward.

Theorem 4.3 *Player 1 has an optimal set of pure strategies having a simple tree representation.*

Proof. First, to see that Player 1 can ignore Player 2's actions, consider a tree representation of a non-oblivious set of pure strategies. In this tree, we represent Player 2's actions by forks with branches that are dashed lines (see Figure 4.2). Suppose there is some node in the tree underneath which is a subtree T^c if Player 2

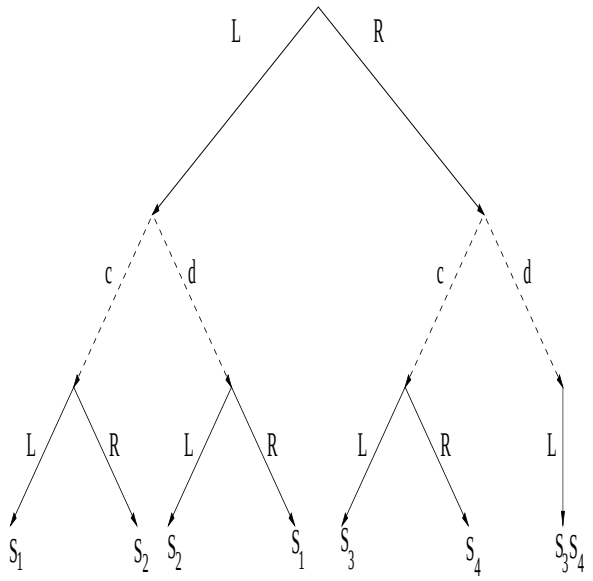

Figure 4.2: A tree representation of a non-oblivious set of strategies consisting of $S_1 = [L, ((c, L), (d, R))]$, $S_2 = [L, ((c, R), (d, L))]$, $S_3 = [R, ((c, L), (d, L))]$, and $S_4 = [R, ((c, R), (d, L))]$

chose action c at the previous stage and T^d if Player 2 chose action d (Figure 4.3). Let r_c and r_d be the expected payoffs achieved by T^c and T^d respectively under an

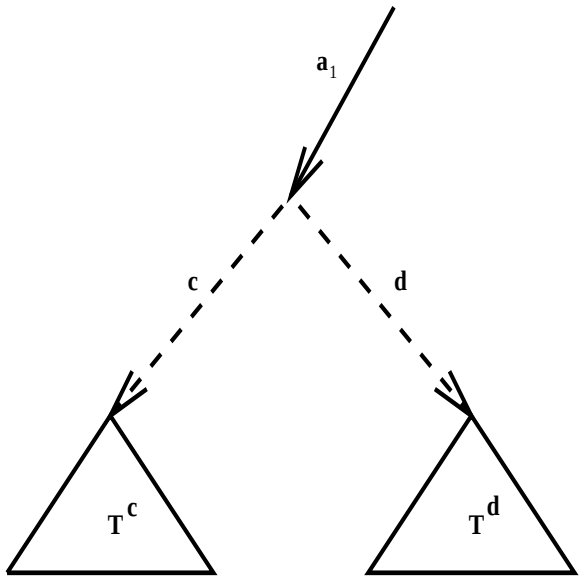

Figure 4.3: A tree showing Player 1 basing his strategy on Player 2's action.

optimal mixed strategy, ρ, for Player 1. Since Player 2 is unrestricted, Player 1 receives an expected payoff of $\min\{r_c + r(a_1, c), r_d + r(a_1, d)\}$ from this point onward. Assume without loss of generality that $r_c \geq r_d$. If Player 1 plays T^c regardless of whether Player 2 plays c or d, his payoff from this point onward is $\min\{r_c + r(a_1, c), r_c + r(a_1, d)\} \geq \min\{r_c + r(a_1, c), r_d + r(a_1, d)\}$. Let $\{s_1, \ldots, s_k\}$ be the subset of pure strategies consistent with the history of play up to the point where T^c and T^d diverge. Let $\rho_1, \ldots, \rho_k$ be the probabilities associated with these strategies by Player 1's optimal mixed strategy ρ. Since T^c is consistent with $(\rho_1, \ldots, \rho_k)$, we can replace T^d by T^c without affecting these probabilities. Furthermore, the change to the strategies s_1 through s_k that corresponds to this replacement only affects outcomes that are consistent with the current history. Thus, the expected outcome at all other points in the tree remains the same and, therefore, the total payoff associated with the tree in which T^d is replaced by T^c is at least as high as the total payoff associated with the original tree. Hence, Player 1 gains nothing by basing his strategies on Player 2's actions.

It is easy to see there is no reason to delay a fork since, for any tree which has a fork following a non-fork in a path, we can construct another tree which achieves the

same expected payoff by simply shifting the fork up one level (Figure 4.4). Finally,

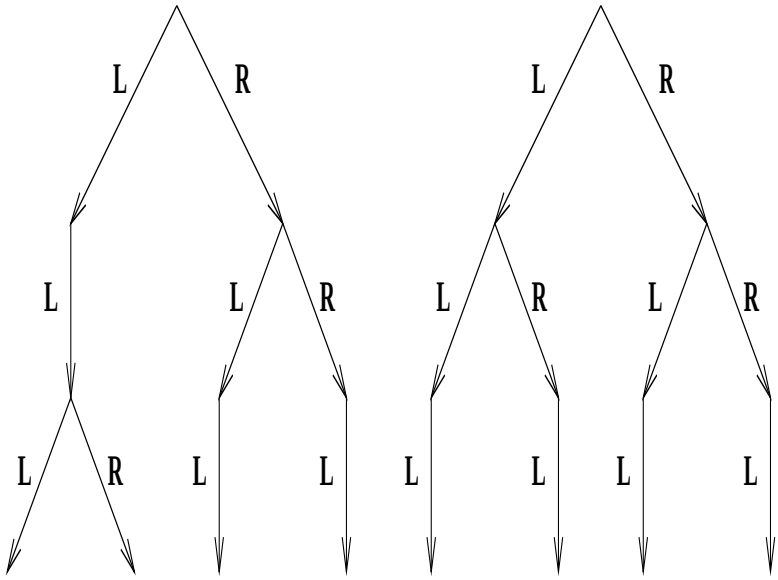

Figure 4.4: A comparison of two trees – one with and one without a delayed fork. The tree on the right achieves the same expected payoff as the tree on the left but with the fork on the leftmost path at stage 3 moved up to stage 2.

once we have passed the last fork on a path, the strategy is uniquely determined by the history of play so far. Since Player 2 has access to this history and therefore knows all of Player 1's subsequent moves, Player 1 can do no better than to play one of his maxmin actions from that point onward. Furthermore, if he has more than one maxmin action, he can play the same maxmin action at the end of any path without affecting his expected payoff. ∎

The implications of Theorem 4.3 are twofold. First, Player 1 does not benefit from spending his resources trying to remember what Player 2 has done in the past. Second, he does not benefit from delaying the point at which two strategies differ, that is, he gains nothing by concealing his chosen strategy until some point later in the game.

4.2.1 Computing an Optimal Mixed Strategy

Given that Player 1 has an optimal set with a simple tree representation, how can he compute this set along with the associated optimal mixed strategy? We can describe a mixed strategy over the set by assigning a probability to each leaf of the corresponding tree since each leaf represents one of the strategies in the set. Alternatively, we could give a behavioral strategy description by assigning probabilities to the branches at each fork. The probability of a leaf is then just the product of the probabilities along the path from the root to the leaf. In general, many different probability vectors may be used throughout the tree. However, for 2×2 games, we show that the same probability vector can be used at every fork. By making use of this homogeneity in the 2×2 case we can compute an optimal set of pure strategies along with the associated optimal mixed strategy in time which is linear in the size of the set.

2×2 games

Proposition 4.4 shows that when G is 2×2, there is an optimal set such that the corresponding optimal mixed strategy can be described by assigning the same probability vector to the branches at every fork and that this probability vector is the optimal mixed strategy for the stage game.

Proposition 4.4 *Let G be a 2×2 stage game in which $A_1 = \{a_1, a_2\}$ and $(p, 1-p)$ is an optimal mixed strategy for Player 1. For every $K \geq 1$, there is an optimal set S_K for G_K^N with associated optimal mixed strategy σ_K such that, at any fork in the simple tree representation of S_K, σ_k assigns probability p to the branch labeled a_1 and probability $(1-p)$ to the branch labeled a_2.*

Proof. Suppose we are given a simple tree representation for S_K which is an optimal set of pure strategies for G_K^N.

Case 1: The stage game has a saddle point.

In this case, by playing the same action at every stage, Player 1 can achieve an expected payoff of $V(G)$ in the repeated game. Thus, Player 1 has an optimal set of pure strategies for G_K^N containing a single element. This set can be represented by a tree consisting of a single node since it is understood that a stage game maxmin action is played at every stage after a leaf is reached.

Case 2: The stage game does not have a saddle point.

Let the payoff matrix for the stage game be $\begin{pmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{pmatrix}$. Consider Player 1's choice of action at a stage corresponding to a fork. There is an expected payoff V_1 for the remainder of the game given that he chooses action a_1 at this stage and an expected payoff V_2 given that he chooses a_2. Player 1's decision at any fork is then equivalent to the decision he faces in a 1-stage game with payoff matrix $\begin{pmatrix} r_{1,1}+V_1 & r_{1,2}+V_1 \\ r_{2,1}+V_2 & r_{2,2}+V_2 \end{pmatrix}$.

Suppose the modified matrix contains a saddle point. Assume without loss of generality that this saddle point occurs in row 2. Player 1 should then choose a_1 at this stage with probability 0. Thus no strategy starting with a_1 from this point is ever used. Let s_1 be any strategy in S_K corresponding to a leaf in the subtree under the branch for a_1. Since s_1 is chosen with probability 0, $S_K \setminus \{s_1\}$ achieves the same value as S_K and is therefore optimal for G_K^N. Thus, we can create a new tree which represents another optimal set of pure strategies for G_K^N by removing the branch for a_1 along with the subtree underneath this branch.

We can assume, therefore, that the modified payoff matrix does not contain a saddle point. Adding a different constant to each row of a 2×2 matrix does not change the optimal mixed strategy as long as no saddle point is created or eliminated. Since there is no saddle point in either the original matrix or the modified matrix, the optimal mixed strategy must be the same for both matrices. Since S_K and σ_K are optimal, at every fork, σ_K must assign probability p to the branch labeled a_1 and $1-p$ the branch labeled a_2. ∎

Proposition 4.4 implies that in a 2×2 game, Player 1, playing optimally, receives an expected payoff of $V(G)$ at each stage corresponding to a fork and receives $\max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$ at every other stage. The value of the repeated game in this case is:

$$V(G_K^N) = \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{F(K,p)}{N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \quad (4.1)$$

where $F(K,p)$ is the maximum expected number of forks in a tree with K leaves in which every fork has two branches assigned probabilities according to the stage game optimal mixed strategy $(p, 1-p)$.

Thus, for 2×2 games, we can compute S_K and σ_K by creating a tree with the maximum expected number of forks given that the actions at each fork are chosen according to $(p, 1-p)$ and that no fork occurs at a depth $\geq N$. This is easily done by starting with one leaf assigned probability 1 and repeatedly expanding the most probable leaf having depth $< N$ according to $(p, 1-p)$ until we have K leaves. The resulting tree represents an optimal set for G_K^N and the probabilities assigned to the leaves is the associated optimal mixed strategy.

An $O(K)$ time algorithm to compute this tree is given in Algorithm 4.1. We need to keep track of the nodes that are currently leaves and expand the most probable of these nodes during each of the $K-1$ iterations. We use two separate lists to keep track of the leaves. $LeftProb$ for leaves that are the left children of their parents and $RightProb$ for leaves that are the right children of their parents. During each iteration, the first elements of each of the lists are compared. The most probable of these is removed from its corresponding list, its left child is added to the end of $LeftProb$ and its right child is added to the end of $RightProb$. Since the algorithm uses $O(1)$ time in each of the $K-1$ iterations, the total running time is $O(K)$.

By keeping a separate list for left and right children, the lists remain ordered when a new element is appended. If we used only one list then new elements would have to be inserted into the middle of the list to maintain the order. Using insertion to add new elements would adversely affect the asymptotic running time of the algorithm. To see that the lists remain ordered when new elements are appended, note that if

Algorithm 4.1: ComputeOptimal(N, K, p).

/* || denotes string or list concatenation depending on the context. */

/* $x[i,..,j]$ denotes the substring of string x from the i-th through the j-th character. */

/* [] denotes the empty list. */

1. $LeftProb[1] \leftarrow p$; $LeftString[1] \leftarrow L$; $LeftDepth[1] \leftarrow 1$
2. $RightProb[1] \leftarrow 1-p$; $RightString[1] \leftarrow R$; $RightDepth[1] \leftarrow 1$
3. $FirstL \leftarrow 1$; $FirstR \leftarrow 1$; $Last \leftarrow 1$; $NumLeaves \leftarrow 2$; $S \leftarrow []$; $\sigma \leftarrow []$
4. **while** $NumLeaves < K$ **do**
5. **if** $LeftProb[FirstL] \geq Right[FirstR]$
6. $ExpandNode(LeftProb[FirstL], LeftString[FirstL], LeftDepth[FirstL], i)$
7. $FirstL \leftarrow FirstL + 1$
8. **else**
9. $ExpandNode(RightProb[FirstR], RightString[FirstR], RightDepth[FirstR], i)$
10. $FirstR \leftarrow FirstR + 1$
11. $NumLeaves \leftarrow NumLeaves + 1$
12. **end while**
13. $S \leftarrow S \;||\; LeftString[FirstL, \ldots, Last] \;||\; RightString[FirstR, \ldots, Last]$
14. $\sigma \leftarrow \sigma \;||\; LeftProb[FirstL, \ldots, Last] \;||\; RightProb[FirstR, \ldots, Last]$
15. Return(S, σ)

ExpandNode$(Prob, String, Depth, i)$

16. **if** $Depth = N - 1$
17. $S \leftarrow S \;||\; [String \;||\; L] \;||\; [String \;||\; R]$
18. $\sigma \leftarrow \sigma \;||\; [p * Prob] \;||\; [(1-p) * Prob]$
19. **else**
20. $Last \leftarrow Last + 1$
21. $LeftProb[Last] \leftarrow p * Prob$
22. $LeftString[Last] \leftarrow String \;||\; L$
23. $LeftDepth[Last] \leftarrow Depth + 1$
24. $RightString[Last] \leftarrow String \;||\; R$
25. $RightProb[Last] \leftarrow (1-p) * Prob[FirstL]$
26. $RightDepth[Last] \leftarrow Depth + 1$
27. Return.

p_i appears before p_j in the $LeftProb$ list then the parent of p_i was expanded before the parent of p_j. But then the probability of p_i's parent is at least as high as the probability of p_j's parent. Since the probability of any left child is just p multiplied by the probability of its parent, $p_i \geq p_j$. A similar argument holds for the $RightProb$ list. Therefore, each of the two lists remains ordered in decreasing order of probability.

m × n games

As the following example shows, when the stage game is not 2×2, the probability vectors used on the forks in the simple tree representation of an optimal set are not always optimal mixed strategies for the stage game and different probability vectors may be required on different forks.

Example 4.5 *Consider a stage game with payoff matrix* $\begin{pmatrix} 0 & 9 \\ 6 & 5 \\ 5 & 6 \end{pmatrix}$. *The optimal mixed strategy is* $(0, 1/2, 1/2)$ *which results in an expected payoff of 5.5. A non-optimal mixed strategy,* $(1/10, 9/10, 0)$ *results in an expected payoff of 5.4. For* $K = 3$ *and* $N = 2$*, Figure 4.5 shows that these payoffs are close enough to make it beneficial to construct the tree with* $(1/10, 9/10, 0)$ *at the root rather than* $(0, 1/2, 1/2)$*. In this case, it is better for Player 1 to give his opponent more information about his current action in order to achieve an expected increase in the uncertainty about his future actions.*

Of course, since Player 1 can guarantee $V(G)$ at every fork by using a stage game optimal mixed strategy at each fork, we have

$$V(G_K^N) \geq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{F(K,p)}{N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \qquad (4.2)$$

where $F(K,p)$ is the maximum expected number of forks in a tree with K leaves in which the branches at every fork are assigned probabilities according to the stage game optimal mixed strategy p.

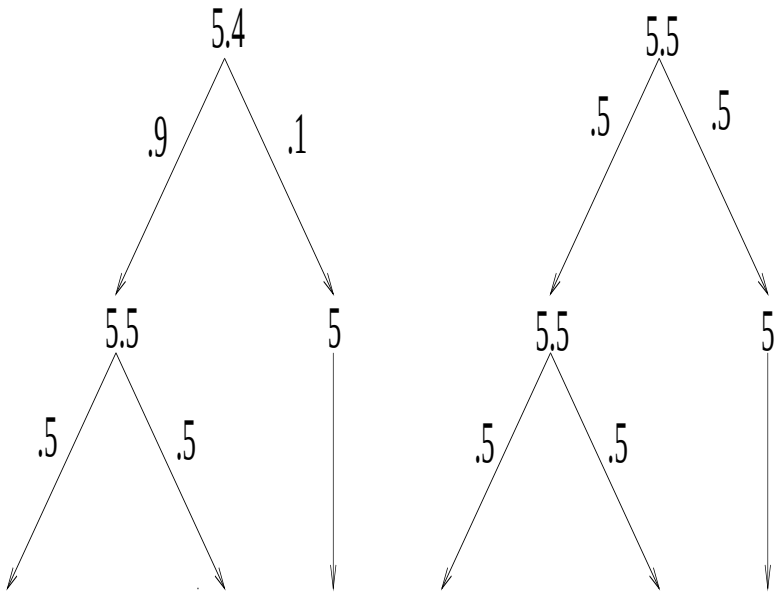

Figure 4.5: An example of an optimal tree that does not use the optimal mixed strategy at every fork. The tree on the left achieves a total expected payoff of $5.4 + .9(5.5) + .1(5) = 10.85$ while the tree on the right achieves a total expected payoff of $5.5 + .5(5.5) + .5(5) = 10.75$. The numbers on the nodes indicate the expected payoff received at the stage when the node is reached.

For games which are not 2×2, we can compute S_K and σ_K using a bottom up dynamic programming algorithm in $O(NK^{|A_1|+1})$ time. The remainder of this section describes this algorithm. Readers not interested in the details may wish to skip to the next section.

If Player 1 has K strategies allocated to some subtree then, at the root of this subtree, he must decide how many branches to include, what actions and probabilities to assign to those branches, and how many of the K strategies to allocate to the subtrees under each branch. Let $m = |A_1|$ and $n = |A_2|$. Let $G|_A$ be the stage game G with Player 1 restricted to actions $A = \{a_1, \ldots, a_l\}$. Then,

$$\forall K \geq 1, V(G_K^1) = \max_{A \subset A_1} \{V(G|_A) : |A| \leq K\} \tag{4.3}$$

$$\forall N \geq 1, V(G_1^N) = \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \tag{4.4}$$

Let $(k_1, \ldots, k_m)$ be a vector of non-negative integers such that $\sum_{i=1}^{m} k_i = K$. Let $G_K^N(k_1, \ldots, k_m)$ be the game defined by:

$$\begin{pmatrix} \frac{1}{N}(r_{11} + (N-1)V(G_{k_1}^{N-1})) & \ldots & \frac{1}{N}(r_{1|A_2|} + (N-1)V(G_{k_1}^{N-1})) \\ \vdots & \ddots & \vdots \\ \frac{1}{N}(r_{m1} + (N-1)V(G_{k_m}^{N-1})) & \ldots & \frac{1}{N}(r_{m|A_2|} + (N-1)V(G_{k_m}^{N-1})) \end{pmatrix} \tag{4.5}$$

For each $N \geq 2$ and each $K \geq 2$,

$$V(G_K^N) = \max \left\{ V\left(G_K^N(k_1, \ldots, k_m)\right) : k_i \geq 0, 1 \leq i \leq m \text{ and } \sum_{i=1}^{m} k_i = K \right\} \tag{4.6}$$

If there are N stages remaining and K strategies allocated to a fork and if $(k_1, \ldots, k_m)$ is the vector that maximizes Equation 4.6 then

(i) the fork should have s branches where s is the number of non-zero elements in $(k_1, \ldots, k_m)$

(ii) the probabilities and actions should be assigned to the branches according to any optimal mixed strategy for $G_K^N(k_1, \ldots, k_m)$

(iii) the number of strategies should be allocated to the subtree under each branch according to $(k_1, \ldots, k_m)$.

$V(G_K^N)$ can be computed in $O(NK^{m+1}mn^3)$ time since there are NK such values to compute, we can solve each game defined by the matrix above using linear programming techniques in $O(mn^3)$ time (see Papadimitriou and Stieglitz, 1982), and there are $O(K^m)$ vectors $(k_1, \ldots, k_m)$ to consider. For fixed m and n, therefore, we can compute S_K and σ_K in $O(NK^{m+1})$ time.

4.3 Executing the optimal strategies

In the previous section, we considered the problem of finding an optimal set of pure strategies. In this section, we address the problem of executing the strategies in an optimal set. Recalling our discussion from the introduction, we would like to determine the amount of memory needed to execute these strategies using only a constant amount of time at each stage. Imagine a scenario of the following form. Given an optimal set S_K of size K and an associated optimal mixed strategy σ_K, a mixing machine chooses a pure strategy from S_K according to σ_K. It then passes a description of this strategy to a game playing machine which executes the strategy. We say a game playing machine **efficiently executes** a pure strategy if the amount of time used at each stage and the total amount of memory used during the entire game beyond the memory needed to store the strategy's description are independent of both the length of the game and the number of strategies available to the player.

Since we have K pure strategies to describe, we can assign each a unique number between 0 and $K-1$. This encoding requires $\log_2 K$ bits which is the minimum number of bits needed to describe K objects. However, at each stage, our game playing machine would require more than a constant amount of time to decode the description and determine the action to be taken.

