ABSTRACT

Title of dissertation: SYNTHESIS OF STRATEGIES FOR NON-ZERO-SUM REPEATED GAMES

Tsz-Chiu Au, Doctor of Philosophy, 2008

Dissertation directed by: Professor Dana Nau
Department of Computer Science

There are numerous applications that involve two or more *self-interested* autonomous agents that repeatedly interact with each other in order to achieve a goal or maximize their utilities. This dissertation focuses on the problem of how to identify and exploit useful structures in agents' behavior for the construction of good strategies for agents in multi-agent environments, particularly non-zero-sum repeated games.

This dissertation makes four contributions to the study of this problem. First, this thesis describes a way to take a set of interaction traces produced by different pairs of players in a two-player repeated game, and then find the best way to combine them into a strategy. The strategy can then be incorporated into an existing agent, as an enhancement of the agent's original strategy. In cross-validated experiments involving 126 agents for the Iterated Prisoner's Dilemma, Iterated Chicken Game, and Iterated Battle of the Sexes, my technique was able to make improvement to the performance of nearly all of the agents. Second, this thesis investigates the issue of uncertainty about goals when a goal-based agent situated in a nondeterministic environment. The results of this investigation include the necessary and sufficiency conditions for such guarantee, and an algorithm

for synthesizing a strategy from interaction traces that maximizes the probability of success of an agent even when no strategy can assure the success of the agent. Third, this thesis introduces a technique, Symbolic Noise Detection (SND), for detecting noise (i.e., mistakes or miscommunications) among agents in repeated games. The idea is that if we can build a model of the other agent's behavior, we can use this model to detect and correct actions that have been affected by noise. In the 20th Anniversary Iterated Prisoner's Dilemma competition, the SND agent placed third in the "noise" category, and was the best performer among programs that had no "slave" programs feeding points to them. Fourth, the thesis presents a generalization of SND that can be wrapped around *any* existing strategy. Finally, the thesis includes a general framework for synthesizing strategies from experience for repeated games in both noisy and noisy-free environments.

SYNTHESIS OF STRATEGIES FOR NON-ZERO-SUM REPEATED GAMES

by

Tsz-Chiu Au

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Advisory Committee:
Professor Dana Nau, Chair/Advisor
Professor Atif Memon
Professor James Reggia
Professor P.S. Krishnaprasad
Professor Sarit Kraus
Professor V.S. Subrahmanian

Acknowledgments

I would like to thank my advisor, my colleagues, and my friends for their continuous support and encouragement during my years at UMD.

First and foremost, I have to thank my advisor, Prof. Dana Nau for his mentorship. It is my fortune to have Dana to be my advisor in my life. I really appreciate his teachings about how to be a more successful researcher. Apart from being my mentor, Dana is also my good friend. The hallmark of Dana is his humor, which is a source of encouragement and amusement to me and many others. His generous support had given me the peace of mind that was much needed in my graduate study.

I learned tremendously from my colleagues and my teachers. In particular, I would like to thank Sarit Kraus, Héctor Muñoz-Avila, and V.S. Subrahmanian. Special thanks go to past and current members of the DSN research group, who shared their knowledge and great passions for AI research with me throughout my research career at UMD.

People in the business office in the CS department, UMIACS and ISR are extremely helpful and friendly. I want to express my gratitude to Fatima in the graduate office. Without her friendly "nagging", I am afraid I would have taken a much long time to finish my degree. In addition, I am grateful to Felicia; it was always joyful to talk to her in hallways. I would like to thank all system staffs for their technical supports.

My friends played a large part in my graduate study. Among many others, I owe my gratuities to Waiyian Chong, Nick Frangiadakis, Annie Hui, and Shu-Hoi Pun for their friendships and the sharing of joys. Their friendships are integrated parts of my life. Most of all, I would like to thank my parents, my brother and my sister for their encouragement

and patience.

University of Maryland, College Park is a superb place for study and research. How joyful writing a paper would be in McKeldin Library with lots of hardworking students overnight? I will definitely miss this beautiful campus and my couch in my office very much in the future.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are numerous applications that involves two or more *self-interested* autonomous agents that repeatedly interact with each other in order to complete a task or maximize their utilities. Some notable applications are computer game playing, negotiation of contracts, and foreign policy making. This dissertation focuses on the problem of finding a good strategy for an agent in multi-agent environments, particularly non-zero-sum repeated games, so as to maximize the utility of the agent.

Game theory has been widely used for studying interactions among self-interested agents whose performance depends on how well they interact with each other. Classical game theory studies the game-theoretic notion of an equilibrium strategy and various solution concepts, based on the assumptions that agents are rational, actions are perfectly executed, and observations have no error. While techniques and analysis in classical game theory enjoyed great success in many interesting and important problems, their success is limited when they are applied directly to more complicated settings in which the assumptions of rationality and perfect acting and sensing are severely derived. For example, the empirical studies of repeated games have shown that agents seldom behaves according to the subgame perfect equilibria of a game, casting doubt on the assumption of rationality based on backward induction in repeated or extensive games. Thus, a key problem in today's game theory research is the disparity between the theoretical predictions of

agents' behavior and the actual behavior of agents in tournaments or real-world situations. This disparity prompts us to rethink about the underlying assumptions of classical game theory. This dissertation is set out to propose new techniques for decision making in multi-agent environments when agents are irrational, mistakes can occur during the execution of actions, and observations are imperfect.

To account for the inconsistency between game theory and empirical results, researchers have proposed alternative models of economical reasoning called the bounded rationality [58, 59, 56], removed the assumptions that agents have infinite computational resources for decision making. However, most of these models focus on the computational limitation of the rationality, and do not take contextual information of a game into account. Experiments conducted by Thomas C. Schelling in 1950s showed that agents often depend on cultural or contextual information in their decision making, and this information is largely ignored in mathematical game theory. These results suggest that mathematical analysis alone may not be sufficient to account for the behavior of agents in a game. One way to get a more precise description of the agent's behavior is based on the observations of the behavior of the agents. This *empirical approach* assumes that we have data about decisions made by the agents in a game in a given context or environment. Then we can infer the behavior of the agents from the collected data. This is the approach this dissertation pursuits.

We shall use the empirical approach to the studies of several non-zero-sum repeated games including the Iterated Prisoner's Dilemma (IPD), the Iterated Chicken Game (ICG), and the Iterated Battles of the Sexes (IBS). In particular, we aim to address two problems in these games: errors in the interaction among agents and the unfamiliar-

ity with other agents' irrational behavior. The first problem is about mistakes can occur during the execution of actions and imperfect observations of agents—if errors can occur during the interaction among agents, the agents may no longer be able to maintain cooperation with each other and the performance of the whole system decreases. The second problem is about how to deal with irrational agents—agents that do not act according to the Nash equilibrium or subgame perfect equilibrium. The techniques we proposed in this dissertation performed pretty well in these games, according to our experiments. Our hope is that these techniques can be extended for other kind of games or multi-agent environments as well.

1.1 Objective and Approach

Broadly speaking, our research question is

How to identify and exploit useful structures in agents' behavior for effective decision making in non-zero-sum repeated games?

Repeated games are traditionally regarded as the "fruit flies" in the studies of multi-agent systems and game theory. Despite the simplicity of the game structures and rules, repeated games can still be considered as a complete multi-agent environment in which agents have to repeatedly interact with each other in order to maximize their own utilities. In fact, repeated games are so rich in the variety of strategies and solutions, and yet existing theoretical analysis of repeated games seems to unable to make perfect predictions of the actual behavior of players in those games, making repeated games an interesting subject. Therefore, we should limit the scope of our studies to repeated games. At various

points in this dissertation we shall discuss the possibility of extending the techniques we developed for these games to other problems.

Among all repeated games we are interested in non-zero-sum repeated games in which players does not always compete with each other. We choose to study non-zero-sum games because many real-world situations are not strictly competitive, and there are rooms for agents to cooperate with each other. To model these situations, non-zero-sum games are more appropriate than zero-sum games. Furthermore, the behavior of agents in non-zero-sum games is so different from the behavior of the agents in zero-sum games such as Roshambo and chess, causing a huge difference in the design of strategies for zero-sum games and non-zero-sum games. Due to these differences, non-zero-sum games are technically interesting to AI researchers.

Our chief concern is decision making in non-zero-sum repeated games. More precisely, we concern with how to generate a good strategy for an agent to interact with other agents. Our approach is to identify inherent structures in agent's behavior and then exploit these structures for the creation of better strategies or the improvement of existing strategies. As opposed to analytical approaches which tackles the problem from the first principles (e.g., alpha-beta pruning for game-tree search and value iteration for MDPs), our approach is based on empirical observations of structures of agents' behavior. What we found is that it is often more easy to discern the structure in the behavior of the agents in non-zero-sum games than in zero-sum games—in zero-sum games players tend to hide their intention by making their moves unpredictable, but in non-zero-sum games agents are more eager to openly express their intention in their moves. Due to the availability of easily discerned structures in agents' behavior, we believe our approach can be more

effective than analytical approaches for non-zero-sum repeated games.