Since the strategies in S_K are oblivious, we can always describe a strategy literally by listing the actions it prescribes at each stage. In that case, our game playing machine does nothing more than read the next action from the description and play it. Since $\log_2(|A_1|)$ bits are required to describe a single action, Lemma 4.6 implies

that the length of this description is no more than $\lceil \frac{(\log_2 |A_1|)(\log_2 K)}{-\log_2 \hat{p}} \rceil$ where $\hat{p}$ is the largest probability assigned to any branch in the tree.

Lemma 4.6 *The maximum depth of any leaf in the tree representation of S_K is $\leq \lceil \frac{\log_2 K}{-\log_2 \hat{p}} \rceil$ where $\hat{p}$ is the largest probability assigned to any branch. Let q_i be the smallest probability used at fork i and let $\hat{q} = \max\{q_i : i \text{ is a fork}\}$. The minimum depth of any leaf in the tree is $\leq \lceil \frac{\log_2 K}{-\log_2 \hat{q}} \rceil$.*

Proof. Whenever a node has probability $\leq \frac{1}{K}$, it must be a leaf. Let x be the depth of any node. The probability of reaching that node is $\leq \hat{p}^x$. If $x \geq -\frac{\log_2 K}{\log_2 \hat{p}}$, then $\hat{p}^x \leq \frac{1}{K}$ so this node must be a leaf. The maximum depth is therefore $\leq \lceil -\frac{\log_2 K}{\log_2 \hat{p}} \rceil$.

Assume that the actions are assigned to the branches at each fork so that the rightmost branch has the lowest probability. Then the rightmost path in the tree has the smallest depth. Let y be the depth of a node on the rightmost path. The probability of that node is $\leq \hat{q}^y$. If $y \geq -\frac{\log_2 K}{\log_2 \hat{q}}$, then $\hat{q}^y \leq 1/K$ so this node must be a leaf. Therefore, the minimum depth is $\leq \lceil -\frac{\log_2 K}{\log_2 \hat{q}} \rceil$. ∎

Since $\hat{p}$ is independent of K and N, the description length required by our game playing machine is a constant factor times the minimum. However, since $\lim_{p \to 1} \frac{1}{-\log_2 p} = \infty$, as p gets close to 1, the constant becomes very large although it is relatively small even for p as large as 0.9 (Figure 4.6).

We see, then, that there is tradeoff between the amount of space needed to describe the strategies and the amount of time taken at each stage to decode the description. We expect there to be encodings for S_K that require description lengths somewhere between $\log_2 K$ and $\lceil \frac{(\log_2 |A_1|)(\log_2 K)}{-\log_2 \hat{p}} \rceil$ such that the strategies can be efficiently executed. For example, Proposition 4.7 shows that, when $|A_1| = 2$, we can get reasonably close to the minimum description length regardless of the magnitude of the probabilities. It remains to be seen whether a general encoding scheme exists for games with $|A_1| > 2$ such that the description length does not depend on $\hat{p}$ and yet the action to be taken can be determined in a constant amount of time at each stage.

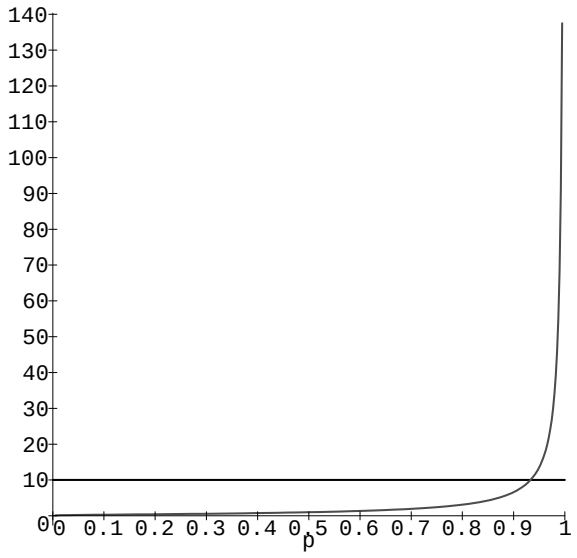

Figure 4.6: A graph of $-\frac{1}{log_2 p}$ Even for p as large as 0.9 the function is below 10.

Proposition 4.7 *If $|A_1| = 2$ then, for sufficiently large K, we can encode the optimal set of size K using at most $5.4 \log_2 K + 2$ bits. Furthermore, there is a game playing machine that, given a description of a pure strategy under this encoding scheme, can efficiently execute the strategy.*

Proof. We can encode a strategy from S_K by specifying in binary the number of L's played before the next R. For example, the description of strategy $LLLLLLLRRLR$ would be $< 111 >< 000 >< 001 >$. However, to ensure that the description can be decoded in constant time, the length of each segment of the encoding must be constant. Let $d = \lfloor \frac{\log_2 (1-p)}{\log_2 p} \rfloor + 1$ where $(p, 1-p)$ is the optimal mixed strategy for the stage game. (If there is more than one strategy to consider at each fork, let $(p, 1-p)$ be the one that is most skewed.) $d-1$ is the maximum number of times that the tree can be expanded to the left before having to expand a node to the right. We will encode the number of L's, up to d, before the next R (See Figure 4.7).

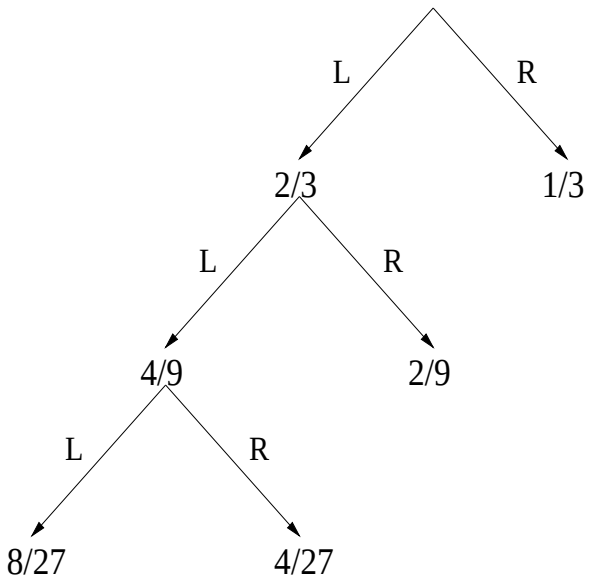

Figure 4.7: The tree representation of S_3 when $(p, 1-p) = (2/3, 1/3)$. The labels on the nodes indicate the probability of reaching the node. Since $8/27 < 1/3$, the next leaf to expand is the right child of the root. Here $d = 3$, so the description of strategy $LLLLLLLRRLR$ would be $< 11 >< 11 >< 01 >< 00 >< 01 >$.

We need $\lceil \log_2(d) \rceil \leq \log_2(d) + 1$ bits to encode d L's. By Lemma 4.6 there are at most $\lceil \frac{\log_2 K}{-\log_2 p} \rceil$ left moves and at most $\lceil \frac{\log_2 K}{-\log_2 (1-p)} \rceil$ right moves in any path. In the worst case, each right move will be encoded using $\log_2(d) + 1$ bits while every d left moves will be encoded using $\log_2(d) + 1$ bits. Assume $K \geq d$. Then the length of the description in the worst case is bounded above by

$$(\log_2(d) + 1) \left(\lceil \frac{\log_2 K}{-d \log_2(p)} \rceil + \lceil \frac{\log_2 K}{-\log_2(1-p)} \rceil \right) \tag{4.7}$$

$$\leq (\log_2(d) + 1) \left(-\frac{\log_2 K}{d \log_2(p)} - \frac{\log_2 K}{\log_2(1-p)} \right) + 2 (\log_2(d) + 1) \tag{4.8}$$

$$\leq \left((\log_2(d) + 1) \left(-\frac{\log_2 K}{d \log_2(p)} - \frac{\log_2 K}{\log_2(1-p)} \right) \right) + 2 \log_2 K + 2 \tag{4.9}$$

$$= \left(\left((\log_2(d) + 1) \left(-\frac{1}{d \log_2(p)} - \frac{1}{\log_2(1-p)} \right) \right) + 2 \right) (\log_2 K) + 2 \tag{4.10}$$

Figure 4.8 shows a graph of the minimum of this coefficient and $\frac{1}{-\log_2 p}$. It can be seen from the figure that this minimum value is at most 5.4. ∎

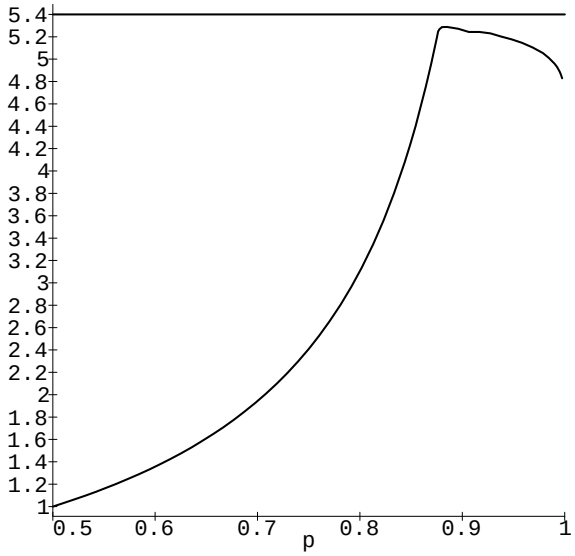

Figure 4.8: A graph of $min(\frac{1}{-\log_2(p)}, ((\log_2(d) + 1)((-\frac{1}{d \log_2(p)}) - \frac{1}{\log_2(1-p)})) + 2)$.

Combining these results with the results of the previous section we see that, for 2×2 games, Player 1 needs $O(K)$ time to produce an optimal set and then $O(\log_2 K)$ space to execute the strategies from this set using only a constant amount of time at each stage. In the general case, Player 1 still needs only $O(\log_2 K)$ space for execution but the constant hidden in the $O(.)$ notation may be quite large depending on the probabilities assigned to the forks. Player 1 may need to use a rather time consuming dynamic programming algorithm to produce an optimal set for games which are not 2×2. However, in Section 4.5, we provide an algorithm to produce an approximately optimal set in $O(K)$ time. First we need to develop upper and lower bounds on $V(G_K^N)$.

4.4 Bounding the Value of the Game

In this section, we develop upper and lower bounds on $V(G_K^N)$ based on an analysis of the simple tree representation of S_K. When $N \leq \frac{\log_2 K}{\log_2 |A_1|}$, Player 1 can create a complete tree of depth N having K leaves where each fork in the tree has $|A_1|$ branches. By assigning probabilities to each fork according to a stage game optimal mixed strategy, Player 1 can achieve an average expected payoff equal to $V(G)$ which is the value of the repeated game in this case. Our work is focused on understanding the other extreme where N is so large compared to K that Player 1 can create an optimal tree with K leaves without any fork in the tree occurring at a depth greater than or equal to N. The value of the game in the intermediate case will obviously fall somewhere in between the values in the two extreme cases. The upper bound that we present is independent of the relationship between N and K while the lower bound requires N to be large enough that we do not need to be concerned with forks occurring beyond the end of the game.

Our analysis makes use of the entropy function which measures the uniformity of a probability vector. (See Cover and Thomas (1991) for details on the properties of entropy.)

Definition 4.8 *The* **entropy** *of a probability vector* $(p_1, p_2, ..., p_m)$ *is defined to be*

$$H(p_1, p_2, ..., p_m) \quad = \quad -\sum_{i=1}^{m} p_i \log_2 p_i \tag{4.11}$$

As indicated by Equation's 4.1 and 4.2, $V(G_K^N)$ is closely related to the expected number of forks in the simple tree representation of S_K. By developing bounds on the expected number of forks we can develop bounds on $V(G_K^N)$. Proposition 4.9 and its corollaries demonstrate how the expected number of forks is related to the entropy of the probability vector used at each fork and the entropy of the corresponding probability distribution over the leaves.

Proposition 4.9 *Suppose we have a tree with* $t \geq 1$ *forks. Let* ρ^t *be the probability vector over the leaves in this tree. Let* $\{p^1, \ldots, p^m\}$ *be the set of probability vectors used at each of the* t *forks. Let* x^i *be the expected number of forks that use probability vector* p^i, $1 \leq i \leq m$. *Then,*

$$H(\rho^t) \quad = \quad x^1 H(p^1) + \ldots + x^m H(p^m) \tag{4.12}$$

Proof. The proof follows by induction on the number of forks t.

Basis: $t = 1$

Let p^j be the probability vector over the branches at the root which is the only fork. Then $H(\rho^1) = H(p^j) = H(p^j)x^j = \sum_{i=1}^{m} x^i H(p^i)$ since $x^j = 1$ and $x^i = 0, i \neq j$.

Induction Hypothesis: Assume, for some $t \geq 1$, all trees with t forks have the property that $H(\rho^t) = x^1 H(p^1) + \ldots + x^m H(p^m)$.

Suppose we have a tree with $t + 1$ forks. Number the forks in order from top to bottom and left to right. Let b_i be the probability of the i-th fork, $1 \leq i \leq t + 1$. By removing the last fork, numbered $t + 1$, we have a new tree with t forks and b_{t+1} as the probability of one of its leaves. Let $\rho^t = (q_1, \ldots, q_l = b_{t+1})$ be the probability vector over the leaves in this tree. Let $p^j = (p_1^j, \ldots, p_n^j)$ be the probability vector used to expand the fork numbered $t+1$. Then $\rho^{t+1} = (q_1, \ldots, q_{l-1}, p_1^j b_{t+1}, \ldots, p_n^j b_{t+1})$

must be the probability vector over the leaves in the tree with $t+1$ forks. Thus,

$$\begin{aligned} H(\rho^{t+1}) &= -\sum_{i=1}^{n} p_i^j b_{t+1} \log_2(p_i^j b_{t+1}) - \sum_{j=1}^{l-1} q_j \log_2(q_j) && (4.13) \\ &= -b_{t+1} \sum_{i=1}^{n} p_i^j \log_2(p_i^j) - b_{t+1} \log_2(b_{t+1}) - \sum_{j=1}^{l-1} q_j \log_2(q_j) && (4.14) \\ &= b_{t+1} H(p^j) + H(\rho^t). && (4.15) \end{aligned}$$

Then by the induction hypothesis,

$$H(\rho^{t+1}) = b_{t+1} H(p^j) + x_t^1 H(p^1) + \ldots + x_t^m H(p^m). \quad (4.16)$$

where x_t^i is the expected number of forks using p^i in the tree with t forks, $1 \leq i \leq m$. Since $x_{t+1}^j = x_t^j + b_{t+1}$ and $x_{t+1}^i = x_t^i$ for all $i \neq j$, $H(\rho^{t+1}) = x_{t+1}^1 H(p^1) + \ldots + x_{t+1}^m H(p^m)$. Thus the proposition is true for all $t \geq 1$. ∎

Corollary 4.10 *Let $f(t)$ be the expected number of forks in a tree with t forks. Let ρ^t be the probability vector over the leaves in this tree. Let Ψ be the set of all probability vectors used on the branches of the t forks. Let $\hat{H} = \max_{p \in \Psi} H(p)$ and $\hat{h} = \min_{p \in \Psi} H(p)$. Then,*

$$\frac{H(\rho^t)}{\hat{H}} \leq f(t) \leq \frac{H(\rho^t)}{\hat{h}} \quad (4.17)$$

Proof. Using the notation from Proposition 4.9 we have $f(t) = x^1 + \ldots + x^m$ and $H(\rho^t) = \sum_{i=1}^{m} x^i H(p^i)$. Since $H(p^i) \geq \hat{h}$, for $1 \leq i \leq m$, we have $H(\rho^t) \geq \hat{h} \sum_{i=1}^{m} x^i$. Therefore, $f(t) \leq \frac{H(\rho^t)}{\hat{h}}$. The left hand inequality follows by a similar argument. ∎

Corollary 4.11 *If a tree has t forks and each of the forks has m branches assigned probabilities $p_1, \ldots, p_m$, then*

$$f(t) = \frac{H(\rho^t)}{H(p_1, \ldots, p_m)} \quad (4.18)$$

where ρ^t is the probability vector over the leaves.

4.4.1 Upper Bound

We are now in position to develop an upper bound on $V(G_K^N)$.

Theorem 4.12 *For each $N \geq 1$, let σ_K be the optimal mixed strategy associated with the optimal set S_K for G_K^N. If G is 2×2,*

$$V(G_K^N) = \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{H(\sigma_K)}{H(p)N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \quad (4.19)$$

where p is Player 1's stage game optimal mixed strategy.

If G is $m \times n$, for m and $n \geq 2$,

$$V(G_K^N) \leq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{H(\sigma_K)}{\hat{h}N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \quad (4.20)$$

$$\leq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{\log_2 K}{\hat{h}N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \quad (4.21)$$

where $\hat{h}$ is as defined in Corollary 4.10.

Proof. According to Proposition 4.4, when G is 2×2, the branches at each fork will be assigned probabilities according to the stage game optimal mixed strategy $(p, 1-p)$. In this case, Corollary 4.11 implies Equation 4.19 since Player 1 receives an expected payoff equal to $V(G)$ at every fork.

In the $m \times n$ case, at every fork Player 1 receives an expected payoff not exceeding $V(G)$ so, by Corollary 4.10,

$$V(G_K^N) \leq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) + \frac{H(\sigma_K)}{\hat{h}N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \quad (4.22)$$

We know from information theory (see Cover and Thomas, 1991) that $H(\sigma_K) \leq \log_2 K$, so Equation 4.21 follows from Equation 4.20. ∎

Neyman and Okada (1999) also used a version of entropy to bound the value of repeated games in which one of the players is computationally restricted. They defined an extension to entropy called strategic entropy and use it to prove their results regarding repeated games with finite automata.

Definition 4.13 *(Neyman and Okada, 1999) The* **strategic entropy**, $H_N^*(\sigma)$, *of a mixed strategy σ for Player 1 after N stages is defined by:*

$$H_N^*(\sigma) = \max_{t \in T} \left\{ - \sum_{\omega \in \Omega_{N+1}} P_{\sigma,t}(\omega) \log_2 P_{\sigma,t}(\omega) \right\} \tag{4.23}$$

where T is the set of pure strategies available to Player 2, Ω_N denotes the set of possible histories of length $N-1$ and $P_{\sigma,t}(\omega)$ is the probability of history ω given that Players 1 and 2 use strategies σ and t respectively.

Neyman and Okada (1999) show that if the strategic entropy of Player 1's mixed strategy is bounded by a function which is $o(N)$, then the value of the repeated game converges to Player 1's stage game maxmin value as N approaches infinity.

Since the strategies over which σ_K is defined are oblivious, $H(\sigma_K) = H_N^*(\sigma_K)$. Thus, Equation 4.20 implies a direct relationship between strategic entropy and the value of the repeated game when Player 1 is restricted to mixing over a finite number of pure strategies and shows that strategic entropy arises naturally in the analysis of repeated games with finite strategy sets.

No matter what computational model one chooses, bounding a player's available resources restricts him to a finite number of pure strategies. Therefore, regardless of the model chosen, if K is the number of strategies that satisfy the resource bounds for that particular model, Equation 4.21 gives an upper bound on the value of the game. For example, if we choose the finite automaton as our computational model and restrict Player 1 to automata of size $m(N)$ then we have $K \leq |A_1|^{m(N)} m(N)^{|A_2| m(N)}$. Equation 4.21 then implies

$$\begin{aligned} V(G_K^N) \leq & \frac{m(n)(|A_2| \log_2 m(N) + \log_2 |A_1|)}{\hat{h} N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \\ & + \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \end{aligned} \tag{4.24}$$

which leads to an alternative proof of Neyman and Okada's result regarding finite automata (i.e. if $m(N) \log_2(m(N)) = o(N)$ then $\lim_{N \to \infty} V(G_K^N) = \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$).

In general, if we let $K(N)$ be the number of pure strategies available to Player 1 and consider the value of the repeated game as N goes to infinity, we have the following

result:

Proposition 4.14 *If* $\lim_{N\to\infty} \frac{\log_2 K(N)}{N} = 0$ *then* $\lim_{N\to\infty} V(G_K^N) = \max_{a_1\in A_1} \min_{a_2\in A_2} r(a_1, a_2)$

Proof. This follows immediately from Equation 4.21. It is also implied by the strategic entropy theorem in Neyman and Okada (1999). ∎

4.4.2 Lower Bound

Suppose Player 1 creates a tree using the same stage game optimal mixed strategy p at each fork. Corollary 4.11 indicates that we can find a lower bound for $V(G_K^N)$ by finding a lower bound for the entropy over the leaves in this tree. As we saw in Section 4.2, Player 1 can create this tree starting with one node and repeatedly expanding the most probable leaf using probability distribution p for each expansion. In this process, each successive probability vector over the leaves should be more uniform than its predecessor since the largest probability is being split up into multiple elements with smaller probabilities. Intuitively, one would expect the entropy of this vector to be moving toward $\log_2 K$ or at least not moving away from it. In fact, as the following lemma shows, the difference between $\log_2 K$ and the entropy over the leaves is bounded above by a constant which of course depends on the probability vector used to expand each node.

Lemma 4.15 *Let* $(p_1, \ldots, p_m)$ *be any probability vector such that* $p_i \geq p_{i+1}$ *for* $1 \leq i \leq m-1$. *By starting with the root and repeatedly expanding the most probable leaf, create a tree with* K *leaves in which each fork has* m *branches assigned probabilities* $p_1, \ldots, p_m$. *Let* $q_1, \ldots, q_K$ *be the probability vector over the leaves in this tree. Then*

$$H(q_1, \ldots, q_K) \geq \log_2 K + \log_2 p_m \tag{4.25}$$

Proof. Let y be the probability of the last node that was expanded. Suppose $p_m y > q_i$ for some $i, 1 \leq i \leq K$. Let z be the probability of leaf i's parent. Then $q_i \geq p_m z$ so $y > z$. But then the node with probability y should have been expanded before the

node with probability z contradicting the assumption that y is the probability of the last node that was expanded. Thus $p_m y$ is the probability of the least probable leaf and is therefore $\leq \frac{1}{K}$.

Let $x = \max_{1 \leq i \leq K} q_i$. Since a node with probability y was expanded and there is a leaf with probability x, $x \leq y$. Therefore, $p_m x \leq p_m y \leq \frac{1}{K}$. So for $1 \leq i \leq K, q_i \leq \frac{1}{p_m K}$ which implies $-\log_2 q_i \geq \log_2 K + \log_2 p_m$. Consequently,

$$\begin{aligned} H(q_1, \ldots, q_K) &= \sum_{i=1}^{K} q_i(-\log_2 q_i) && (4.26) \\ &\geq \sum_{i=1}^{K} q_i(\log_2 K + \log_2 p_m) && (4.27) \\ &= \log_2 K + \log_2 p_m && (4.28) \end{aligned}$$

∎

Lemma 4.15 implies the following lower bound on $V(G_K^N)$:

Theorem 4.16 *Let $(p_1, \ldots, p_m)$ be any optimal mixed strategy for G and assume $p_i \geq p_{i+1}$ for $1 \leq i \leq m-1$. Let $c \geq 0$ be the smallest integer such that $K - c = t(m-1)+1$ for some integer $t > 0$. If $N \geq \lceil \frac{\log_2 (K-c)}{-\log_2 p_1} \rceil$ then*

$$\begin{aligned} V(G_K^N) \geq\ & \frac{\log_2 (K-c) + \log_2 p_m}{H(p_1, \ldots, p_m) N} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) \\ & + \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \qquad (4.29) \end{aligned}$$

Proof. By Lemma 4.6, if $N \geq \lceil \frac{\log_2 (K-c)}{-\log_2 p_1} \rceil$ then the tree with the maximum expected number of forks given that $(p_1, \ldots, p_m)$ is assigned to every fork contains no leaf at a depth $> N$. Hence, Equation 4.29 is an immediate consequence of Lemma 4.15. (The parameter c is needed since it may not be possible to create a tree with K leaves having m branches at each fork.) ∎

This lower bound also gives us a lower bound for games where one player's strategic entropy is limited rather than the size of his set of pure strategies. Let G_η^N be the N-stage repetition of G when Player 1 is limited to mixed strategies with entropy at most η. Since this set of mixed strategies may not be convex the game may not have a

value as defined by the Minimax Theorem. Let $W(G_\eta^N) = \max_{\sigma \in \Delta_\eta(A_1)} \min_{a_2 \in A_2} E_\sigma[r(a_1, a_2)]$ where $\Delta_\eta(A_1)$ is the set of mixed strategies over A_1 with entropy at most η. This is Player 1's maxmin value in mixed strategies with entropy at most η.

Proposition 4.17 *Let $p = (p_1, \ldots, p_m)$ be any optimal mixed strategy for G. Assume $p_i \geq p_{i+1}$ for $1 \leq i \leq m-1$. Let η be the bound on strategic entropy. Let $K = \lfloor 2^\eta \rfloor$. Let $c \geq 0$ be the smallest integer such that $K - c = t(m-1) + 1$ for some integer $t > 0$. If $N \geq \lceil \frac{\log_2 (K-c)}{-\log_2 p_1} \rceil$ then*

$$W(G_\eta^N) \geq \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \tag{4.30}$$

$$+ \frac{(\log_2 (K-c) + \log_2 p_m)\,(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2))}{H(p)N} \tag{4.31}$$

Proof. Since $K = \lfloor 2^\eta \rfloor$, $H(\sigma_K) \leq \log_2 K \leq \eta$. In other words, the entropy of the optimal mixed strategy σ_K corresponding to an optimal set of size K is below the strategic entropy bound. Theorem 4.16, therefore, also provides a lower bound on $W(G_\eta^N)$. ∎

Neyman and Okada (2000) provide upper and lower bounds on $W(G_\eta^N)$ which have a very different form than Equation 4.31. Whether relating these bounds can provide any additional insights is an open question.

While the upper bounds given in Theorem 4.12 hold with K equal to the number of strategies which can be implemented within Player 1's computational constraints, this is not true of the lower bound given in Theorem 4.16. For example, let S_K^m be the set of pure strategies used in the proof of Lemma 4.15 and suppose Player 1 is restricted to finite automata of size $\log_2 K$. Player 1 can certainly implement more than K strategies but he cannot implement all the strategies in S_K^m unless $p_1 \leq 1/2$. However, if K' is the largest integer such that Player 1 can implement all the strategies in $S_{K'}^m$ given his resource bounds, then the lower bound holds with $K = K'$. Again returning to the finite automaton example, if Player 1 is allowed to use automata with at least $\lceil \frac{\log_2 K}{-\log_2 p_1} \rceil$ states then he can implement all the strategies in S_K^m and Equation 4.29 provides a lower bound on the value of the repeated game.

Evaluating our lower bound in the limit where K is a function of N we have the following proposition:

Proposition 4.18 *Let $(p_1, \ldots, p_m)$ be any optimal mixed strategy for G and assume $p_i \geq p_{i+1}$ for $1 \leq i \leq m-1$. If $N \geq \lceil \frac{\log_2 K(N)}{-\log_2 p_1} \rceil$ and $\lim_{N\to\infty} \frac{\log_2 K(N)}{N} = l$ for some constant $l > 0$ then*

$$\lim_{N\to\infty} V(G_K^N) \geq \frac{l}{H(p_1, \ldots, p_m)} \left(V(G) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \right) + \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) \tag{4.32}$$

Proof. This follows immediately from Theorem 4.16. ∎

It is tempting to conclude from Proposition 4.18 that if $l \geq H(p_1, \ldots, p_m)$ then $\lim_{N\to\infty} V(G_K^N) = V(G)$. However, the restriction that $N \geq \lceil \frac{\log_2 K}{-\log_2 p_1} \rceil$ implies that $-\log_2 p_1 \geq l$. But $-\log_2 p_1 \leq H(p_1, \ldots, p_m)$ for $(1/m) \leq p_1 \leq 1$ with equality holding only when $p_1 = (1/m)$ or $p_1 = 1$. These two cases are already trivial. To determine when $V(G)$ converges to $V(G_K^N)$, further work needs to be done on the case where $\frac{\log_2 K}{\log_2 |A_1|} < N < \lceil \frac{\log_2 K}{-\log_2 p_1} \rceil$.

Another implication of Theorem 4.16 is that an approximately optimal tree can be created by starting at the top and repeatedly allocating the strategies to each subtree according to $(\lceil pK \rceil, \lfloor (1-p)K \rfloor)$ when $(p, 1-p)$ is an optimal mixed strategy for Player 1 in the stage game. The following proposition shows that this provides a total payoff that is within an additive constant of the optimal:

Proposition 4.19 *Let $(p, 1-p)$ be an optimal mixed strategy for Player 1 in G. Create a tree by allocating $\lceil pK \rceil$ strategies to the left subtree and $\lfloor (1-p)K \rfloor$ strategies to the right subtree and then build these subtrees using Algorithm 4.1. If $N \geq \lceil \frac{\log_2 K}{-\log_2 (1-p)} \rceil$ and K is sufficiently large, the resulting tree achieves a total expected payoff that is within an additive constant of the optimal.*

Proof. Choose $\epsilon > 0$ such that $\frac{1}{2+\epsilon} \geq 1-p$. Let K_0 be large enough that $\lfloor (1-p)K_0 \rfloor \geq \frac{K_0}{2+\epsilon}$ and let K be greater than K_0. Let γ be the mixed strategy created as described

by allocating $\lceil pK \rceil$ leaves under the left branch and $\lfloor (1-p)K \rfloor$) under the right branch and then building the left and right subtrees according to Algorithm 4.1. By Corollary 4.11,

$$\frac{H(\gamma)}{H(p)} = p\frac{H(\sigma_{\lceil pK \rceil})}{H(p)} + (1-p)\frac{H(\sigma_{\lfloor (1-p)K \rfloor})}{H(p)} + 1 \tag{4.33}$$

By Theorem 4.16 we have,

$$\begin{aligned}
H(\gamma) &= pH(\sigma_{\lceil pK \rceil}) + (1-p)H(\sigma_{\lfloor (1-p)K \rfloor}) + H(p) \\
&\geq p\left(\log_2(\lceil pK \rceil) + \log_2(1-p)\right) \\
&\quad +(1-p)\left(\log_2(\lfloor (1-p)K \rfloor) + \log_2(1-p)\right) + H(p) \qquad (4.34) \\
&\geq p\log_2(pK) + p\log_2(1-p) \\
&\quad +(1-p)\log_2(\frac{K}{2+\epsilon}) + (1-p)\log_2(1-p) + H(p) \\
&= p\log_2(p) + p\log_2 K + p\log_2(1-p) + (1-p)\log_2 K \\
&\quad +(1-p)\log_2\left(\frac{1}{2+\epsilon}\right) + (1-p)\log_2(1-p) + H(p) \\
&= \log_2 K + (1-p)\log_2\frac{1}{2+\epsilon} + p\log_2(1-p) \\
&\geq \log_2 K + (1-p)\log_2(1-p) + p\log_2(1-p) \\
&= \log_2 K + \log_2(1-p) \qquad (4.35)
\end{aligned}$$

Therefore, for all $K \geq K_0$ allocating $\lceil pK \rceil$ strategies to the left subtree and $\lfloor (1-p)K \rfloor$ strategies to the right subtree creates a tree with an expected number of forks $\geq \frac{\log_2 K + \log_2(1-p)}{H(p)}$ which is within an additive constant of optimal. ∎

Since the lower bound on the expected number of forks implied by Equation 4.35 is the same as the lower bound from Theorem 4.16 used to obtain Equation 4.34, a simple induction on K shows that the lower bound holds for any tree created by repeatedly allocating $\lceil pK \rceil$ strategies to the left subtree and $\lfloor (1-p)K \rfloor$ strategies to the right subtree until the number of strategies in the subtree is smaller than K_0. Once the number of strategies is less than K_0, Algorithm 4.1 can be used to build the subtree.

4.5 Approximating the Optimal Set

In Section 4.2, we showed that, when G is not 2×2, the optimal mixed strategy does not necessarily assign the same probability vector to every fork. In this section, we investigate how well Player 1 can do when he restricts himself to mixed strategies that do assign the same probability vector to every fork. In particular, we show that Player 1 can achieve a total expected payoff that is within an additive constant of the optimal total expected payoff for G_K^N. Once Player 1 determines which one probability vector is best to use, he can compute his set in $O(K)$ time using Algorithm 4.1 generalized to handle forks with more than two branches[1]. We show that Player 1 can find an appropriate probability vector in time which is independent of K and N and, therefore, can produce an approximately optimal set in $O(K)$ time.

In choosing a probability vector for a fork, Player 1 must consider both how the probability vector affects the expected payoff at that fork and how it affects the expected number of forks below that fork. We define the gain of a mixed strategy p as the expected amount above the pure maxmin payoff that the player receives by playing mixed strategy p in the one-stage game. Intuitively, it represents the effect that assigning p to a fork has on the expected payoff at that fork.

Definition 4.20 *The* **gain**, $g(p)$, *of a mixed strategy p for Player 1 in a 1-stage game G is defined by:*

$$g(p) = \left(\min_{a_2 \in A_2} \sum_{a_1 \in A_1} p(a_1) r(a_1, a_2) \right) - \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$$

Similarly, the gain of a mixed strategy α for Player 1 in an N-stage game G^N is defined by:

$$g^N(\alpha) = \left(\min_{\tau \in \Delta(T)} E_{\alpha,\tau} \left[\sum_{n=1}^{N} r(a_1^n, a_2^n) \right] \right) - N \max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$$

[1]We can generalize the algorithm to the case where there are $m > 2$ branches at each fork by using m lists to keep track of the leaves.

We also need a way to measure the effect that assigning p to a fork has on the expected number of forks below that fork. Note that the expected number of forks in a subtree is the expected number of remaining stages in which Player 2 is uncertain about which pure strategy Player 1 is playing. Therefore, the less the expected number of forks below a fork, the more information Player 1's action at that fork reveals. Taking an information theoretic view of entropy, we can think of $H(p)$ as representing the average number of bits of information revealed by a single sample drawn according to probability vector p. Loosely equating these two notions of information, we compare mixed strategies based on their "gain per bit", $\frac{g(p)}{H(p)}$.

We have the following upper and lower bounds on the total expected payoff that Player 1 can achieve by using probability vector p at every fork:

Lemma 4.21 *Suppose Player 1 uses Algorithm 4.1 to create a mixed strategy σ_K^p with probability vector $p = (p_1, \ldots, p_m)$ assigned to every fork. Assume $p_i \geq p_{i+1}$ for $1 \leq i \leq m-1$. Let $c \geq 0$ be the smallest integer such that $K - c = t(m-1) + 1$ for some integer $t > 0$. Then,*

$$g^N(\sigma_K^p) \leq \frac{g(p)}{H(p)} \log_2 K \tag{4.36}$$

and if $N \geq \lceil \frac{\log_2 (K-c)}{-\log_2 p_1} \rceil$ then

$$g^N(\sigma_K^p) \geq \frac{g(p)}{H(p)} \left(\log_2 (K-c) + \log_2 p_m \right) \tag{4.37}$$

Proof. At each fork the gain achieved is $g(p)$ and at every non-fork the gain is 0. The lemma then follows from the same arguments used to prove Theorems 4.12 and 4.16. ∎

Using these bounds we obtain the following result:

Theorem 4.22 *By using the same probability vector at every fork, Player 1 can achieve a total expected payoff in G_K^N which is at most an additive constant below the optimal total expected payoff assuming N is sufficiently large with respect to K.*

Proof. Let $\{p^1, \ldots, p^l\}$ be the set of probability vectors used on the forks in the simple tree representation of some optimal set S_K. Assume the p^i are ordered in decreasing order of their gain per bit, i.e $\frac{g(p^i)}{H(p^i)} \geq \frac{g(p^{i+1})}{H(p^{i+1})}$ for $1 \leq i < l$. The expected number of forks in this tree is $x^1 + \ldots + x^l$ where x^i is the expected number of forks which use probability vector p^i, $1 \leq i \leq l$. If σ_K is the optimal mixed strategy associated with S_K then the optimal gain is $g^N(\sigma_K) = x^1 g(p^1) + \ldots + x^l g(p^l)$. From Proposition 4.9 we know

$$x^1 H(p^1) + \ldots + x^l H(p^l) = H(\sigma_K) \leq \log_2 K$$

which implies

$$x^1 \leq \frac{log_2 K - x^2 H(p^2) - \ldots - x^l H(p^l)}{H(p^1)}$$

This leads to the following bound on the optimal gain:

$$g^N(\sigma_K) \leq \frac{g(p^1)}{H(p^1)} \log_2 K + \sum_{i=2}^{l} x^i \left(g(p^i) - \frac{g(p^1)}{H(p^1)} H(p^i) \right) \tag{4.38}$$

Since $g(p^i) \leq \frac{g(p^1)}{H(p^1)} H(p^i)$ for all $i, 2 \leq i \leq l$, we have

$$g^N(\sigma_K) \leq \frac{g(p^1)}{H(p^1)} \log_2 K \tag{4.39}$$

Let $\sigma_K^{p^1}$ be the mixed strategy created by repeatedly expanding the most probable leaf according to probability vector p^1. By Lemma 4.21, assuming N is sufficiently large with respect to K,

$$g^N(\sigma_K^{p^1}) \geq \frac{g(p^1)}{H(p^1)} \log_2 (K - c) + \frac{g(p^1)}{H(p^1)} \log_2 p_s^1 \tag{4.40}$$

$$\geq g^N(\sigma_K) + \frac{g(p^1)}{H(p^1)} \log_2 p_s^1 - \frac{g(p^1)}{H(p^1)} \tag{4.41}$$

where p_s^1 is the smallest of the s elements in the support of p^1, $0 \leq c < s - 1$ is the smallest integer such that $K - c - 1$ is a multiple of $s - 1$ and we assume $c \leq \frac{K}{2}$.

Since $g^N(\sigma_K^{p^1})$ is within an additive constant of the optimal gain, the total expected payoff achieved by $\sigma_K^{p^1}$ is within an additive constant of the optimal total expected payoff. ∎

Let S_K be an optimal set for G_K^N and let $P(S_K)$ be the set of probability vectors assigned by the associated optimal mixed strategy to the forks in the simple tree representation of S_K. Since we have an $O(K)$ time algorithm to generate the tree with the maximum expected number of forks when we use the same probability vector at every fork, Theorem 4.22 implies that we can compute an approximately optimal set using $O(K)$ time plus the time needed to find the probability vector with the maximum gain per bit in $P(S_K)$. By using the results of Shapley and Snow (1950), we can compute a set P^* in time which is independent of K and N such that $P^* \supseteq P(S_K)$ for some optimal set S_K. By applying Algorithm 4.1 to the element of P^* with the maximum gain per bit, we have an $O(K)$ time algorithm which generates a set S'_K and a corresponding mixed strategy σ'_K such that the total expected payoff achieved by σ'_K is within an additive constant of the optimal.

Definition 4.23 *(Shapley and Snow, 1950) Let X and Y be the sets of optimal mixed strategies for Players 1 and 2 respectively in a game G. Let X^* and Y^* be the smallest sets whose convex hulls are X and Y respectively. A pair (x, y) is called a* **solution** *of G if $x \in X$ and $y \in Y$. A pair (x, y) is called a* **basic solution** *of G if $x \in X^*$ and $y \in Y^*$.*

Proposition 4.24 *(Shapley and Snow, 1950) A necessary and sufficient condition that a solution (x, y) of G be basic is that there exists a square sub-matrix M of G such that*

$$x^T = \frac{1^T(\text{adj } M)}{1^T(\text{adj } M)1} \tag{4.42}$$

$$y = \frac{(\text{adj } M)1}{1^T(\text{adj } M)1} \tag{4.43}$$

$$V(G) = \frac{\det M}{1^T(\text{adj } M)1} \tag{4.44}$$

where 1 is the appropriately sized column vector consisting entirely of 1's and the condition is considered not to hold when the right hand terms are indeterminate.

Definition 4.25 *if $x \in X^*$, where X^* is as defined in Definition 4.23, then x is called a* **basic mixed strategy** *for Player 1.*

Definition 4.26 *The set of* **potentially basic mixed strategies** *for Player 1 is the set, P^*, of mixed strategies, x, such that $x^T = \frac{1^T(\text{adj } M)}{1^T(\text{adj } M)1}$ for some square sub-matrix M of G.*

To show that there is an optimal set S_K which uses only mixed strategies from P^* on its forks, we need the following property of the adjoint of a matrix:

Lemma 4.27 *Let A be a square matrix and let A' be created from A by adding a different constant to each row i.e.*

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}$$

$$A' = \begin{pmatrix} a_{11}+c_1 & \dots & a_{1n}+c_1 \\ \vdots & \ddots & \vdots \\ a_{n1}+c_n & \dots & a_{nn}+c_n \end{pmatrix}$$

Then $1^T\text{adj } A = 1^T\text{adj } A'$.

Proof. It suffices to show that $1^T\text{adj } A = 1^T\text{adj } A'$ when only one of the c_i's say c_l is nonzero for some $l, 1 \leq l \leq n$. Let $A(j|i)$ be the matrix formed by removing row j and column i from A. We need to show

$$\sum_{i=1}^{m}(-1)^{i+j}\det A(j|i) \;=\; \sum_{i=1}^{m}(-1)^{i+j}\det A'(j|i) \tag{4.45}$$

for every $j, 1 \leq j \leq n$.

When $j = l$, Equation 4.45 holds since $A(l|i)$ is identical to $A'(l|i)$ for all $i, 1 \leq i \leq n$. By the linearity of the determinant (see Hoffman and Kunze, 1971, p. 142),

$\det A'(j|i) = \det A(j|i) + \det B(j|i)$ where

$$B = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{(l-1)1} & \dots & a_{(l-1)n} \\ c_l & \dots & c_l \\ a_{(l+1)1} & \dots & a_{(l+1)n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \tag{4.46}$$

We need to show $\sum_{i=1}^{n}(-1)^{i+j} \det B(j|i) = 0$ for $j \neq l$. Let $\beta_{jk} = (-1)^{j+k} \det B(j|k)$. We know $b_{i1}\beta_{j1} + \ldots + b_{in}\beta_{jn} = 0$ for each i and j such that $1 \leq j \neq i \leq n$ (see Agnew and Knapp, 1983, Theorem 8, p. 111). In particular, since $b_{lk} = c_l$ for each $k, 1 \leq k \leq n$, we have $\beta_{j1} + \ldots + \beta_{jn} = 0, 1 \leq j \neq l \leq n$ which implies Equation 4.45 for every $j, 1 \leq j \neq l \leq n$. Therefore, $1^T \text{adj } A = 1^T \text{adj } A'$. ∎

Theorem 4.28 *The set of potentially basic mixed strategies for Player 1 in G contains the set of mixed strategies used on the forks in the tree representation of some optimal set S_K for G_K^N.*

Proof. Consider any fork with l branches in the tree representation of an optimal set S_K. Let V_i be the expected payoff associated with the subtree under branch $i, 1 \leq i \leq l$. Player 1's decision at this fork is then equivalent to the decision he faces in a 1-stage game defined by

$$M = \begin{pmatrix} r_{1,1} + V_1 & \dots & r_{1,n} + V_1 \\ \vdots & \ddots & \vdots \\ r_{l_1} + V_l & \dots & r_{l,n} + V_l \end{pmatrix} \tag{4.47}$$

As in the proof of Proposition 4.4 we can assume that M does not have a saddle point. It is sufficient for Player 1 to choose a basic mixed strategy, x, for M. By Proposition 4.24, x must satisfy $x^T = \frac{1^T(\text{adj } M')}{1^T(\text{adj } M')1}$ for some square sub-matrix M' of M. By Lemma 4.27, $1^T(\text{adj } M') = 1^T(\text{adj } M'')$ where M'' is formed from M' by

subtracting the V_i's from their corresponding rows. M'' is therefore a square submatrix of the payoff matrix for the stage game G which implies that x is a potentially basic mixed strategy. ∎

Theorem 4.29 *If N is sufficiently large with respect to K, there is a set S'_K and an associated mixed strategy σ'_K which can be computed in $O(K)$ time such that the total expected payoff achieved by σ'_K in G_K^N is within an additive constant of the optimal.*

Proof. Let $p = \arg\max_{x \in P^*} \left\{ \frac{g(x)}{H(x)} \right\}$. Create a set S'_K and a corresponding mixed strategy σ'_K by repeatedly expanding the most probable leaf according to probability vector p using Algorithm 4.1. Since P^* can be computed directly from G, the time needed to compute p is independent of K and N. Therefore, the total time needed to compute S'_K and σ'_K is $O(K)$. That σ'_K achieves an total expected payoff within an additive constant of the optimal follows from Theorem 4.28 and the discussion following Theorem 4.22. ∎

Using the potentially basic mixed strategy p with the maximum gain per bit, Player 1 can achieve a total expected payoff within $\frac{g(p)}{H(p)}(\log_2 p_s - 1)$ of the optimal where p_s is the smallest element in the support of p. Since potentially basic mixed strategies are the only strategies that need to be considered, there is no other mixed strategy that achieves a total expected payoff within this constant of the optimal for every K. In other words, when K is sufficiently large, p is the best mixed strategy Player 1 can use if he creates his set using the same mixed strategy at every fork.