The difficult question, of course, is how to identify the structures in agents' behavior and then effectively exploit them. We consider two directions to address this question: (1) manually identify any recurrent feature in the agents' behavior from records of interactions in previous games, and (2) automatically discover those features using an algorithm. Either ways, the focus of the identification is to discover structures in agent's behavior that can be used for improving the performance of existing agents or creating new agents. In this dissertation, we will show (1) how to exploit the clarity of behavior we observed to deal with noise, and (2) how to automatically identify a set $\mathcal{T}$ of interactions from a database of interaction traces, such that $\mathcal{T}$ can be used to form a new strategy and enhance existing strategies.

1.2 Contributions

This dissertation revolves around the following two theses:

Thesis 1: Clarity of agents' behavior can be used for detecting errors in the interaction among agents.

Thesis 2: Records of Interactions among agents can be used to improve the performance of existing agents, especially when the distribution of agents' behavior is highly skewed.

Our investigation of these theses leads to several technical contributions to the studies of repeated games and multi-agent systems. The following are the key contributions.

1. **Synthesis of Strategies from Interaction Traces**

   We devised a novel technique for combining records of interaction, produced by many different agents in a two-player repeated game, into a strategy which is called a *composite* strategy. Our algorithm, called the CIT algorithm, can synthesize the best composite strategy from a database of interaction traces in polynomial time. Our experimental results show that given a collection of agents, our technique can produce composite strategies that does better than all of the given agents, except the best agents, when combining the composite strategies with existing strategies.

2. **Solvability of Task-Completion Problems with Goal Uncertainty**

   We introduce the notion of strong solvability and weak solvability for problems in which a goal-based agent must interact with a nondeterministic environment in order to achieve a goal, but the agent is uncertain about the goals. We state the necessary and sufficient conditions of strong solvability in terms of interaction traces by that an agent can successfully reach the goals, provide an algorithm to find an optimal solution given the successful interaction traces for weakly solvable problems, and evaluate the algorithm empirically when the size of its inputs is small.

3. **Symbolic Noise Detection**

   Accidents can cause great difficulty in cooperation with others. For example, in the Iterated Prisoner's Dilemma (IPD) with noise, actions chosen by the players can be randomly changed by noise, and that can trigger a long sequence of mutual defections between the players. Previous approaches for dealing with noise in the IPD are based on forgiveness. Our approach, however, is based on the detection of

noise. We developed a technique called symbolic noise detection (SND) for noise detection in the IPD with noise. The performance of SND is demonstrated in the 2005 IPD tournament.

4. **Analysis of Symbolic Noise Filter**

   A symbolic noise filter (SNF) is a wrapper that can be placed around any existing strategy in repeated games in order to endow the strategy with the capability of noise detection and correction. We evaluated the performance of SNF in two repeated games, the Iterated Chicken game and the Iterated Battle of the Sexes, and found that SNF is highly effective in these games. Our experimental analysis indicated that SND will be more effective in any games in which strategies often show a stable behavior.

5. **A Framework for Experience-Based Strategy Construction for Repeated Games**

   To construct a better strategy for a repeated game, one may want to examine the historical records of previous games, in order to learn from the data. We consider a learning process that involves the construction of several key data structures from data, the transformation from one data structure to another, and eventually the construction of strategies from the data structures. We proposed a framework that describes this learning process whose data structures are opponent models, action-value functions, and strategies.

Apart from the above technical contributions, this dissertation also includes two scientific discoveries pertaining to the behavior of agents in the IPD, ICG, and IBS, and

perhaps non-zero-sum repeated games in general. First, we found that agents in those games often display deterministic behavior during interaction, and therefore noise are often detectable in those games. Second, we found that the distributions of interaction traces are highly skewed in these games, and therefore only a small number of interaction traces is needed in order to form a partial strategy that can be used to interact successfully with the agents in the those games. We will discuss about these empirical properties of agents' behavior at the end of this dissertation.

1.3 Outline of this Thesis

The contents of the rest of this dissertation are as follows:

Chapter 2. Synthesis of Strategies from Interaction Traces

This chapter presents a technique called *the CIT technique*, for the synthesis of strategies for repeated games. Starting from a set of interaction traces produced by different pairs of players in a two-player repeated game, the *CIT algorithm* selects a subset of interaction traces and combines them into a *composite strategy*. The CIT algorithm is a polynomial-time algorithm that can generate the best such composite strategy, which potentially outperforms most of the players who contributed the interaction traces. This chapter also describes how to incorporate the composite strategy into an existing agent, as an enhancement of the agent's original strategy.