Experiments indicate that the total expected payoff is closer to the upper bound than the lower bound which suggests that the upper bound is probably a better estimate of the total expected payoff achieved by the approximately optimal strategy. For example consider the following game:

$$\begin{pmatrix} 3 & 1 & 5 \\ 0 & 0 & 9 \\ 5 & 1 & 3 \\ 2 & 4 & 1 \end{pmatrix}$$

There are eight potentially basic mixed strategies to consider. These are shown in Table 4.2 along with their gain, entropy, and gain per bit. Several other potentially basic mixed strategies were eliminated from consideration because they did not have a positive gain or because their entropy was the same as another strategy but their gain was lower. The strategies are listed in decreasing order of their gain per bit. In the table, $\mathrm{UB}_K = \frac{g(p)}{H(p)} \log_2 K$ is the upper bound on the gain given that p is used at every fork. $\mathrm{LB}_K = \frac{g(p)}{H(p)} \left(\log_2 (K - c) + \log_2 p_s\right)$ is the lower bound on the gain given that p is used at every fork where c is as defined in Lemma 4.21 and p_s is the smallest element in the support of p. The upper bound, lower bound, and actual gain $g^N(\sigma_K^p)$ achieved by using p at every fork are shown for $K = 100$ and $K = 1,000,000$. Note that since the strategies are ordered by their gain per bit they are also ordered by their upper bounds. Had we chosen a large enough K they would also have been ordered by LB_K. In fact, for large enough K, the lower bound for strategy i will be larger than the upper bound for strategy $i + 1$.

Table 4.2: A comparison of eight potentially basic mixed strategies.

	p	$g(p)$	$H(p)$	$\frac{g(p)}{H(p)}$	UB_{100}	$g^N(\sigma^p_{100})$	LB_{100}	UB_{10^6}	$g^N(\sigma^p_{10^6})$	LB_{10^6}
1	$\left(0,\frac{1}{10},0,\frac{9}{10}\right)$	0.80	0.469	1.706	11.33	10.48	5.67	34.00	33.28	28.33
2	$\left(\frac{1}{2},0,0,\frac{1}{2}\right)$	1.50	1.000	1.500	9.97	9.84	8.47	29.90	29.86	28.40
3	$\left(\frac{1}{3},0,0,\frac{2}{3}\right)$	1.33	0.981	1.452	9.65	9.54	7.35	28.94	28.83	26.63
4	$\left(\frac{3}{7},0,0,\frac{4}{7}\right)$	1.43	0.985	1.450	9.63	9.57	7.86	28.90	28.84	27.12
5	$\left(\frac{4}{11},0,\frac{1}{11},\frac{6}{11}\right)$	1.64	1.322	1.238	8.22	7.87	3.92	24.67	24.34	20.39
6	$\left(0,0,\frac{3}{5},\frac{2}{5}\right)$	1.20	0.971	1.236	8.21	8.15	6.57	24.63	24.58	23.00
7	$\left(0,\frac{4}{31},\frac{9}{31},\frac{18}{31}\right)$	1.61	1.355	1.191	7.91	7.65	4.37	23.73	23.49	20.21
8	$\left(0,\frac{1}{4},0,\frac{3}{4}\right)$	0.50	0.811	0.616	4.09	4.01	2.86	12.28	12.20	11.05

The optimal mixed strategy is $(4/11, 0, 1/11, 6/11)$ which is strategy number 5 in the table. Strategies 7 and 8 will not be used in an optimal tree since there are other potentially basic mixed strategies with the same size support which have higher gain and lower entropy. The upper bound is a fairly accurate measure of the true gain for all of the strategies and is closer than the lower bound to the actual gain in

each case. Strategy 1 has the largest gain per bit at 1.706. It is already significantly superior to the other strategies when $K = 100$ even though the lower bound for Strategy 1 is well below the lower bounds for other strategies in the list. The optimal gain for G^N_{100} was computed to be 10.98 which is only 0.5 above the gain achieved by our approximately optimal strategy. The value of $G^N_{10^6}$ was not computed since the dynamic programming algorithm to compute the value of the game exactly would require an enormous amount of time for such a large value for K. This demonstrates the benefit of having the $O(K)$ time approximation algorithm for S_K.

4.6 Conclusion

We have approached the problem of bounded rationality in repeated games by fixing the number of strategies available to a player rather than restricting him to a particular type of machine. Nevertheless, we have been able to discuss the computational requirements of optimal play both from the point of view of finding an optimal set of strategies and of executing the strategies from this set. In particular, we have shown that Player 1 needs $O(K)$ time to find an approximately optimal set and an $O(\log_2 K)$-bit memory to execute the strategies from this set using only a constant amount of time at each stage. Thus, even when we place time restrictions similar to that of a finite automaton on the player, the amount of memory he requires is nearly optimal. We have also given upper and lower bounds on the value of the repeated game when one player is restricted to a finite set of strategies. These bounds make clear the connection between strategic entropy and the value of repeated games with finite strategy sets.

This approach does not readily provide a way to analyze games in which both players are bounded. If we restrict both players to choosing sets of size K, the best set for Player 1 depends on the set that Player 2 chooses and vice versa. If we allow the choice of sets to be randomized, then any restriction on the players caused by limiting them to sets of size K vanishes. This is the case because a mixed strategy

in the unrestricted repeated game is a randomized choice over sets of size one. There does not appear to be any other obvious way to model this type of game.

Our analysis treats the characteristics of the stage game as constant. An alternative approach might be to treat the stage game as input and limit the time a player is allowed at each stage to be polynomial in the size of the stage game. The time taken at each stage would still be independent of the length of the repeated game. We could abandon the constant time requirement altogether and allow the players to use time at each stage which is a slow growing function of the length of the game. Future work should investigate whether any of these alternative models can provide insight into the study of bounded rationality.

Part II

Polynomial Time Mechanism Design

Chapter 5

Mechanism Design

5.1 Introduction

Consider a world inhabited by a number of self-interested agents that have different and possibly conflicting goals. By self-interested we mean that each agent is concerned only with satisfying its own goals and does not care whether any of the other agents satisfy their goals. Rather than spending time negotiating with one another when a conflict arises, the agents rely on an outside arbitrator to resolve the conflict quickly and equitably. The arbitrator's only goal is that its decisions satisfy some measure of social desirability called a *social choice rule*. For example, the decision might be required to be *pareto optimal* – no other possible decision makes one of the agents better off without making another agent worse off.

In the economics literature, the arbitrator in the example above is called a *mechanism*. When a mechanism guarantees that a social choice rule is satisfied, the mechanism is said to *implement* the social choice rule. The problem of defining a mechanism to implement a particular social choice rule is known as the *mechanism design* or *implementation problem*. The field of mechanism design has recently attracted the interest of researchers in multiagent systems, particularly those studying automated negotiation (Rosenschein and Zlotkin, 1994; Sandholm, 1999). Before mechanism

design can make practical contributions to multiagent system design, however, the computational issues involved must be clearly understood.

The literature on mechanism design is vast (see Moore, 1992 and Chapter 23 of Mas-Colell, Whinston, and Green, 1995 for surveys). However, Moore (1992) points out that much of this research ignores issues of practicality. He cites two areas for potential research along these lines:

1. modelling bounded rationality on the part of the agents
2. designing mechanisms that are robust to errors in the specification of the problem.

In this thesis, we address the idea of practicality in mechanism design from the point of view of theoretical computer science. We model bounded rationality by limiting the agents to polynomial time computation. Since we are interested in decisions that can be computed quickly, we also limit the mechanism to polynomial time computation. Our goal is to investigate the difficulties that arise when trying to implement NP-hard social choice rules and to determine what can be done to overcome these difficulties. Section 5.2 provides a short introduction to mechanism design. Section 5.3 formalizes what we mean by a polynomial time mechanism. Sections 6.1 and 6.2 look at designing polynomial time mechanisms using dominant strategy and Nash equilibrium respectively for a multiagent version of MAXSAT. In multiagent MAXSAT, each agent's preferences over the set of possible outcomes can be described by a disjunction of boolean variables. Since MAXSAT is NP-hard, the mechanism does not have time to find an outcome that satisfies the maximum number of simultaneously satisfiable agents so it must settle for an outcome that is approximately optimal. Section 6.3 provides a proof that the best a mechanism can guarantee is that half of the maximum number of simultaneously satisfiable agents will be satisfied. This is true for implementation in the four main equilibrium concepts used in the study of mechanism design in complete information environments.

We are not the first to study approximation algorithms in mechanism design.

Nisan and Ronen (1999) recently studied mechanism design for the task allocation problem under dominant strategy equilibrium. Ronen (1999) showed that, under certain conditions, dominant strategy implementation of social choice rules that approximately maximize the sum of the agents' utilities is impossible. Our work extends the idea of computational mechanism design to Nash implementation. In addition, we define precisely[1] what is necessary for a mechanism to be polynomial time computable.

It certainly could be argued that the multiagent MAXSAT problem as we define it is somewhat restrictive since each agent is limited to preferences that are simple disjunctions rather than more general boolean formulae. Our goal, however, is not to develop mechanisms for a particular application. Rather, our interest is in determining how restricting the agents and the mechanism to polynomial time computation affects the existing results in the economics literature on mechanism design. In particular, we would like to determine which mechanisms in the literature are polynomial time when the social choice rule consists of approximately optimal solutions to an NP-hard optimization problem. With these goals in mind and in order to focus our analysis, we need a conceptually simple NP-hard problem with many known approximation algorithms. Multiagent MAXSAT satisfies these requirements. On the other hand, we know of no approximation algorithms for the version of this problem in which the agents' preferences are defined by more general boolean formulae.

5.2 Mechanism Design

We begin with a formal description of the mechanism design problem. This section is based on material from Mas-Colell, Whinston, and Green (1995), Maskin (1985), and Moore (1992).

[1]Because Nisan and Ronen (1999) restricted their investigation to truthful implementation of single valued social choice rules, they did not need to be concerned with the amount of computation performed by the agents.

Definition 5.1 *A* **mechanism design problem** *consists of the following:*

- *a set of I* **agents** *that must make a collective choice over some finite set X of possible* **outcomes**.
- *for each agent i, a set Θ_i of possible* **types**. *An agent's type determines its preferences over the outcomes.*
- *for each agent i, a* **utility function** $u_i : X \times \Theta_i \rightarrow \mathcal{R}$ *that represents the agent's preferences given the agent's type. Each agent is assumed to be trying to maximize its utility.*
- *a* **social choice rule** $F : \Theta_1 \times \ldots \times \Theta_I \rightarrow 2^X \setminus \{\emptyset\}$. *If $F(\cdot)$ is single valued it is called a* **social choice function**.[2]

Definition 5.2 *A* **mechanism** $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *is a collection of action sets $A_1, \ldots, A_I$ and an outcome function $g : A_1 \times \ldots \times A_I \rightarrow X$. Each agent i chooses an action from action set A_i. The mechanism then sets the outcome to $g(a_1, \ldots, a_I)$.*

For example, consider the following multiagent version of the MAXSAT problem:

Definition 5.3 *The* **Multiagent** MAXSAT *problem is defined as follows:*

- *The parameters of the problem consist of a set of boolean variables.*
- *The set of outcomes consists of all truth assignments to the boolean variables.*
- *An agent's type consists of preferences over the truth assignments. The preferences are restricted to those that can be described by a simple disjunction over a subset of the variables and their negations. We will refer to such a disjunction as a clause. Each agent prefers that its clause be satisfied but does not differentiate between satisfying truth assignments. A type profile, therefore, is a vector of clauses where each clause represents a single agent's type.*

[2]We use lower case $f(\cdot)$ when discussing social choice functions and upper case $F(\cdot)$ when discussing social choice rules.

- *The goal of the mechanism is to maximize the total number of satisfied clauses. The social choice rule for this problem is defined by:*

$$\mathsf{MAXSAT}(\theta) = \{t : t \text{ is a truth assignment that satisfies } m(\theta) \text{ clauses}\} \quad (5.1)$$

where $m(\theta)$ is the maximum number of simultaneously satisfiable clauses in the type profile θ.

Consider a warehouse inhabited by several robots and assume that the states of the world can be represented by truth assignments over a set of boolean variables. For example, suppose there are two blocks B_1 and B_2 and one table. Let x_i = True represent B_i being on the table while x_i = False represents B_i being on the floor for $i = 1, 2$. A robot's type corresponds to its goal since the robot's goal determines its preferences over the outcomes. Suppose each robot's goal can be represented by disjunctions over the two variables and their negations. Let the mechanism's goal be to satisfy as many of the robots as possible. This is an example of a multiagent MAXSAT problem. Suppose the mechanism requires the robots to declare their types. In other words, the action sets are just the set of clauses over the boolean variables. Nothing prevents the robots from lying to the mechanism since the mechanism has no way to verify what the robots' goals really are. Notice, however, that the social choice rule which the mechanism is trying to satisfy depends on the robots' true type profile not the declared type profile. The mechanism wants to choose a truth assignment that maximizes the total number of true clauses that are satisfied regardless of the clauses the robots declared. To achieve its goal, the mechanism must be designed in such a way that the robots are enticed to reveal enough information about their true types for the mechanism to make an appropriate decision.

The situation that we study in this paper is one in which the mechanism does not have time to choose an optimal outcome so it must settle for an approximately optimal outcome. For example, suppose a mechanism for the robot example above uses Johnson's first approximation algorithm[3] to determine the outcome. This al-

[3]See Section 6.1.3 for a detailed description of this algorithm.

gorithm takes the literal[4] that appears in the most clauses and sets it to True. It then repeatedly chooses the unassigned literal that appears most in the remaining unsatisfied clauses and sets that to True. Let ties be broken by choosing the least numbered variable first and assigning True before False. If there are five robots with goals defined by the type profile $\theta = (x_1, \bar{x}_1 \vee \bar{x}_2, \bar{x}_1 \vee \bar{x}_2, x_2, x_2)$ then the mechanism chooses $t = \bar{x}_1 x_2$. Robot 1 is not satisfied by this outcome. However, if Robot 1 declared its type to be $x_1 \vee \bar{x}_2$, the outcome would be $t = x_1 \bar{x}_2$ which does satisfy Robot 1. Therefore, in this instance, it is better for Robot 1 to lie about its goals.

The literature considers two types of environments with regard to the information that the agents possess about each other. In the *complete information* case, which is the case we consider in this thesis, each agent knows his own type as well as the types of all the other agents. In the *incomplete information* case, each agent has no information or only partial information about the other agents' types. In either case, the mechanism does not know the agents' types. A mechanism $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ combined with a set of possible types Θ_i and a utility function u_i for each agent i induces a game of complete or incomplete information depending on the information available to the agents. At the beginning of the game, each agent somehow comes to know its type. Formally, we say each agent receives a signal that indicates its type. Each agent then chooses an action based on its type. In the complete information case, a strategy for agent i is a mapping $s_i : \Theta_1 \times \ldots \times \Theta_I \rightarrow A_i$ while in the incomplete information case, a strategy for agent i is a mapping $s_i : \Theta_i \rightarrow A_i$.

The mechanism designer's goal is to design the mechanism in such a way that, for every possible profile of preferences, some "rational" outcome of the induced game is socially desirable. In game theory, a "rational" outcome of a game is called an *equilibrium outcome*. There are many competing equilibrium concepts. In this thesis, we will be discussing dominant strategy and Nash equilibrium which are defined as

[4]A literal is a variable or its negation.

follows:[5]

Definition 5.4 *A strategy profile* $s^*(\cdot) = (s_1^*(\cdot), \ldots, s_I^*(\cdot))$ *is a* **dominant strategy equilibrium** *of a mechanism* $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *if, for all* i *and all type profiles* θ*,*

$$u_i(g(s_i^*(\theta), a_{-i}), \theta_i) \geq u_i(g(a_i', a_{-i}), \theta_i)$$

for all $a_i' \in A_i$ *and all* $a_{-i} \in A_{-i}$*.*

In other words, a strategy profile is a dominant strategy equilibrium if no matter what the other agents do, no agent has another action that can increase its utility.

Definition 5.5 *A* **Nash equilibrium** *of a mechanism* $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *is a strategy profile* $s^*(\cdot)$ *such that, for each agent* i *and all type profiles* $\theta = (\theta_1, \ldots, \theta_I)$*,*

$$u_i(g(s_i^*(\theta), s_{-i}^*(\theta)), \theta_i) \geq u_i(g(a_i', s_{-i}^*(\theta)), \theta_i)$$

for all $a_i' \in A_i$*.*

In other words, a strategy profile is a Nash equilibrium if no agent can unilaterally deviate from the prescribed action and increase its utility.

Definition 5.6 *An* **equilibrium outcome** *of a mechanism* $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *for some type profile* θ *is the outcome produced by an equilibrium strategy profile* $s^*(\theta)$ *of* Γ*.*

There are different degrees to which a mechanism can satisfy a social choice rule. These are defined as follows:

Definition 5.7 *Given a mechanism* Γ*, let* $E(\theta)$ *be the set of equilibrium outcomes for a type profile* θ*.*

[5]We use the notation a_{-i} to indicate the vector consisting of all elements of vector $(a_1, \ldots, a_I)$ except element i.

1. Γ **implements** $F(\cdot)$ *if* $E(\theta) \cap F(\theta) \neq \emptyset$ *for all* θ.

2. Γ **strongly implements** *if* $\emptyset \neq E(\theta) \subset F(\theta)$ *for all* θ.

3. Γ **fully implements** *if* $E(\theta) = F(\theta)$ *for all* θ.

If multiple equilibria exist, it might be difficult to argue that one equilibrium will be played while another will not since all equilibria are equally rational according to any particular equilibrium criterion. The definition of implementation does not exclude the possibility that undesirable equilibrium outcomes will occur. Strong implementation eliminates mechanisms that do not guarantee that every equilibrium outcome is socially desirable. Full implementation receives a great deal of attention in the literature on Nash implementation. However, in line with the philosophy of approximation algorithms, we want to guarantee that all outcomes reach some threshold of acceptability. We are not necessarily concerned with making all acceptable outcomes possible. In designing mechanisms to control the collective decision making process of a group of autonomous agents, strong implementation should suffice.[6]

5.3 Polynomial Time Mechanisms

From a computational perspective, the mechanism design problem consists of two parts: the computation performed by the mechanism and the computation performed by the agents. From the mechanism's point of view, an instance of the problem consists of the description of the problem's parameters plus a description of the actions chosen by the agents. From the agent's point of view, an instance of the problem consists of the description of the problem's parameters and a description of the agent's type.[7] Hence, we have the following definitions:

[6]However, we return to this discussion in Section 6.2 where we provide some justification for trying to achieve full implementation.

[7]It should also include a description of the mechanism but we will assume that either the description of the mechanism is fixed or that its size is polynomial in the size of the problem's parameters.

Definition 5.8 $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *is a* **polynomial time mechanism** *if there is some polynomial* $p(\cdot)$ *such that* $g(a_1, \ldots, a_I)$ *is computable in time which is* $O(p(|a_1| + \ldots + |a_I| + l))$ *where* $|a_i|$ *is the size of the binary description of action* a_i *and* l *is the size of the binary description of the problem's parameters.*

Definition 5.9 *A* **polynomial time strategy profile** *is a strategy profile* $s(\cdot) = (s_1(\cdot), \ldots, s_I(\cdot))$ *such that for some polynomial* $p(\cdot)$ *and for all* i, $s_i(\theta)$ *is computable in time which is* $O(p(|\theta| + l))$ *where* $|\theta|$ *is the size of the binary description of the agents' type profile and* l *is the size of the binary description of the problem's parameters.*

Definition 5.10 *A* **polynomial time equilibrium outcome** *for a mechanism* $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ *is an outcome* $x \in X$ *such that* $x = g(s^*(\theta))$ *for some type profile* θ *and some polynomial time equilibrium strategy profile* $s^*(\cdot)$.

We are interested in situations in which the computation performed both by the mechanism and the agents is restricted to polynomial time. This leads to the following variations on Definition 5.7:

Definition 5.11 *Given a mechanism* Γ, *let* $PE(\theta)$ *be the set of polynomial time equilibrium outcomes for a type profile* θ. *If* Γ *is a polynomial time mechanism then we say that,* **in polynomial time for polynomial time bounded agents**,

1. Γ **implements** $F(\cdot)$ *if* $PE(\theta) \cap F(\theta) \neq \emptyset$ *for all* θ.

2. Γ **strongly implements** $F(\cdot)$ *if* $\emptyset \neq PE(\theta) \subset F(\theta)$ *for all* θ.

3. Γ **fully implements** $F(\cdot)$ *if* $PE(\theta) = F(\theta)$ *for all* θ.

The restriction to polynomial time equilibrium outcomes requires some discussion. As the following Lemma shows, if there is at least one polynomial time equilibrium strategy profile then every equilibrium outcome is a polynomial time equilibrium outcome:

Lemma 5.12 *Given a mechanism Γ, let $PE(\theta)$ be the set of polynomial time equilibrium outcomes for type profile θ. Let $E(\theta)$ be the set of all equilibrium outcomes for type profile θ. If $PE(\theta) \neq \emptyset$ for all θ then $PE(\theta) = E(\theta)$ for all θ.*

Proof. If suffices to show that $E(\theta) \subseteq PE(\theta)$. Since $PE(\theta) \neq \emptyset$ for all θ there exists a polynomial time equilibrium strategy profile, $s^*(\cdot)$, for Γ. Let $\hat{\theta}$ be any type profile and let t be any member of $E(\hat{\theta})$. Then $t = g(\hat{s}(\hat{\theta}))$ for some equilibrium strategy profile $\hat{s}(\cdot)$. Define a new strategy profile $s'(\cdot)$ such that for all i,

$$s'_i(\theta) = \begin{cases} s^*_i(\theta) & \text{if } \theta \neq \hat{\theta} \\ \hat{s}(\theta) & \text{otherwise} \end{cases}$$

For each i, $s'_i(\cdot)$ is polynomial time computable since $s^*_i(\cdot)$ is (assuming it is easy to compare types). Furthermore, $s'(\cdot)$ is an equilibrium strategy profile since $s^*(\cdot)$ and $\hat{s}(\cdot)$ are. This implies, $t = g(\hat{s}(\hat{\theta})) = g(s'(\hat{\theta})) \in PE(\hat{\theta})$. Therefore, $E(\theta) \subseteq PE(\theta)$. ∎

As a result, we have the following proposition:

Proposition 5.13 *If Γ strongly [fully] implements $F(\cdot)$ in polynomial time for polynomial time bounded agents then Γ strongly [fully] implements $F(\cdot)$.*

Proof. Since Γ strongly implements $F(\cdot)$ in polynomial time for polynomial time bounded agents, Γ has some polynomial time equilibrium strategy profile. The result then follows from Lemma 5.12 and Definitions 5.7 and 5.11. ∎

Proposition 5.13 implies that any property of social choice rules that is required for strong or full implementation is also required for strong or full implementation respectively in polynomial time for polynomially time bounded agents.

5.3.1 Revelation Mechanisms

Mechanisms in which $A_i = \Theta_i$ for all i form an important class of mechanisms known as *revelation mechanisms*. In other words, revelation mechanisms require the agents to declare their types. We say that a social choice rule $F(\cdot)$ is *truthfully implementable*

if there is a revelation mechanism for which truthful type declarations by all the agents constitutes an equilibrium with an outcome in $F(\cdot)$. More formally:

Definition 5.14 *A social choice rule $F(\cdot)$ is* **truthfully implementable** *if there is a revelation mechanism $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ that has an equilibrium strategy profile $s^*(\cdot)$ such that $s_i^*(\theta) = \theta_i$ and $g(s^*(\theta)) \in F(\theta)$ for all type profiles θ and all agents i.*

Definition 5.15 *A revelation mechanism Γ is said to be* **truthful** *if truth telling by all the agents constitutes a dominant strategy equilibrium for Γ.*

The following result known as the *Revelation Principle* states that truthfully implementable social choice rules are the only social choice rules that can be implemented. This principle applies to many equilibrium concepts including dominant strategy and Nash equilibrium so we state it without specifying a particular equilibrium concept (see the discussion in Maskin, 1985 on p. 182-183):

Proposition 5.16 *(The Revelation Principle) Suppose there exists a mechanism Γ that implements social choice rule $F(\cdot)$. Then $F(\cdot)$ is truthfully implementable.*

Proof. See Mas-Colell, Whinston, and Green (1995). ∎

The Revelation Principle has a polynomial time analogue but only if the agents are restricted to polynomial time strategies. This result will hold for any equilibrium concept for which the standard Revelation Principle applies.

Proposition 5.17 *(The Polynomial Time Revelation Principle) If a social choice rule $F(\cdot)$ is polynomial time implementable for polynomial time bounded agents then $F(\cdot)$ is truthfully polynomial time implementable for polynomial time bounded agents.*

Proof. We provide the proof for the case of dominant strategy equilibrium. The proofs for other equilibrium concepts are similar. Let $\Gamma = (A_1, \ldots, A_I, g(\cdot))$ be a polynomial time mechanism that implements $F(\cdot)$. Let $s^*(\cdot)$ be a polynomial time equilibrium strategy profile for Γ such that $g(s^*(\theta)) \in F(\theta)$ for all θ. Let $g^*(\theta) =$

$g(s^*(\theta))$. Then $\Gamma' = (\Theta_1, \ldots, \Theta_I, g^*(\cdot))$ is a polynomial time mechanism. To see that truth telling is a dominant strategy for Γ', let θ_i be agent i's true type and suppose that for some θ_{-i} agent i is strictly better off in Γ' by declaring his type to be $\theta_i' \neq \theta_i$. Then $u_i(g^*(\theta), \theta_i) < u_i(g^*(\theta_i', \theta_{-i}), \theta_i)$. Define an alternative strategy for agent i playing Γ by:

$$s_i'(\hat{\theta}_i) = \begin{cases} s_i^*(\hat{\theta}_i) & \text{if } \hat{\theta}_i \neq \theta_i \\ s^*(\theta_i') & \text{otherwise} \end{cases}$$

Clearly, $s_i'(\cdot)$ is polynomial time computable since $s_i^*(\cdot)$ is (again assuming it is easy to compare types). Furthermore, $u_i(g(s_i^*(\theta_i), s_{-i}^*(\theta_{-i})), \theta_i) < u_i(g(s_i'(\theta_i), s_{-i}^*(\theta_{-i})), \theta_i)$ which contradicts $s_i^*(\cdot)$ being a dominant strategy for agent i playing Γ. Therefore, Γ' truthfully implements $F(\cdot)$ in dominant strategies. ∎

The above argument crucially depends on the agents being restricted to polynomial time since, if the agents could use exponential time, they could simply pad their input to the mechanism so that the size of the input was exponential in $|\theta|$. Computing $g(\cdot)$ in time polynomial in its input size in this case would be equivalent to computing $g^*(\cdot)$ in time which is exponential in its input size.

Using the Polynomial Time Revelation Principle, we can show that $\mathsf{MAXSAT}(\cdot)$ defined in Equation 5.1 is not polynomial time implementable.

Proposition 5.18 *If* $\mathsf{MAXSAT}(\cdot)$ *is polynomial time implementable for polynomial time bounded agents then* $\mathsf{P} = \mathsf{NP}$.

Proof. If $\mathsf{P} \neq \mathsf{NP}$ there is no polynomial time algorithm for finding a member of $\mathsf{MAXSAT}(\cdot)$ (see Garey and Johnson, 1979). Suppose $\mathsf{MAXSAT}(\cdot)$ is polynomial time implementable for polynomial time bounded agents. By the Polynomial Time Revelation Principle, there exists a polynomial time mechanism $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ such that $g(\theta) \in \mathsf{MAXSAT}(\theta)$ for all $\theta \in \Theta$. But then the polynomial time algorithm that computes $g(\cdot)$ finds a member of $\mathsf{MAXSAT}(\cdot)$ in polynomial time which implies $P = NP$. ∎

Since MAXSAT$(\cdot)$ is not polynomial time implementable for polynomial time bounded agents, the best one can hope for is to implement some approximation to MAXSAT$(\cdot)$ in polynomial time. An approximation to MAXSAT$(\cdot)$ is defined as a social choice rule of the form:

$$c\text{-MAXSAT}(\theta) = \{t : t \text{ satisfies at least } cm(\theta) \text{ clauses}\}$$

where $c \in (0,1)$ and $m(\theta)$ is the maximum number of simultaneously satisfiable clauses in θ. In general, we define an approximation to a social choice rule as follows:

Definition 5.19 *Let $F(\cdot)$ be a social choice rule that maximizes some real valued objective function $h : X \times \Theta_1 \times \ldots \times \Theta_I \to \mathcal{R}$. In other words, for all θ,*

$$F(\theta) = \{t \in X : t \in \arg\max_{x \in X} h(x, \theta)\}.$$

A **c-approximation** *for $F(\cdot)$ is a social choice rule c-$F(\cdot)$ such that*

$$c\text{-}F(\theta) = \{t \in X : h(t, \theta) \geq c \max_{x \in X} h(x, \theta)\}$$

where c is a constant between 0 and 1.

Definition 5.20 *A social choice rule $F(\cdot)$ is* **approximately implementable** *if there is some constant $c \in (0,1)$ such that c-$F(\cdot)$ is implementable.*

In Sections 6.1 and 6.2, we examine whether there exists a constant c such that c-MAXSAT$(\cdot)$ is implementable in dominant strategy or Nash equilibrium and which approximation algorithms for MAXSAT can be used to create polynomial time mechanisms that implement c-MAXSAT$(\cdot)$.

Chapter 6

The Multiagent MAXSAT Implementation Problem

6.1 Dominant Strategy Implementation

In this section, we investigate whether there is a constant c such that c-MAXSAT$(\cdot)$ is truthfully implementable in dominant strategies. We examine three different approximation algorithms for MAXSAT – two from Johnson (1974) and one which we call the complement algorithm. We begin with a discussion of the existing impossibility theorems for dominant strategy implementation.

6.1.1 Existing Impossibility Theorems

Dominant strategy equilibrium is a very strong equilibrium concept and there has been a great deal of work in determining whether "satisfactory" social choice functions can be implemented in dominant strategies. An example of a social choice function that would be considered unsatisfactory is a function for which one agent alone determines the outcomes that are considered desirable. Such a function is called dictatorial.

Definition 6.1 *A social choice function $f(\cdot)$ is* **dictatorial** *if there is a single agent i such that, for all type profiles θ, $f(\theta)$ is one of agent i's most preferred outcome.*

Unfortunately, in many cases it is impossible to implement a non-dictatorial social choice function in dominant strategies. The following result is known as the Gibbard-Satterthwaite Theorem:

Proposition 6.2 *(Gibbard, 1973; Satterthwaite, 1975) Let $f(\cdot)$ be a social choice function. Suppose the set of possible outcomes, X, is finite and contains at least three elements and that the range of $f(\cdot)$ is X. Further suppose that the set of possible preference relations over X contains the set of strict preferences over X. Then $f(\cdot)$ is truthfully implementable in dominant strategies if and only if $f(\cdot)$ is dictatorial.*

Proof. See Mas-Colell, Whinston, and Green (1995) p. 874-875. ∎

This impossibility result for such a large class of problems has lead people to study dominant strategy implementation in restricted environments. The most commonly studied is the quasilinear environment where it is assumed that there is a way for the mechanism to transfer money among the agents in order to induce the agents to provide accurate information.

Definition 6.3 *In a* **quasilinear environment** *we assume the following:*

- *There is a finite set K called the* **project choice set**.
- *The set of outcomes X is $\{(k, t_1, \ldots, t_I) : k \in K, t_i \in \mathcal{R}, \text{ and } \sum_i t_i \leq 0\}$ where t_i is a* **transfer** *of money to agent i.*
- *Each agent has a* **quasilinear utility function** *i.e. for each i*

$$u_i((k, t_1, \ldots, t_I), \theta_i) = v_i(k, \theta_i) + t_i$$

where $v_i(k, \theta_i)$ is a real valued function called agent i's **valuation** *of project choice k.*

- *The outcome function takes the form*

$$g(\theta) = (k(\theta), t_1(\theta), \ldots, t_I(\theta))$$

where $\theta \in \Theta_1 \times \ldots \times \Theta_I$, $k : \Theta_1 \times \ldots \times \Theta_I \rightarrow K$ *and* $t_i : \Theta_1 \times \ldots \times \Theta_I \rightarrow \mathcal{R}$ *for* $1 \leq i \leq I$ *and* $\sum_{i=1}^{I} t_i \leq 0$.

Definition 6.4 *Let* $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ *be a revelation mechanism in a quasilinear environment where* $g(\theta) = (k(\theta), t_1(\theta), \ldots, t_I(\theta))$. Γ *is called a* **Groves mechanism** *if both of the following hold:*

1. $k(\theta)$ *maximizes the sum of the agents' valuations i.e.*

$$\sum_i v_i(k(\theta), \theta_i) \geq \sum_i v_i(k', \theta_i)$$

for all $\theta \in \Theta_1 \times \ldots \times \Theta_I$ *and for all* $k' \in K$.

2. $t_i(\theta) = h_i(\theta_{-i}) + \sum_{j \neq i} v_j(k(\theta), \theta_j)$ *where* $h(\cdot)$ *is an arbitrary function of* θ_{-i}.

Proposition 6.5 *(Groves, 1973; Green and Laffont, 1979) A Groves mechanism is truthful. Furthermore, in a quasilinear environment, if, for each agent* i*, the set of possible valuation functions consists of all functions from* $K \times \Theta_I$ *to* $\mathcal{R}$*, then every truthful mechanism is a Groves mechanism.*

There is another impossibility theorem to contend with when dealing with approximate dominant strategy implementation in quasilinear environments. Ronen (1999) proves that, in quasilinear environments with certain conditions on the agents' valuations, it is impossible to implement social choice functions that approximately maximize the sum of the agents' valuations.

Definition 6.6 *Let* $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ *be a revelation mechanism in a quasilinear environment where* $g(\theta) = (k(\theta), t_1(\theta), \ldots, t_I(\theta))$. Γ *is called an* **approximate Groves mechanism** *if both of the following hold:*

1. *$k(\theta)$ approximately maximizes the sum of the agents' valuations i.e. for some constant $c \in (0,1)$*

$$\sum_i v_i(k(\theta), \theta_i) \geq c \sum_i v_i(k', \theta_i)$$

for all $\theta \in \Theta_1 \times \ldots \times \Theta_I$ and for all $k' \in K$.

2. *$t_i(\theta) = h_i(\theta_{-i}) + \sum_{j \neq i} v_j(k(\theta), \theta_j)$ where $h(\cdot)$ is an arbitrary function of θ_{-i}.*

Proposition 6.7 *(Ronen, 1999) If, for each agent i, the set of possible valuation functions consists of all functions from $K \times \Theta_I$ to $\mathcal{R}$ then an approximate Groves mechanism which makes a non-optimal choice for at least one type profile is not truthful.*[1]

Proof. We provide the proof using slightly different notation than Ronen (1999) in order to highlight the choice of valuation. Let $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ be an approximate Groves mechanism where $g(\theta) = (k^*(\theta), t_1(\theta), \ldots, t_I(\theta))$. Let θ be a type profile that results in a non-optimal choice, i.e. $\sum_i v_i(k^*(\theta_i, \theta_{-i}), \theta_i) < \sum_i v_i(k_{opt}, \theta_i)$ for some $k_{opt} \in K$. Since the set of valuations is sufficiently rich, there must be some type $\hat{\theta}_i$ such that

$$v_i(k, \hat{\theta}_i) = \begin{cases} -\sum_{j \neq i} v_j(k, \theta_j) + 10^6 & \text{if } k = k_{opt} \\ -\sum_{j \neq i} v_j(k, \theta_j) & \text{otherwise} \end{cases} \tag{6.1}$$

Since $\sum_i v_i(k^*(\hat{\theta}_i, \theta_{-i}), \theta_i) \geq c \left(v_i(k', \hat{\theta}_i) + \sum_{j \neq i} v_j(k', \theta_j) \right)$ for all $k' \in K$, it must be the case that $k^*(\hat{\theta}_i, \theta_{-i}) = k_{opt}$. Hence,

$$\begin{aligned} u_i(k^*(\hat{\theta}_i, \theta_{-i}), \theta_i) &= v_i(k_{opt}, \theta_i) + \sum_{j \neq i} v_j(k_{opt}, \theta_j) + h(\theta_{-i}) \\ &> v_i(k^*(\theta_i, \theta_{-i}), \theta_i) + \sum_{j \neq i} v_j(k^*(\theta_i, \theta_{-i}), \theta_j) + h(\theta_{-i}) \\ &= u_i(k^*(\theta_i, \theta_{-i}), \theta_i) \end{aligned}$$

[1]The original statement of this theorem in Ronen (1999) restricts the set of valuations to all functions from $K \times \Theta_I$ to $(-\infty, 0]$.

Agent i, therefore, has incentive to lie when his type is θ_i and the other players declare their types to be θ_{-i}. Thus, truth-telling is not a dominant strategy for player i. ∎

Notice that the agent's lie in Equation 6.1 forces the mechanism to choose an optimal outcome. If it is NP-hard to find a k that maximizes the sum of the agents' valuations and if the agents are restricted to polynomial time strategies then it is impossible for an agent to declare such a false type every time the mechanism would otherwise choose a non-optimal outcome. The proof of Proposition 6.7, however, only requires the agent to declare this false type in a single instance where the mechanism chooses a non-optimal outcome. Proposition 6.7, therefore, holds even when the agents are restricted to polynomial time strategies.

To complete the proof that it is impossible to implement social choice functions that approximately maximize the sum of the agents' valuations we need the following proposition:

Proposition 6.8 *In a quasilinear environment, suppose that, for each agent i, the set of possible valuation functions consists of all functions from $K \times \Theta_I$ to $\mathcal{R}$. If a mechanism which approximately maximizes the sum of the agent's valuations has a dominant strategy equilibrium in which every agent tells the truth then it must be an approximate Groves mechanism.*

Proof. The proof is identical to the proof given in Mas-Colell, Whinston, and Green (1995) on p. 879 for the non-approximate version of the same result so we omit it. ∎

6.1.2 Implementing MAXSAT(·)

The Gibbard-Satterthwaite Theorem does not apply to multiagent MAXSAT since the set of preference relations for MAXSAT does not include the set of strict preference relations.[2] To see this, observe that if an agent's clause does not include a variable

[2]The Gibbard-Satterthwaite Theorem has been extended in several ways (see Note 62 in Moore, 1992). However, none of these extensions can apply to the multiagent MAXSAT problem since we

x_i then the agent is indifferent between truth assignments that are identical except in their assignment to x_i. Furthermore, if the agent's clause includes more than one literal then the agent is indifferent between truth assignments that satisfy at least one of the literals. As the following proposition shows, MAXSAT$(\cdot)$ is truthfully implementable in dominant strategies.

Proposition 6.9 MAXSAT$(\cdot)$ *is truthfully implementable in dominant strategies. Furthermore, there is a revelation mechanism that truthfully implements* MAXSAT$(\cdot)$ *and that uses a non-dictatorial outcome function.*

Proof. Order the truth assignments lexicographically where $x_i =$ True comes before $x_i =$ False for $1 \leq i \leq n$. Define a revelation mechanism $\Gamma = (\Theta_1, \ldots, \Theta_I, g(\cdot))$ such that $g(\theta)$ is the first truth assignment in the lexicographic ordering that satisfies the maximum number of simultaneously satisfiable clauses in the declared type profile θ. To prove that truth telling is a dominant strategy for Γ it suffices to show that no agent can improve the outcome for itself by removing a literal or adding a literal. This is sufficient since all lies are combinations of these two operations.

For any truth assignment $\hat{t}$ and any type profile $\hat{\theta}$, define $S(\hat{\theta}, \hat{t})$ to be the number of clauses in $\hat{\theta}$ that $\hat{t}$ satisfies. Let θ be the type profile declared by the agents. Let $t = g(\theta)$. Suppose agent i is not satisfied by t. Let t' be any truth assignment that does satisfy θ_i. It must be the case that $S(\theta, t') \leq S(\theta, t)$.

Suppose agent i lies about its type by declaring θ_i' which is θ_i with one literal removed. Since changing θ_i cannot change the number of clauses in θ_{-i} that are satisfied by any truth assignment, $S(\theta', t') \leq S(\theta, t') \leq S(\theta, t) = S(\theta', t)$

Now suppose θ_i' is created from θ_i by adding one literal. This doesn't change the number of clauses satisfied by t' since t' satisfies θ_i' and does not decrease the number of clauses satisfied by t. Therefore, $S(\theta', t') = S(\theta, t') \leq S(\theta, t) \leq S(\theta', t)$.

---

show below that MAXSAT$(\cdot)$ is truthfully implementable in dominant strategies and the outcome function for the mechanism used in the proof is not dictatorial.

In either case, t still satisfies at least as many clauses as t' and the tie breaking rules still favor t so $t' \neq g(\theta')$. Since this is true for any t' that satisfies θ_i, agent i cannot improve the outcome by lying. Hence, truth telling is a dominant strategy.

To see that $g(\cdot)$ is not dictatorial, suppose $\theta = (\bar{x}_1, x_1, \ldots, x_1)$. The mechanism sets x_1 to True so the outcome is not agent 1's most preferred outcome. The same argument can be applied to the other agents which implies that $g(\theta)$ is not the same agent's most preferred outcome for every θ. ∎

We know from Proposition 5.18 that MAXSAT$(\cdot)$ cannot be implemented in polynomial time so we are interested in determining whether there is some constant c such that c-MAXSAT$(\cdot)$ can be implemented in polynomial time. In general, if $F(\cdot)$ is a social choice rule that maximizes the sum of the agents' valuations and if it is NP-hard to find a member of $F(\cdot)$, then the Polynomial Time Revelation Principle (Propositions 5.17) and Ronen's Theorem (Propositions 6.7and 6.8) together imply that, in a quasilinear environment with a sufficiently rich set of valuations, c-$F(\cdot)$ is not polynomial time implementable in dominant strategies for polynomial time bounded agents.[3] Fortunately, although MAXSAT$(\cdot)$ does maximize the sum of the agents' utilities, the conditions of Ronen's Theorem are not met since the range of the valuations is restricted to $\{0, 1\}$. There is hope, then, that for some c, c-MAXSAT$(\cdot)$ can be implemented in polynomial time. The question is how to convert one of the many existing approximation algorithms for MAXSAT into a mechanism that truthfully implements c-MAXSAT$(\cdot)$. In the remainder of this section, we consider three different algorithms – two from Johnson (1974) and one which we call the complement algorithm.

[3] Assuming P $\neq$ NP.

6.1.3 Implementing c-MAXSAT($\cdot$) in polynomial time

Johnson's algorithms

Johnson (1974) provides two approximation algorithms for MAXSAT. The first is a 1/2-approximation and the second is a 2/3-approximation (see Battiti, 1998). Johnson's first algorithm, given in Algorithm 6.1 repeatedly finds the literal the appears most in the clauses that are currently unsatisfied and assigns True to that literal.

Algorithm 6.1: Johnson1($\mathbf{V}, \mathbf{C}$).

/* V is the set of variables */

/* C is the set of clauses */

1. Initialize truth assignment t to be the null truth assignment.
2. Let $A = \emptyset$ /* A is the set of literals corresponding to assigned variables. */
3. Let $S = \emptyset$ /* S is the set of clauses that are currently satisfied by t.*/
4. Repeat
5. Of the literals not in A, let u be the literal appearing in the most clauses in $C \setminus S$ with ties broken by taking the lowest numbered variable first and assigning True before False.
6. $t(u) \leftarrow$ True.
7. $S \leftarrow S \cup \{c \in C : c \text{ contains } u\}$.
8. $A \leftarrow A \cup \{u, \bar{u}\}$.
9. Until A contains the entire set of literals.
10. Return t.

Johnson's second algorithm, given in Algorithm 6.2, assigns each clause a weight based on the size of the clause. For each variable v, it sets v to True if the sum of

the weights of the clauses that are currently unsatisfied and that contain v is at least as large as the sum of the weights of the clauses containing $\bar{v}$ that are unsatisfied. It sets v to False otherwise. It then updates the weights of the clauses to reflect the fact that v is no longer available to assign.

Algorithm 6.2: Johnson2(V, C).

/* V is the set of variables */

/* C is the set of clauses */

1. Initialize truth assignment t to be the null truth assignment.
2. Let $A = \emptyset$ /* A is the set of literals corresponding to assigned variables. */
3. Let $S = \emptyset$ /* S is the set of clauses that are currently satisfied by t.*/
4. For each clause $c \in C$, let $w(c) = 2^{-|c|}$.
5. Repeat
6. Let u be the least numbered variable that appears in a clause from $C \setminus S$.
7. Let $C_T = \{c \in C \setminus S : c \text{ contains } u\}$.
8. Let $C_F = \{c \in C \setminus S : c \text{ contains } \bar{u}\}$.
9. If $\sum_{c \in C_T} w(c) \geq \sum_{c \in C_F} w(c)$
10. then $t(u) \leftarrow$ True.
11. $S \leftarrow S \cup C_T$
12. $A \leftarrow A \cup \{u, \bar{u}\}$
13. for all $c \in C_F$,
14. $w(c) \leftarrow 2w(c)$.
15. else $t(u) \leftarrow$ False.
16. $S \leftarrow S \cup C_F$
17. $A \leftarrow A \cup \{u, \bar{u}\}$

18. for all $c \in C_T$,

19. $w(c) \leftarrow 2w(c)$.

20. Until no clause in $S \setminus C$ contains a variable from $V \setminus A$.

21. For all variables $v \in V \setminus A$

22. $t(v) \leftarrow$ True.

23. Return t.

Using Johnson's algorithms to choose the outcomes does not generate mechanisms in which truth telling is a dominant strategy.

Proposition 6.10 *Let $\Gamma_1 = (\Theta_1, \ldots, \Theta_I, g_1(\cdot))$ where $g_1(\cdot)$ is the outcome function defined by Algorithm 6.1. Let $\Gamma_2 = (\Theta_1, \ldots, \Theta_I, g_1(\cdot))$ where $g_2(\cdot)$ is the outcome function defined by Algorithm 6.2. If $I \geq 5$, Γ_1 is not truthful. If $I \geq 2$, Γ_2 is not truthful.*

Proof. Let $I = 5$ and let $\theta = (x_1, \bar{x}_1 \vee \bar{x}_2, \bar{x}_1 \vee \bar{x}_2, x_2, x_2)$. Since ties are broken by choosing the least numbered variable first and assigning True before False, $g_1(\theta) = \bar{x}_1 x_2$. Agent 1 is not satisfied by this outcome. However, if agent 1 declared its type to be $x_1 \vee \bar{x}_2$, the outcome would be $x_1 \bar{x}_2$ which does satisfy agent 1. Therefore, truth telling is not a dominant strategy for agent 1 in Γ_1.

Let $I = 2$ and let $\theta = (x_1, \bar{x}_1 \vee x_2)$. Since ties are broken by choosing the least numbered variable first and assigning True before False, $g_2(\theta) = \bar{x}_1 x_2$ which does not satisfy agent 1. If agent 1 declared its type to be $x_1 \vee \bar{x}_2$, the outcome would be $x_1 x_2$ which satisfies agent 1. Therefore, truth telling is not a dominant strategy for agent 1 in Γ_2. ∎

The Complement Algorithm

The following property of MAXSAT provides a very simple 1/2-approximation algorithm for MAXSAT.

Lemma 6.11 *For a truth assignment t, let $\bar{t}$ denote the truth assignment such that for every variable v, $\bar{t}(v) =$ True if and only if $t(v) =$ False. For all truth assignments t, either t or $\bar{t}$ satisfies at least half the maximum number of satisfiable clauses.*

Proof. Let θ_i be any clause that t does not satisfy. Let l_i be a literal in θ_i. Then $t(l_i) =$ False which implies $\bar{t}(l_i) =$ True. Therefore, $\bar{t}$ satisfies θ_i. ∎

We can use this property to develop a polynomial time mechanism that strongly implements $\frac{1}{2}$-MAXSAT$(\cdot)$. Define a social choice function $f(\cdot)$ as follows:

$$f(\theta) = \begin{cases} \bar{t} & \text{if } \bar{t} \text{ satisfies more clauses in } \theta \text{ than } t \\ t & \text{otherwise} \end{cases} \tag{6.2}$$

This social choice function is not dictatorial since for each agent i we can define θ such that $\bar{t}$ does not satisfy θ_i but does satisfy every clause in θ_{-i} while t does not satisfy any clause in θ_{-i}. For example, suppose $t = x_1\bar{x}_2\bar{x}_3$. Let $\theta = (x_1, x_2, x_3)$. We have $f(\theta) = \bar{t}$ which is not agent 1's most preferred outcome. A similar argument applies to the other two agents which implies that $f(\theta)$ is not the same agent's most preferred outcome for every θ. Lemma 6.11 implies that $f(\theta)$ is a $\frac{1}{2}$-approximation for MAXSAT$(\cdot)$. Therefore, any mechanism that truthfully implements $f(\cdot)$, truthfully implements $\frac{1}{2}$-MAXSAT$(\cdot)$.

Define the following revelation mechanism:

Mechanism 6.1.

1. Let t be the fixed truth assignment from the definition of $f(\cdot)$ in Equation 6.2.
2. If t satisfies at least as many of the declared clauses as $\bar{t}$
3. then choose t as the outcome
4. else choose $\bar{t}$.

Theorem 6.12 *Mechanism 6.1 truthfully implements $f(\cdot)$ in dominant strategies in polynomial time for polynomial time bounded agents.*

Proof. Mechanism 6.1 is polynomial time since it need only compare the number of clauses in θ that are satisfied by two fixed truth assignments. Truth telling is certainly a polynomial time strategy so it suffices to show that truth telling constitutes a dominant strategy equilibrium.

Let t be the fixed truth assignment used in the definition of $f(\cdot)$ in Equation 6.2. Let θ_i be agent i's true type.

Case 1: t satisfies θ_i and $\bar{t}$ doesn't.

Removing literals from θ_i potentially decreases the number of clauses satisfied by t and does not affect the number of clauses satisfied by $\bar{t}$. Adding literals cannot increase the number of clauses satisfied by t and cannot decrease the number of clauses satisfied by $\bar{t}$. Since lying either leaves the number of clauses satisfied by t the same or decreases it and leaves the number of clauses satisfied by $\bar{t}$ the same or increases it, lying either leaves the outcome unaffected or results in an outcome that is worse for agent i.

Case 2: $\bar{t}$ satisfies θ_i and t doesn't. A symmetric argument applies to this case.

Case 3: Both t and $\bar{t}$ satisfy θ_i. In this case, agent i does not care which of the two truth assignments is chosen so lying cannot be beneficial or harmful.

Since lying is never beneficial, truth telling is a dominant strategy. ∎

Corollary 6.13 $\frac{1}{2}$-MAXSAT$(\cdot)$ *is strongly implementable in dominant strategies in polynomial time for polynomial time bounded agents.*

Proof. Let t be the fixed truth assignment used in the definition of $f(\cdot)$ in Equation 6.2. In the proof of Theorem 6.12, Case 3 is the only case in which lying can affect the outcome and not be harmful to the agent. However, we claim that if an agent who is indifferent between t and $\bar{t}$ affects the outcome by lying then the new outcome is still in $\frac{1}{2}$-MAXSAT$(\cdot)$.

To see this, suppose agent i's true type is θ_i but agent i falsely declares its type to be θ_i'. Further suppose that $g(\theta) \neq g(\theta_i', \theta_{-i})$ and both t and $\bar{t}$ satisfy agent i's true

type θ_i. Define $S(t', \theta')$ to be the number of clauses in θ' that a truth assignment t' satisfies.

Case 1: Both t and $\bar{t}$ satisfy θ_i'.

Then $g(\theta_i', \theta_{-i}) = g(\theta)$ contradicting our assumption.

Case 2: t does not satisfy θ_i' and $\bar{t}$ satisfies θ_i'.

Since $\bar{t}$ satisfies both θ_i and θ_i', we have $S(\bar{t}, (\theta_i', \theta_{-i})) = S(\bar{t}, \theta)$. Since t satisfies θ_i but not θ_i', we have $S(t, (\theta_i', \theta_{-i})) = S(t, \theta) - 1$. Since the number of clauses that $\bar{t}$ satisfies remains the same and the number of clauses that t satisfies decreases and since $g(\theta) \neq g(\theta_i', \theta_{-i})$, it must be the case that $g(\theta) = t$ and $g(\theta_i', \theta_{-i}) = \bar{t}$. This implies:

$$\begin{aligned} S(t, \theta) &\geq S(\bar{t}, \theta) \\ &= S(\bar{t}, (\theta_i', \theta_{-i})) \\ &> S(t, (\theta_i', \theta_{-i})) \\ &= S(t, \theta) - 1 \end{aligned}$$

which implies that $S(t, \theta) = S(\bar{t}, \theta)$. Since $S(t, \theta) + S(\bar{t}, \theta) \geq I$, $S(\bar{t}, \theta) \geq \frac{I}{2}$ where I is the number of agents. Therefore, $g(\theta_i', \theta_{-i}) = \bar{t} \in \frac{1}{2}\text{-}\mathsf{MAXSAT}(\theta)$.

Case 3: t satisfies θ_i' and $\bar{t}$ does not satisfy θ_i'.

Since t satisfies both θ_i and θ_i', we have $S(t, (\theta_i', \theta_{-i})) = S(t, \theta)$. Since $\bar{t}$ satisfies θ_i but not θ_i', we have $S(t, (\theta_i', \theta_{-i})) = S(t, \theta) - 1$. Therefore, since $g(\theta) \neq g(\theta_i', \theta_{-i})$, it must be the case that $g(\theta) = \bar{t}$ and $g(\theta_i', \theta_{-i}) = t$. This implies:

$$\begin{aligned} S(\bar{t}, \theta) &> S(t, \theta) \\ &= S(t, (\theta_i', \theta_{-i})) \\ &\geq S(\bar{t}, (\theta_i', \theta_{-i})) \\ &= S(\bar{t}, \theta) - 1 \end{aligned}$$

which implies that $S(t, \theta) = S(\bar{t}, \theta) - 1$. Since both t and $\bar{t}$ satisfy θ_i and every clause is satisfied by at least one of t and $\bar{t}$, it must be the case that $S(\bar{t}, \theta) + S(t, \theta) \geq I + 1$

which implies that $2S(t,\theta)+1 \geq I+1$. Hence, $S(t,\theta) \geq \frac{I}{2}$. Therefore, $g(\theta_i', \theta_{-i}) = t \in \frac{1}{2}$-MAXSAT$(\theta)$.

Thus, all dominant strategy equilibrium outcomes are in $\frac{1}{2}$-MAXSAT$(\cdot)$. ∎

6.2 Nash Implementation

We now consider mechanism design in environments with complete information using Nash equilibrium. First, we consider whether the exact social choice rule for multi-agent MAXSAT is strongly implementable. Then, we consider what is necessary for an approximate social choice rule to be strongly implementable in polynomial time for polynomial time bounded agents.

There are two properties of social choice rules that are extremely important for Nash implementation – *no veto power* and *monotonicity*. No veto power states that one agent cannot override the wishes of all the other agents. Monotonicity says that whenever the type profile changes from θ to θ' and the set of outcomes that an alternative $t \in F(\theta)$ is preferred to remains the same or expands then t must also be in $F(\theta')$. These two properties are formalized in the following definitions.

Definition 6.14 *A social choice rule $F(\cdot)$ satisfies* **no veto power** *if for any type profile θ such that $I-1$ agents rank an alternative t as their weakly most preferred choice, we have $t \in F(\theta)$.*

Let $L_i(t,\theta_i) = \{t' : u_i(t,\theta_i) \geq u_i(t',\theta_i)\}$. $L_i(t,\theta_i)$ is called agent i's *lower contour set* for t.

Definition 6.15 *A social choice rule $F(\cdot)$ is* **monotonic** *if for all type profiles θ and θ' and all outcomes t such that $t \in F(\theta)$ and $L_i(t,\theta_i) \subseteq L_i(t,\theta_i')$ for all i, we have $t \in F(\theta')$.*

The following result known as Maskin's Theorem establishes a necessary condition and a sufficient condition for Nash implementation:

Proposition 6.16 *(Maskin, 1985) If a social choice rule is fully Nash implementable then it is monotonic. If there are at least three agents then a social choice rule that is monotonic and satisfies no veto power is fully Nash implementable.*

Monotonicity is also required for full Nash implementation in polynomial time for polynomial time bounded agents.

Proposition 6.17 *If a social choice rule $F(\cdot)$ is fully Nash implementable in polynomial time for polynomial time bounded agents then it is monotonic.*

Proof. The result follows from Propositions 5.13 and 6.16. ∎

Proposition 6.18 *If the number of variables is at least two then* $\mathsf{MAXSAT}(\cdot)$ *is not monotonic.*

Proof. We provide a proof for the case of two agents and two variables. It can be generalized easily. Let $\theta = (x_1, \bar{x}_1)$. The maximum number of simultaneously satisfiable clauses in θ is 1. Let $t = x_1 x_2$ which is in $\mathsf{MAXSAT}(\theta)$. Let $\theta' = (x_2, \bar{x}_1)$. Since t satisfies agent 1 when the type profile is either θ or θ', $L_1(t, \theta_1) = L_1(t, \theta'_1)$. Since agent 2's type is the same in either case, $L_2(t, \theta_2) = L_2(t, \theta'_2)$. However, $t \notin \mathsf{MAXSAT}(\theta')$ which implies that $\mathsf{MAXSAT}(\cdot)$ is not monotonic. ∎

Even if $\mathsf{MAXSAT}(\cdot)$ were monotonic, by Proposition 5.18, it cannot be implemented in polynomial time for polynomial time bounded agents assuming $\mathsf{P} \neq \mathsf{NP}$. If the mechanism and the agents are restricted to polynomial time computation, the best one could hope for is to implement some approximation to $\mathsf{MAXSAT}(\cdot)$. Therefore, we need to find a constant c such that c-$\mathsf{MAXSAT}(\cdot)$ is monotonic even though $\mathsf{MAXSAT}(\cdot)$ is not. We also need to find a polynomial time mechanism that implements c-$\mathsf{MAXSAT}(\cdot)$.

There are several different mechanisms used to prove Maskin's theorem (Maskin, 1985; Repullo, 1987; Saijo, 1988; McKelvey, 1989). For example, Saijo (1988) defines a mechanism that fully implements a monotonic social choice rule satisfying no veto

power as follows. The action sets are defined by $A_i = \Theta_i \times \Theta_{i+1} \times X \times \{1, \ldots, I\}$. Each agent i declares his own type θ_i^i, the type of his neighbor θ_i^{i+1} where we take $I + 1 = 1$, an outcome x_i, and a number k_i between 1 and I. The outcome function $g(\cdot)$ is defined by the following rules:

Mechanism 6.2: Saijo (1988).

Let $a = \left[(\theta_i^i, \theta_i^{i+1}, x_i, k_i)\right]_{i=1}^{I}$ be the action profile.

Rule I: If $\theta_i^i = \theta_{i-1}^i$ and $x_i = x$ for all i and $x \in F(\theta_1^1, \ldots, \theta_I^I)$ then $g(a) = x$.

Rule II: If $\theta_i^i = \theta_{i-1}^i$ for all i except j or $j+1$, and $x_i = x$ for all i except j, and $x \in F(\theta_1^1, \ldots, \theta_{j-1}^{j-1}, \theta_{j-1}^j, \theta_{j+1}^{j+1}, \ldots, \theta_I^I)$ then

$$g(a) = \begin{cases} x_j & \text{if } x_j \in L_j(x, \theta_{j-1}^j), \\ x & \text{otherwise} \end{cases}$$

Rule III: If neither Rule I or II apply then let $n = \left(\sum_{i \in I} k_i\right) (\mathrm{mod} I) + 1$ and set $g(a) = x_n$.

Notice that Mechanism 6.2 checks whether a given outcome is a member of the social choice rule for the declared type profile. This check, which also appears in the mechanisms used by Repullo (1987) and Maskin (1985) corresponds to the following decision problem when $F(\cdot)$ is c-MAXSAT$(\cdot)$:

Definition 6.19 c-MAXSAT **Membership Problem**
Let $V = \{v_1, \ldots, v_n\}$ be a set of boolean variables, let θ be a set of clauses over V and let t be a truth assignment to the variables in V. Does t satisfy at least c times the maximum number of simultaneously satisfiable clauses in θ?

Proposition 6.20 shows that this problem is NP-hard which implies that Saijo's mechanism is not a polynomial time mechanism assuming P $\neq$ NP.

Proposition 6.20 *Let $c \in (0,1)$ be such that there is a polynomial time approximation algorithm for* c-MAXSAT. *The* c-MAXSAT *Membership problem is* NP-hard.

Proof. It suffices to show that if this problem can be solved in polynomial time then there is a polynomial time algorithm for SAT.

Suppose we are given a set of clauses θ and we want to determine whether there is a truth assignment that satisfies all the clauses in θ. Let $m(\theta)$ be the maximum number of clauses in θ that can be simultaneously satisfied. The set of clauses θ is satisfiable if and only if $m(\theta) = I$ where I is the number of clauses in θ.

Consider the following algorithm for finding $m(\theta)$.

1. Use the polynomial time approximation algorithm for c-MAXSAT to choose a truth assignment t that satisfies more than $cm(\theta)$ clauses.
2. Let p be the number of clauses in θ that t satisfies.
3. Let $k = 0$.
4. Repeat
5. $k \leftarrow k + 1$.
6. Extend the set of variables to the set V^k by adding variable v_{n+k}.
7. Extend the set of clauses to θ^k by adding clause v_{n+k}.
8. Extend t to V^k by setting $t(v_{n+k})$ to False.
9. Until t extended to V^k does not satisfy $cm(\theta^k)$ clauses.
10. Return $\lfloor (1/c)p \rfloor - k + 1$.

It is not hard to show that the algorithm returns the correct value of $m(\theta)$. Since, for each k, v_{n+k} does not appear in any clauses other than θ^k, any truth assignment for V^{k-1} can be extended to satisfy v_{n+k} without affecting the number of clauses it satisfied in θ. Therefore, $m(\theta^k) = m(\theta) + k$. Since t does not satisfy v_{n+k} for any $k \geq 1$, t satisfies p of the clauses in each of the θ^k. The repetition is guaranteed to

terminate by the time $k = \lceil \frac{1-c}{c} m(\theta) \rceil + 1$ since t satisfies at most $m(\theta)$ clauses in θ^k and $m(\theta^k) = m(\theta) + \lceil \frac{1-c}{c} m(\theta) \rceil + 1 \geq \frac{m(\theta)}{c} + 1$ when $k = \lceil \frac{1-c}{c} m(\theta) \rceil + 1$. Since the algorithm begins with a truth assignment that satisfies at least $cm(\theta)$ clauses, it stops extending the set of clauses when k is the smallest integer such that $p < c(m(\theta) + k)$. Therefore, k is the smallest integer such that $\frac{1}{c}p - k < m(\theta)$ which implies $m(\theta) = \lfloor (1/c)p \rfloor - k + 1$.

To evaluate the asymptotic running time of the algorithm, first note that, given our assumptions, we know that each of the steps of this algorithm other than Step 9 takes polynomial time. Furthermore, the maximum number of iterations is $\lceil \frac{1-c}{c} I \rceil + 1$ so there are a polynomial number of iterations. Hence, if Step 9 takes only a polynomial amount of time then the entire algorithm is polynomial time.

We can use this algorithm to decide SAT by comparing the output to the number of clauses in θ. The CNF formula θ is satisfiable if and only if the value for $m(\theta)$ returned in Step 10 is I. Therefore, if Step 9 takes only a polynomial amount of time we have a polynomial time algorithm to decide SAT. Thus, the c-MAXSAT Membership problem is NP-hard. ∎

Theorem 6.21 *Let $c \in (0,1)$ be such that there is a polynomial time approximation algorithm for c-MAXSAT. If Saijo's mechanism with $F(\cdot) = c\text{-MAXSAT}(\cdot)$ is a polynomial time mechanism then* P = NP.

Proof. For the mechanism to check whether $x \in F(\theta_1^1, \ldots, \theta_I^I)$ in Rules I and II, it requires a solution to the c-MAXSAT Membership problem which Proposition 6.20 shows is NP-hard. Hence, if the mechanism is polynomial time then P = NP. ∎

In Saijo's mechanism, for a strategy profile to be an equilibrium strategy profile, it must always be the case that all but at most one of the agents choose an outcome that is in $F(\theta)$. To see this, observe that if more than one agent declares an alternative which is not in $F(\theta)$ then Rule III applies. There can be no equilibrium outcome caused by Rule III since each agent i that is not satisfied by the outcome has incentive to deviate by specifying a k_i that changes the outcome to x_i and specifying an x_i that satisfies θ_i. Theorem 6.21 implies that if c is such that Saijo's mechanism with

$F(\cdot) = c\text{-MAXSAT}(\cdot)$ is polynomial time then the agents cannot find an alternative in $c\text{-MAXSAT}(\cdot)$ in polynomial time. Therefore, either the mechanism is not polynomial time or there is no polynomial time equilibrium strategy profile. Whether there is a general mechanism to implement any monotonic social choice rule satisfying no veto power that does not check membership in $F(\cdot)$ is an open question.

As discussed in Section 5.2, we are willing to settle for strong implementation as opposed to full implementation of $c\text{-MAXSAT}(\cdot)$. If $c\text{-MAXSAT}(\cdot)$ is strongly implementable then there must be some fully implementable social choice rule $F(\cdot)$ such that $F(\theta) \subseteq c\text{-MAXSAT}(\theta)$ for all θ. Thus, it is sufficient for our purposes to find a social choice rule $F(\cdot)$ such that the following three conditions hold:

1. $F(\theta) \subseteq c\text{-MAXSAT}(\theta)$ for all θ
2. $F(\cdot)$ is monotonic and satisfies no veto power
3. Checking membership in $F(\cdot)$ is easy.

Since, we have a mechanism that strongly implements $\frac{1}{2}\text{-MAXSAT}(\cdot)$ in dominant strategies (Mechanism 6.1), it does not seem particularly important to develop a mechanism that strongly Nash implements $\frac{1}{2}\text{-MAXSAT}(\cdot)$. However, Mechanism 6.1 is tied to a particular truth assignment t. The outcome is always either t or $\bar{t}$. An unfortunate choice of t eliminates many outcomes that would be more desirable from a social viewpoint. For example, let $I = 4$, $\theta = (x_1, x_1, \bar{x}_2, \bar{x}_2)$ and $t = x_1x_2$. The number of clauses in θ that are satisfied by t is 2. Both of Johnson's algorithms would choose $t' = x_1\bar{x}_2$ which is an optimal outcome. A better option would be to implement a social choice rule that included any truth assignment that satisfied as many clauses as t and $\bar{t}$ for some fixed t. In other words, it would be better to implement the following social choice rule where, for all truth assignments $\hat{t}$, $S(\theta, \hat{t})$ is defined to be the number clauses in θ that $\hat{t}$ satisfies:

$$\begin{aligned} F(\theta) &= \{t' : S(\theta, t') \geq \max(S(\theta, t), S(\theta, \bar{t}))\} \\ &\quad \cup \{t' : t' \text{ satisfies at least } I - 1 \text{ clauses in } \theta\}. \end{aligned} \tag{6.3}$$

Using a mechanism like Saijo's (Mechanism 6.2) would allow the agents to achieve better outcomes than the t or $\bar{t}$ fixed by Mechanism 6.1. It could be argued that full Nash implementation of $\frac{1}{2}$-MAXSAT$(\cdot)$ using Saijo's mechanism would be better than strong dominant strategy implementation using Mechanism 6.1. Unfortunately, although $F(\cdot)$ satisfies no veto power it is not fully implementable in Nash equilibrium.

Proposition 6.22 *If there are at least two variables, the social choice rule defined in Equation 6.3 is not monotonic.*

Proof. We provide a proof for the case of two agents and two variables. It can be easily generalized. Let $\theta = (x_1, \bar{x}_2)$ and $\theta' = (x_1 \vee \bar{x}_2, \bar{x}_2)$ and $t = x_1 x_2$. We have $t \in F(\theta)$ but $t \notin F(\theta')$. However, for each agent i, $L_i(t, \theta_i) \subseteq L_i(t, \theta_i')$ so $F(\cdot)$ is not monotonic. ∎

Although the social choice rule defined in Equation 6.3 is not fully Nash implementable, Theorem 6.23 defines an alternative social choice rule that is fully Nash implementable and which maps each type profile θ to a subset of $\frac{1}{2}$-MAXSAT(θ).

Theorem 6.23 *Let $F(\theta)$ be the set of outcomes t such that t satisfies at least $\frac{I}{2}$ clauses in θ. If there are at least three agents, $F(\cdot)$ is fully Nash implementable using Saijo's mechanism.*

Proof. $F(\cdot)$ clearly satisfies no veto power since there are at least three agents. Let θ, θ' and t be such that $t \in F(\theta)$ and $L_i(t, \theta_i) \subseteq L_i(t, \theta_i')$ for all i. Since every clause has a satisfying truth assignment, if t satisfies θ_i then t must satisfy θ_i'. Therefore, t satisfies at least half of the clauses in θ' so $t \in F(\theta')$. Hence, $F(\cdot)$ is monotonic. Since $F(\cdot)$ is monotonic and satisfies no veto power, Saijo's mechanism fully Nash implements $F(\cdot)$. ∎

Corollary 6.24 $\frac{1}{2}$-MAXSAT$(\cdot)$ *is strongly Nash implementable in polynomial time for polynomial time bounded agents.*

Proof. Let $F(\cdot)$ be the the social choice rule used in Theorem 6.23. It is easy to check

whether a truth assignment t is in $F(\theta)$ so Saijo's mechanism is polynomial time computable. Let θ be the agents true type profile. For any truth assignment t such that $t \in F(\theta)$ and any $k, 1 \leq k \leq I$, let $s_i(\theta) = (\theta_i, \theta_{i+1}, t, k)$ for all i. The strategy profile $s(\theta) = (s_1(\theta), \ldots, s_I(\theta))$ is a Nash equilibrium.[4] Such a strategy profile is clearly easy for the agents to compute since the agents can use the complement algorithm to find a $t \in F(\theta)$. Therefore, $F(\cdot)$ is fully Nash implementable in polynomial time for polynomial time bounded agents. Since $F(\theta)$ is a subset of $\frac{1}{2}$-MAXSAT(θ) for all θ, $\frac{1}{2}$-MAXSAT$(\cdot)$ is strongly Nash implementable in polynomial time for polynomial time bounded agents. ∎

In Section 6.3, we show that in general $\frac{1}{2}$-MAXSAT$(\cdot)$ is the best approximation for MAXSAT$(\cdot)$ that can be strongly Nash implemented in polynomial time for polynomial time bounded agents.

6.3 Upper Bounds on Approximability

Existing work in mechanism design shows that the set of social choice rules that are implementable in Nash equilibrium is much smaller than the set of social choice rules that are implementable in refinements such as undominated Nash equilibrium[5] (Palfrey and Srivastava, 1991; Jackson, Palfrey, and Srivastava, 1994) and subgame perfect equilibrium[6] (Moore and Repullo, 1988). Palfrey and Srivastava (1991) point out that for a social choice rule to be fully implementable in Nash, undominated Nash, or subgame perfect equilibrium, it must satisfy Property Q defined below. Property Q must also be satisfied for full implementation in dominant strategies.

[4] See Saijo (1988) p. 698.

[5] An undominated Nash equilibrium is a Nash equilibrium in which no agent plays a weakly dominated strategy.

[6] Subgame perfect equilibrium is defined for games that are played in stages. A strategy profile is a subgame perfect equilibrium if it is a Nash equilibrium in every subgame.

Definition 6.25 *(Palfrey and Srivastava, 1991) A social choice rule satisfies* **Property Q** *if, when θ, θ' and t are such that $t \in F(\theta)$ and $t \notin F(\theta')$, there is an i such that $\theta_i \neq \theta'_i$ and agent i is not completely indifferent under θ'_i.*

Theorem 6.26 *Let $F(\cdot)$ be a social choice rule such that, for some constant $c \in (0,1)$, $F(\theta) \subseteq c\text{-MAXSAT}(\theta)$ for all θ. Suppose for some type profile θ there exists a truth assignment $t \in F(\theta)$ that satisfies less than cI clauses. Then $F(\cdot)$ does not satisfy Property Q .*

Proof. Let $F(\cdot)$, θ, c, and t be as defined above. For each i such that t satisfies θ_i, let l_i be any literal in θ_i such that $t(l_i) =$ True. Create θ' from θ by adding $\bar{l}_i$ to θ_i for each clause θ_i that t satisfies. For each i, either $\theta'_i = \theta_i$ or agent i is completely indifferent under type θ'_i. Hence, for $F(\cdot)$ to satisfy Property Q , it must be the case that $t \in F(\theta')$.

Since $\bar{t}$ must satisfy every clause that t does not satisfy, and $\bar{t}$ must satisfy θ'_i if t satisfies θ_i, $m(\theta') = I$. Since, the clauses in θ that t doesn't satisfy also appear in θ', the number of clauses in θ' that t satisfies is less than $cI = cm(\theta')$. Therefore, $t \notin c\text{-MAXSAT}(\theta')$. This implies that $t \notin F(\theta')$ since $F(\theta) \subseteq c\text{-MAXSAT}(\theta)$ for all θ. Thus, $F(\cdot)$ does not satisfy Property Q . ∎

Theorem 6.26 says that the only approximate social choice rules for MAXSAT that can be fully implemented in dominant strategy, Nash, undominated Nash, or subgame perfect equilibrium are those that guarantee the number of agents that the outcomes satisfy is a constant times the total number of agents rather than a constant times the maximum number of simultaneously satisfiable agents. The following corollary shows that this is impossible for $c > 1/2$.

Corollary 6.27 *For any c such that $\lceil cI \rceil > \lceil \frac{I}{2} \rceil$, c-MAXSAT$(\cdot)$ cannot be strongly implemented in dominant strategy, Nash, undominated Nash or subgame perfect equilibrium.*

Proof. Let Γ be any mechanism that strongly implements c-MAXSAT$(\cdot)$. Define $E : \Theta_1 \times \ldots \times \Theta_I \rightarrow X$ such that $E(\theta)$ is the set of equilibrium outcomes of Γ for type profile θ. By definition, Γ fully implements $E(\cdot)$. Since Γ strongly implements c-MAXSAT$(\cdot)$, $E(\theta) \subseteq c$-MAXSAT(θ) for all θ.

Let $\theta'_i = x_1$ for $1 \leq i \leq \lfloor \frac{I}{2} \rfloor$. Let $\theta'_i = \bar{x}_1$ for $\lfloor \frac{I}{2} \rfloor + 1 \leq i \leq I$. Since the maximum number of simultaneously satisfiable clauses is $\lceil \frac{I}{2} \rceil$, no truth assignment can satisfy more than $\lceil \frac{I}{2} \rceil$ clauses. Therefore, any $t \in E(\theta')$ satisfies less than cI clauses in θ'. According to Theorem 6.26, this implies that $E(\cdot)$ does not satisfy Property Q. But then $E(\cdot)$ is not fully implementable which is a contradiction. ∎

By Proposition 5.13, Corollary 6.27 also holds for strong implementation in polynomial time for polynomial time bounded agents.

6.4 Repeated Implementation

To this point, we have only considered mechanism design for collective decisions that occur in isolation. The agents see the mechanism once and the mechanism sees each agent once. There is no opportunity for the mechanism or the agents to gain information from their previous encounters. In many applications, repeated interaction may be the more likely scenario. In this section, we briefly consider repeated implementation.

There is currently little in the economics literature on repeated implementation. Kalai and Ledyard (1998) present a very simple model of repeated implementation to show that in some cases repetition may make things easier for the mechanism designer. They show that if the mechanism is more patient that the agents, any social choice function can be implemented in dominant strategies in the long run. This is in sharp contrast to the Gibbard-Satterthwaite Theorem (Proposition 6.2). In this section, we evaluate Kalai and Ledyard's model from a computational perspective again using MAXSAT to focus our analysis. We begin with an overview of the model.

Definition 6.28 *An* **infinitely repeated mechanism design problem** *consists of the following:*

- *a set of I agents that must make a collective choice over some finite set of possible outcomes X.*
- *X^∞ is the set of infinite sequences over X.*
- *for each agent i,*
  - *a set Θ_i of possible* **types**.
  - *a* **utility function** $u_i : X \times \Theta_i \to \mathcal{R}$
  - *a* **discount factor** $\delta_i \in (0, 1]$.
  - *Agent i's* **utility for the repeated game** *is defined to be:*

$$u_i(\theta_i, x^\infty) = \sum_{t=1}^{\infty} \delta_i^{t-1} u_i(\theta_i, x^i) \tag{6.4}$$

  where $x^\infty = (x^1, x^2, \ldots, x^t, \ldots)$ is an infinite sequence of outcomes.

  Each agent is assumed to be trying to maximize its utility.
- *A social choice rule $F : \Theta_1 \times \ldots \times \Theta_I \to 2^{X^\infty} \setminus \emptyset$.*

Kalai and Ledyard (1998) study situations in which the mechanism designer is only concerned with eventual implementation while the agents care more about their immediate payoff. The mechanism is considered successful if the social choice rule is satisfied from some time onward.

Definition 6.29 *A revelation mechanism with outcome function $g(\cdot)$* **patiently dominance-implements** *a social choice rule $F(\cdot) = (F_1(\cdot), F_2(\cdot), \ldots, F_n(\cdot), \ldots)$ if, for each type profile θ,*

1. *θ_i is a dominant strategy for each agent i*
2. *for some time N, $g_n(\theta) \in F_n(\theta)$ for all $n \geq N$.*

The following restriction on the agents' types is sufficient to permit any social choice function to be patiently dominance-implemented.

Definition 6.30 *Types are* **separable** *if for every player i and for all distinct types θ_i and $\hat{\theta}_i$ there is a pair of outcomes x and y such that $u_i(\theta_i, x) < u_i(\theta_i, y)$ and $u_i(\hat{\theta}_i, x) > u_i(\hat{\theta}_i, y)$.*

Proposition 6.31 (Kalai and Ledyard, 1998) *If θ_i is finite for all i then in a types separable environment with bounded utility functions every social choice function is patiently dominance implementable.*

The mechanism used to obtain Proposition 6.31 is formally described in Mechanism 6.3 below. The idea behind the mechanism is that each agent i will be a dictator for a time period of length D_i. During each stage of agent i's dictatorship, the mechanism chooses a pair of types for agent i and a separating pair of alternatives for those types. The outcome of the stage is the alternative that is most preferred by the agent given its declared type. Lying on the part of the agent results in at least one outcome of his dictatorial period being suboptimal. The dictatorial phase is followed by a "cooling off" phase where the outcome is some fixed alternative that does not depend on the declared types of the agents. Since the mechanism knows the discount factor for each agent, the length of this cooling off phase can be made large enough that any benefit from lying that occurs after the cooling off phase will be overwhelmed by the loss that occurs during the agent's dictatorial period.

Mechanism 6.3.

1. For i from 1 to I
2. Let D_i be the number of unordered pairs of distinct types in Θ_i.
3. Let T_i be any ordering of the unordered pairs of distinct types in Θ_i.
4. For $k = 1$ to D_i
5. Let $p_i^k = (x_i^k, y_i^k)$ be a pair of alternatives that separates the types in

the k-th pair from T_i.

6. For i from 1 to I

7. For n from 1 to D_i

8. If $u_i(x_i^k, \theta_i) \geq u_i(y_i^k, \theta_i)$

9. then let the outcome be x_i^k

10. else let the outcome be y_i^k

11. Let the outcome be some fixed alternative z for some number of phases which depends on the discount factors.

12. For each stage n beyond the cooling off phase above,

13. Let the outcome be $f_n(\theta)$.

Unfortunately, we cannot use Mechanism 6.3 to implement $\mathsf{MAXSAT}(\cdot)$ since, as the following proposition shows, MAXSAT does not provide a types separable environment.

Proposition 6.32 *Types in* MAXSAT *are not separable.*

Proof. Let $\theta_i = (x_1)$ and $\theta_i' = (x_1 \vee x_2)$. Every truth assignment that satisfies θ_i also satisfies θ_i'. Therefore, $u_i(\theta_i, x) \leq u_i(\theta_i', x)$ for all outcomes $x \in X$. ∎

We know from Proposition 6.9 that $\mathsf{MAXSAT}(\cdot)$ is truthfully implementable in dominant strategies so it is a trivial exercise to implement it in a repeated situation. Suppose, however, that the time that the repeated mechanism is allowed to use at each stage is only polynomial in the size of the agents' types and the problem parameters. The mechanism cannot choose an outcome in $\mathsf{MAXSAT}(\theta)$ at every stage in polynomial time. However, the mechanism can use the repetition to its advantage by spending some time at each stage trying to find an optimal solution. It will require an exponential number of stages to find an optimal solution but if the mechanism is only concerned with implementation in the long run this is acceptable.

Consider the following revelation mechanism which makes use of the complement mechanism defined in Mechanism 6.1:

Mechanism 6.4.

1. Let $(\theta_1, \ldots, \theta_I)$ be the type profile declared by the agents.
2. Let $t_1, \ldots, t_{2^n}$ be the lexicographic ordering of all possible truth assignments over boolean variables $x_1, \ldots, x_n$ as defined in the proof of Proposition 6.9.
3. Let $MAX = 0$
4. For stage j from 1 to 2^n
5. if t_j satisfies at least as many clause in θ as $\bar{t}_j$
6. then choose t_j as the outcome
7. else choose $\bar{t}_j$ as the outcome
8. if $S(\theta, t_j) > MAX$ then $m = j$
9. After stage 2^n, choose t_m as the outcome.

Proposition 6.33 *Mechanism 6.4 patiently dominance-implements* $\mathsf{MAXSAT}(\cdot)$. *The time used at each stage is polynomial in the size of the type profile and the problem parameters. Furthermore, the outcome at every stage is in* $\frac{1}{2}$-$\mathsf{MAXSAT}(\theta)$.

Proof. Since the mechanism examines every truth assignment in the first 2^n stages, $t_m \in \mathsf{MAXSAT}(\theta)$. Hence, from stage $2^n + 1$ onward the outcome is in $\mathsf{MAXSAT}(\theta)$ so it suffices to show that truth-telling is a dominant strategy for each agent. Since truth-telling is dominant for the complement mechanism based on any truth assignment, lying cannot result in a better outcome at any stage prior to stage $2^n + 1$. Truth-telling is also dominant for the mechanism given in the proof of Proposition 6.9. The outcome of this mechanism is the outcome chosen by Mechanism 6.4 from stage $2^n + 1$ onward so lying cannot result in a better outcome at any stage after stage 2^n. Therefore, truth-telling is a dominant strategy for each agent. ∎

The model presented in Kalai and Ledyard (1998) is intentionally simplistic. For one thing, it is not clear how the mechanism comes to know the agents' discount factors. Without this knowledge the mechanism would not be able to determine how long a cooling off phase is required. The mechanism we presented for MAXSAT$(\cdot)$ does not need a cooling off phase and does not need to know the discount factors. However, an exponential number of stages occurs before the mechanism is guaranteed to choose an optimal outcome. This may or may not be acceptable depending on how strictly one views the assumption that the mechanism is concerned only with the eventual outcomes.

6.5 Conclusion

We have formalized the computational problem of mechanism design in such a way that classic results from the field can be applied. Using a multiagent version of MAXSAT, we have investigated the difficulties that arise in applying these results to NP-hard optimization problems. Our results are summarized in Table 6.1.

We have demonstrated that, despite the impossibility results regarding dominant strategy implementation, it is possible to implement an approximate social choice rule for MAXSAT in dominant strategy equilibrium. We have also investigated computational issues in repeated implementation in dominant strategies using the model of Kalai and Ledyard (1998). The repetition permitted patiently dominance-implementation of MAXSAT$(\cdot)$ even when the mechanism is restricted to polynomial time at each stage. An exponential number of stages is required, of course, before optimal outcomes can be guaranteed. Future work in repeated implementation should consider more realistic models which take into account that the agents' preferences might change over time or even depend on previous outcomes. It should also consider environments in which the agents have less than complete information about each other and perhaps the workings of the mechanism.

With regard to Nash implementation, we showed that approximation was benefi-

Table 6.1: A summary of results on the implementability of MAXSAT$(\cdot)$ and c-MAXSAT$(\cdot)$. We use $\neg$ before the result to indicate that the result is negative. For example, "$\neg$Strong" indicates that strong implementation was shown to be impossible while "Strong" indicates that the social choice rule was shown to be strongly implementable. "Strong-Poly" indicates that the social choice rule was shown to be strongly implementable in polynomial time for polynomial time bounded agents. In the repeated context, "Poly" means the implementation can be done using polynomial time at every stage. The negative result regarding strong implementation of c-MAXSAT$(\dot{)}$ for $c > 1/2$ applies to undominanted Nash and subgame perfect equilibrium as well.

Rule	Dominant Strategy	Nash	Patiently dominance
MAXSAT$(\cdot)$	Strong	$\neg$Full	Poly
$\frac{1}{2}$-MAXSAT$(\cdot)$	Strong–Poly	Strong–Poly	—
$> \frac{1}{2}$-MAXSAT$(\cdot)$	$\neg$Strong	$\neg$Strong	—

cial in two ways. It enabled strong Nash implementation when the exact social choice rule was not monotonic and it enabled polynomial time implementation when finding a member of the exact rule was NP-hard. In addition, we showed that the standard mechanisms[7] used to prove Maskin's theorem cannot be computed in polynomial time for an approximate social choice rule c-MAXSAT such that there is a polynomial time c-approximation algorithm for MAXSAT. This implies that these mechanisms do not fully Nash implement c-MAXSAT$(\cdot)$ for any c in polynomial time for polynomial time bounded agents. Since we are more concerned with strong implementation than full implementation, fully implementing a subset of c-MAXSAT$(\cdot)$ would be acceptable. However, we showed that the largest c for which c-MAXSAT$(\cdot)$ is strongly Nash implementable in general is $c = 1/2$. This upper bound applies to dominant strategy, Nash, undominated Nash, and subgame perfect implementation.

It is somewhat remarkable that although there are approximation algorithms that

[7]We are currently evaluating the complexity of the mechanism due to McKelvey (1989).

provide better lower bounds than 1/2, none of these algorithms produce implementing mechanisms for any of the four main equilibrium concepts studied for environments with complete information. Since the multiagent MAXSAT problem as we have defined it is a special case of the more general problem in which each agent specifies a separate CNF formula, these negative results apply to the more general problem as well. These results along with the result given in Ronen (1999) for dominant strategy implementation suggest that our ability to implement approximate social choice rules is very limited even when we have a number of approximation algorithms for the problem in question.

Appendix A

Overview of Game Theory

This section provides a short overview of game theory. For a more detailed introduction see Owen (1995), Osborne and Rubinstein (1990), Myerson (1991) or Luce and Raiffa (1957).

A.1 Two-Person Zero-Sum Games

Definition A.1 *A* **two-person zero-sum** *game is a 3-tuple* $G = \langle A_1, A_2, r(\cdot)\rangle$ *where:*

A_1 *is a finite set of strategies for player 1.*

A_2 *is a finite set of strategies for player 2.*

$r : A_1 \times A_2 \to \mathcal{R}$ *is a payoff function.*

Player 1 attempts to maximize the payoff while Player 2 attempts to minimize the payoff.

A two-person zero-sum game $G = \langle A_1, A_2, r(\cdot)\rangle$ can be represented by a real valued matrix where the set of rows corresponds to A_1, the set of columns corresponds to A_2, and the entry in row a_1 column a_2 is $r(a_1, a_2)$. Player 1, therefore, chooses a row so as to maximize the payoff while Player 2 chooses a column so as to minimize the payoff.

Game theory attempts to determine which outcome would be expected to occur if the players played "rationally". There are many different notions of what is meant by rational but they all assume that the players care only about maximizing or minimizing the payoff. One notion of a rational outcome is the outcome that each player can guarantee. The largest payoff that Player 1 can guarantee is called his *maxmin value in pure strategies*. Similarly, the smallest payoff that Player 2 can guarantee is called his *minmax value in pure strategies*.

Definition A.2 *Player 1's* **maxmin value in pure strategies** *is* $\max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2)$ *while Player 2's* **minmax value in pure strategies** *is* $\min_{a_2 \in A_2} \max_{a_1 \in A_1} r(a_1, a_2)$.

Example A.3 *Let G be defined by* $\begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$

$\max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) = 1 \quad \min_{a_2 \in A_2} \max_{a_1 \in A_1} r(a_1, a_2) = 1.$

Example A.4 *Let G be defined by* $\begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$

$\max_{a_1 \in A_1} \min_{a_2 \in A_2} r(a_1, a_2) = 1, \quad \min_{a_2 \in A_2} \max_{a_1 \in A_1} r(a_1, a_2) = 2.$

When Player 1's maxmin value is equal to Player 2's minmax value that value is said to be the *value of the game in pure strategies*. It is a single outcome that is predicted as the outcome of the game. As can be seen from Example A.4, not every game has a value in pure strategies. If we allow the players to randomize over their strategies, however, every finite two-person zero-sum game has a value.

Definition A.5 *A* **mixed strategy** *for a player is a probability distribution over the set of pure strategies available to the player.*

Definition A.6 *Player 1's* **maxmin value in mixed strategies** *is:*

$\max_{\sigma \in \Delta(A_1)} \min_{a_2 \in A_2} E_\sigma[r(a_1, a_2)]$ *while Player 2's* **minmax value in mixed strategies** *is:*

$\min_{\gamma \in \Delta(A_2)} \max_{a_1 \in A_1} E_\gamma[r(a_1, a_2)]$ *where* $E_\alpha[r(a_1, a_2)]$ *is the expected value of the payoff function given mixed strategy* α *is being played by Player 1 or Player 2 as the case may be.*

The following theorem known as the Minimax Theorem was proven by von Neumann (1928).

Proposition A.7 (Minimax Theorem)

$$\begin{aligned} \max_{\sigma \in \Delta(A_1)} \min_{a_2 \in A_2} E_\sigma[r(a_1, a_2)] &= \min_{\gamma \in \Delta(A_2)} \max_{a_1 \in A_1} E_\gamma[r(a_1, a_2)] \\ &= \max_{\sigma \in \Delta(A_1)} \min_{\gamma \in \Delta(A_2)} E_{\sigma,\gamma}[r(a_1, a_2)] \end{aligned}$$

Definition A.8 *The* **value** *of a two-person zero-sum game in mixed strategies is* $\max_{\sigma \in \Delta(A_1)} \min_{\gamma \in \Delta(A_2)} E_{\sigma,\gamma}[r(a_1, a_2)]$.

A.1.1 Repeated Two-Person Zero-Sum Games

Definition A.9 *A* **repeated game** *consists of a sequence of stages such that at each stage the same game G is played. G is called the* **stage game**. *We refer to the pure strategies played at each stage of a repeated game as* **actions**.

A repeated game could be finite in which case the game is played for N stages for some finite integer $N \geq 2$ or infinite in which case the game is repeated indefinitely. For a finitely repeated game the payoff is defined as

$$\frac{1}{N} \sum_{i=1}^{N} r(a_1^i, a_2^i)$$

where a_1^i and a_2^i are the actions played at stage i by Players 1 and 2 respectively.

A **pure strategy** for a player in an N-stage game is a specification of the action the player chooses at each stage given the complete history of actions chosen so far. A **mixed strategy** is a probability distribution over the set of pure strategies.

Definition A.10 *Let* $\Upsilon_0 = \emptyset$. *Let* $\Upsilon_i = \{((a_1^1, a_2^1), \ldots, (a_1^i, a_2^i)) : a_1^j \in A_1, a_2^j \in A_2, \text{ for } 1 \leq j \leq i\}$. *For an N-stage game,* $\Upsilon = \bigcup_{i=0}^{N-1} \Upsilon_i$ *is the set of all possible histories of play. For an infinitely repeated game,* $\Upsilon = \bigcup_{i=0}^{\infty} \Upsilon_i$ *is the set of all possible histories of play.*

Definition A.11 *Let* A_k *be the set of actions for Player k in the stage game. A* **pure strategy** *s for Player k in a repeated game is a function from histories to actions, i.e.,* $s : \Upsilon \rightarrow A_k$. *Let* S_k *be the set of pure strategies for Player k. A* **mixed strategy** *for Player k is a probability distribution over the* S_k.

A.2 General sum games

In Part II of the thesis, we use results from the theory of general sum games.

Definition A.12 *An* n**-person general sum game** *is an* $(n+1)$*-tuple* $G = \langle A_1, A_2, \ldots, A_n, r(\cdot) \rangle$ *where:*

> A_i *is a finite set of strategies for Player i,* $1 \leq i \leq n$.
>
> $u : A_1 \times A_2 \rightarrow \mathcal{R}^n$ *is a* **payoff** *or* **utility function** *where* $u_i(a_1, \ldots, a_n)$ *is the payoff to Player i given that Player j plays strategy* a_j *for each j,* $1 \leq j \leq n$.

Each player attempts to maximize his own payoff without regard for any other players' payoff.

An n-person general sum game can be represented by an n-dimensional matrix in which each entry of the matrix is an n-tuple $(r_1, \ldots, r_n)$ indicating the vector of payoffs for the players. For example, the following matrix represents a two-person general sum game in which Player I chooses a row and Player II chooses a column:

$$\begin{array}{c} \\ \mathrm{I} \end{array} \begin{pmatrix} 3,4 & 1,3 & 2,4 \\ 0,1 & 1,1 & 2,0 \\ 2,2 & 0,1 & 1,0 \end{pmatrix}$$

As with zero-sum games, in general sum games we are interested in determining what outcomes to expect from a game when it is played by "rational" players. Rational outcomes are called *equilibrium* outcomes. There are many different equilibrium concepts used in game theory. The two equilibrium concepts we discuss in this thesis are *dominant strategy* and *Nash* equilibrium.

A strategy $a_i \in A_i$ is a *dominant strategy* for Player i if no matter what strategies the other players play, Player i can do no better than to play a_i. A strategy profile in which every player plays a dominant strategy is called a dominant strategy equilibrium. Formally:[1]

Definition A.13 *A strategy profile* $(a_1, \ldots, a_I)$ *for an* I*-person game is a* **dominant strategy equilibrium** *if, for each player* i*,*

$$u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i})$$

for all $a_i' \in A_i$ *and all* $a_{-i} \in A_{-i}$*.*

To understand the concept of a Nash equilibrium, consider what would happen if prior to announcing their strategies for a game, the players got together and agreed to play a particular strategy profile. If no player could increase his payoff by unilaterally deviating from this agreement, then the strategy profile is a (pure) Nash equilibrium. Formally:

Definition A.14 *A strategy profile* $(a_1, \ldots, a_I)$ *for an* I*-person game is a* **Nash equilibrium** *if, for each player* i*,*

$$u_i(a_i, a_{-i}) \geq u_i(a_i', a_{-i})$$

[1]We use the notation a_{-i} to indicate the vector consisting of all elements of vector $(a_1, \ldots, a_I)$ except element i.

for all $a_i' \in A_i$.

Consider the following two-person game:

$$\begin{array}{c} \\ a_1 \\ a_2 \end{array} \begin{array}{c} \begin{array}{cc} b_1 & b_2 \end{array} \\ \left(\begin{array}{cc} 1,1 & 2,0 \\ 0,1 & 2,1 \end{array} \right) \end{array}$$

The strategy profile (a_1, b_1) is both a dominant strategy and a Nash equilibrium while (a_2, b_2) is a Nash equilibrium but not a dominant strategy equilibrium. Neither of the other two strategy profiles are dominant strategy or Nash equilibrium.

Appendix B

Review of Complexity Theory

This section provides a very brief review of complexity theory. For a more detailed discussion see Garey and Johnson (1979).

Definition B.1 *A* **decision problem** *is a problem that requires a yes or no answer. Formally, a decision problem* Π *is a set* D_Π *of instances and a set* $Y_\Pi \subseteq D_\Pi$ *of Yes instances.*

For example, one of the most widely studied decisions problems is the *Satisfiability* problem:

Definition B.2 SAT *is the decision problem defined as follows: Let* V *be a set of boolean variables and* θ *be a conjunctive normal form (CNF)*[1] *formula over those variables. Is there a truth assignment to the variables in* V *such that every clause in* θ *is satisfied?*

An *instance* of SAT is a set of boolean variables and a CNF formula over those variables. For example, $(\{x_1, x_2\}, (x_1 \vee x_2) \wedge (\bar{x}_1))$ is an instance of SAT.

[1] A CNF formula is a conjunction of disjunctions. For example, $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3)$ is a CNF formula. Each disjunction is called a clause. A clause is satisfied if at least one of the literals in the clause is True .

There are two types of algorithms that we consider – deterministic algorithms which do not involve any randomness and nondeterministic algorithms which are allowed to use randomness.

Definition B.3 *An algorithm is said to be* **polynomial time** *if there is a polynomial* $p(\cdot)$ *and a constant* $c > 0$ *such that the time the algorithm takes to compute an output is* $\leq cp(|x|)$ *for all inputs* x *where* $|x|$ *is the size of the binary description of* x.

Definition B.4 *A decision problem* Π *is* **decidable in deterministic polynomial time** *if there is a deterministic polynomial time algorithm* $A(\cdot)$ *such that for all instances* $x \in D_\Pi$, $A(x) =$ *Yes if and only if* $x \in Y_\Pi$.

Definition B.5 P *is the set of problems that can decided in deterministic polynomial time.*

Since a nondeterministic algorithm involves randomness, the output of a nondeterministic algorithm may vary from one invocation to another even if the input to the algorithm is the same. We say such an algorithm has many possible computations for each input. A nondeterministic algorithm solves a decision problem if it returns Yes with nonzero probability if and only if the input is a Yes instance of the problem. Formally:

Definition B.6 *Let* Π *be a decision problem.* Π *is* **decidable in nondeterministic polynomial time** *if there is a nondeterministic polynomial time algorithm* $A(\cdot)$ *such that for all instances* $x \in D_\Pi$, $A(x) =$ *Yes for some possible computation if and only if* $x \in Y_\Pi$.

Definition B.7 NP *is the set of problems that can be decided in non-deterministic polynomial time.*

Determining whether nondeterminism provides any benefit is the major open problem in computer science.

Open B.8 *Is* P = NP*?*

A problem is NP-hard if every problem in NP can be reduced to it in deterministic polynomial time. Formally:

Definition B.9 *A decision problem* Π *is* NP-hard *if for every problem* Π' *in* NP *there exists a function* $f : D_{\Pi'} \to D_{\Pi}$ *such that*

1. $f(\cdot)$ *can be computed by a deterministic polynomial time algorithm*
2. $f(x) \in Y_{\Pi}$ *if and only if* $x \in Y_{\Pi'}$.

Π *is said to be* NP-complete *if in addition* Π *is in* NP .

If an NP-hard problem can be solved by a deterministic polynomial time algorithm, then, since every problem in NP can be reduced to it in deterministic polynomial time, every problem in NP can be solved in deterministic polynomial time. In other words, P = NP. Since it is not known whether P = NP, we take the fact that a problem is NP-hard to be an indication that it is not possible to solve the problem in a reasonable amount of time for all instances.

Proposition B.10 (Cook, 1971) SAT *is* NP-complete.

Proof. The standard proof is to reduce a generic problem in NP to SAT in polynomial time. ∎

Corollary B.11 SAT *is in* P *if and only if* P = NP.

Having a problem that is NP-complete provides a way to prove other problems to be NP-hard since, to prove that a problem Π is NP-hard, it is enough to find a polynomial time transformation from a known NP-complete problem to Π. This is true since the composition of polynomial time transformations is another polynomial time transformation. For example, it is possible to reduce SAT to the following problem which we study in Part II:

Definition B.12 MAXSAT *is the decision problem defined as follows. Let V be a set of boolean variables, let θ be a CNF formula over those variables, and let k be an integer between 2 and $|\theta|$. Is there a truth assignment to the variables in V that satisfies at least k clauses in θ?*

Proposition B.13 MAXSAT *is* NP-hard.

Proof. Define $f : D_{\mathsf{SAT}} \to D_{\mathsf{MAXSAT}}$ by $f((V,\theta)) = (V,\theta,|\theta|)$. This function is certainly computable in polynomial time. Since an instance (V,θ) of SAT is in Y_{SAT} if and only if $|\theta|$ clauses are satisfied, $f(x) \in Y_{\mathsf{MAXSAT}}$ if and only if $x \in Y_{\mathsf{SAT}}$. ∎

Bibliography

Agnew, J. and R. C. Knapp (1983). *Linear Algebra with Applications*. Brooks/Cole Publishing Co., Monterey, CA.

Aumann, R. (1981). Survey of repeated games. In *Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern*, pp. 11–42. Zurich: Bibliographisches Institut.

Aumann, R. (1997). Rationality and bounded rationality. *Games and Economic Behavior 21*, 2–14.

Aumann, R. and M. Maschler (1995). *Repeated Games with Incomplete Information*. MIT Press, Cambridge, MA.

Battiti, R. (1998). Approximation algorithms and heuristics for MAX-SAT. In *Handbook of Combinatorial Optimization* (D.-Z. Zhu and P. Pardalos Eds.), Volume 1, pp. 77–148. Norwell, MA: Kluwer Academic Publishers.

Ben-Porath, E. (1993). Repeated games with finite automata. *Journal of Economic Theory 59*, 17–32.

Cook, S. (1971). The complexity of theorem proving procedures players. In *Proceedings of the 3rd ACM Symposium on the Theory of Computing*, New York, NY, pp. 151–158.

Cover, T. and J. Thomas (1991). *Elements of Information Theory*. John Wiley and Sons, Inc., New York, NY.

Garey, M. and D. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY.

Gibbard, A. (1973). Manipulation of voting schemes: A general result. *Econometrica 41*, 587–602.

Green, J. and J. Laffont (1979). *Incentives in Public Decision Making*. North-Holland, Amsterdam.

Groves, T. (1973). Incentives in teams. *Econometrica 41*, 617–631.

Hoffman, K. and R. Kunze (1971). *Linear Algebra*. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA.

Jackson, M., T. Palfrey, and S. Srivastava (1994). Undominated Nash implementation in bounded mechanisms. *Games and Economic Behavior 6*, 474–501.

Johnson, D. (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and Systems Sciences 9*, 256–278.

Kalai, E. (1990). Bounded rationality and strategic complexity in repeated games. In *Game Theory and Applications* (T. Ichiishi, A. Neyman, and Y. Tauman Eds.), pp. 131–157. San Diego, CA: Academic Press.

Kalai, E. and J. Ledyard (1998). Repeated implementation. *Journal of Economic Theory 83*, 308–317.

Luce, R. D. and H. Raiffa (1957). *Games and Decisions*. John Wiley and Sons, Inc., New York, NY.

Mas-Colell, A., M. Whinston, and J. Green (1995). *Microeconomic Theory*. Oxford University Press, New York, NY.

Maskin, E. (1985). The theory of implementation in Nash equilibrium: a survey. In *Social goals and social organization: Essays in memory of Elisha Pazner* (L. Hurwicz, D. Schmeidler, and H. Sonnenschein Eds.), pp. 173–204. New York, NY: Cambridge University Press.

McKelvey, R. (1989). Game forms for Nash implementation of general social choice correspondences. *Social Choice and Welfare 6*, 139–156.

Moore, E. (1956). Gedanken experiments on sequential machines. *Annals of Mathematics Studies 34*, 129–153.

Moore, J. (1992). Implementation, contracts and renegotiation in environments with complete information. In *Advances in economic theory: Sixth World Congress* (J.-J. Laffont Ed.), pp. 182–282. Cambridge, UK: Cambridge University Press.

Moore, J. and R. Repullo (1988). Subgame perfect implementation. *Econometrica 56*, 1191–1220.

Myerson, R. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, MA.

Neyman, A. (1997). Cooperation, repetition and automata. In *Cooperation: Game Theoretic Approaches* (S. Hart and A. Mas-Colell Eds.), Volume NATO ASI–Series F, Vol. 155, pp. 233–255. New York, NY: Springer Verlag.

Neyman, A. and D. Okada (1999). Strategic entropy and complexity in repeated games. *Games and Economic Behavior 29*, 191–223.

Neyman, A. and D. Okada (2000). Repeated games with bounded entropy. *Games and Economic Behavior 30*(2), 228–247.

Nisan, N. and A. Ronen (1999). Algorithmic mechanism design. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, Atlanta, GA, pp. 129–140.

Osborne, M. and A. Rubinstein (1990). *Bargaining and Markets*. Academic Press, San Diego, CA.

Owen, G. (1995). *Game Theory* (Third ed.). Academic Press, San Diego, CA.

Palfrey, T. and S. Srivastava (1991). Nash implementation using undominated strategies. *Econometrica 59*, 479–501.

Papadimitriou, C. (1992). On games with a bounded number of states. *Games and Economic Behavior 4*, 122–131.

Papadimitriou, C. and K. Stieglitz (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, New York, NY.

Repullo, R. (1987). A simple proof of Maskin's theorem on Nash implementation. *Social Choice and Welfare 4*, 39–41.

Ronen, A. (1999, September). A note of strategy-proof approximation mechanisms. Working Paper, Institute of Computer Science, Hebrew University of Jerusalem.

Rosenschein, J. and G. Zlotkin (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, Cambridge, MA.

Rubinstein, A. (1998). *Modeling Bounded Rationality*. MIT Press, Cambridge, MA.

Saijo, T. (1988). Strategy space reductions in Maskin's theorem: Sufficient conditions for Nash implementation. *Econometrica 56*(3), 693–700.

Sandholm, T. (1999). Distributed rational decision making. In *Mutiagent Systems: A Modern Approach to Distributed Artificial Intelligence* (G. Weiss Ed.), pp. 201–258. Cambridge, MA: MIT Press.

Satterthwaite, M. (1975). Strategy-proofness and Arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory 10*, 187–217.

Shapley, L. S. and R. N. Snow (1950). Basic solutions of discrete games. In *Contributions to the Theory of Games* (Annals of Mathematics studies, 24) (H. W. Kuhn and A. W. Tucker Eds.), pp. 27–35. Princeton, NJ: Princeton University Press.

Simon, H. A. (1982). *Models of Bounded Rationality*, Volume 2. MIT Press, Cambridge, MA.

Stearns, R. E. (1967). A formal information concept for games with incomplete information. In *Report of the U.S. Arms Control and Disarmament Agency ST-116*, pp. 405–433. Washington, D.C.

Stearns, R. E. (1989). Memory bounded game playing automata. Technical Report No. 547, Institute for Mathematical Studies in the Social Sciences, Stanford

University.

von Neumann, J. (1928). Zur theorie der gesellschftsspiele. *Mathematische Annalen 100*, 295–320.