Chapter 3. Task-Completion Problems with Goal Uncertainty

This chapter provides a way to adapt the CIT technique presented in Chapter 2 to *any* problem that involves an agent who must achieve a goal or complete a task

by interacting an environment that responds to the agent's actions nondeterministically. First, we give the necessary and sufficient conditions under which an agent can guarantee to accomplish the goal. For problems that no agent can guarantee to successfully accomplish the goal, we provide an algorithm that takes a set of interaction traces and generates a composite strategy with the highest probability of success.

Chapter 4. Noise Detection in the Iterated Prisoner's Dilemma

This chapter proposes the noise detection approach to cope with noise in the Noisy Iterated Prisoner's Dilemma, a version of the IPD in which actions or observations can be randomly changed by accidental events. We describe the philosophy of symbolic noise detection (SND) and the details of the DBS strategies that uses SND for noise detection in the IPD. Then we present the performance of the DBS strategies in the 2005 IPD tournament.

Chapter 5. Symbolic Noise Filter

This chapter introduces a wrapper that can be placed around any agent in 2×2 repeated games so as to filter the noise that may present in the observed actions generated by the other player. The wrapper is called the symbolic noise filter (SNF) and is a simplified version of the SND mechanism in DBS. We conducted experiments to evaluate SNF, and empirically explained the performance of SNF in the Iterated Chicken Game and the Iterated Battles of the Sexes.

Chapter 6. A Framework for Building Strategies for Repeated Games

This chapter describes a general framework for constructing strategies from data in repeated games. This framework revolves around three key data structures for opponent modeling and decision making in repeated games: opponent models, action-value functions, and strategies. This framework elucidates the relationships between these data structures, and suggests a number of ways by which data can be turned into a strategy.

Chapter 7. Conclusions and Future Work

The last chapter reviews the contributions of this dissertation and proposes directions for future work.

In addition, Appendix A contains a list of acronyms that appears in this dissertation. The list also contains the names of several well-known strategies for the IPD. In Appendix B, there are tables describing mathematical notations that we define and use in this dissertation.

Chapter 2

Synthesis of Strategies from Interaction Traces

In this section, we describe how to take a set of interaction traces produced by different pairs of players in a two-player repeated game, and combine them into a *composite strategy*. We provide an algorithm that, in polynomial time, can generate the best such composite strategy. We describe how to incorporate the composite strategy into an existing agent, as an enhancement of the agent's original strategy.

We provide experimental results using interaction traces from 126 agents (most of them written by students as class projects) for the Iterated Prisoner's Dilemma, Iterated Chicken Game, and Iterated Battle of the Sexes. We compared each agent with the enhanced version of that agent produced by our algorithm. The enhancements improved the agents' scores by about 5% in the IPD, 11% in the ICG, and 26% in the IBS, and improved their rank by about 12% in the IPD, 38% in the ICG, and 33% in the IBS.

2.1 Introduction

To create new and better agents in multi-agent environments, we may want to examine the strategies of several existing agents, in order to combine their best skills. One problem is that in general, we won't know what those strategies are. Instead, we'll only have observations of the agents' interactions with other agents. The question is how to synthesize, from these observations, a new strategy that performs as well as possible.

In this paper we present techniques for taking *interaction traces* (i.e., records of observed interactions) from many different pairs of agents in a 2-player iterated game, and synthesizing from these traces a new strategy called a *composite strategy* (see Figure 2.1). We also show how an existing agent can enhance its performance by combining its own strategy with the composite strategy. Our contributions include the following:

- We give a formal definition of a composite strategy, and present necessary and sufficient conditions under which a set of interaction traces from different agents can be combined together to form a composite strategy.

- We provide a polynomial-time algorithm for synthesizing, from a given set of interaction traces $\mathcal{T}$, a composite strategy that is *optimal*, in the sense that it performs at least as well as any other composite strategy that can be formed from $\mathcal{T}$.

- We provide a way to enhance an existing agent's performance by augmenting its strategy with a composite strategy.

- We provide experimental results demonstrating our algorithm's performance in the Iterated Prisoner's Dilemma (IPD), Iterated Chicken Game (ICG), and Iterated Battle of the Sexes (IBS), using interaction traces from 126 agents (117 written by students as class projects, and 9 standard agents from the published literature). For each agent, we compared its performance with the performance of our enhanced version of that agent. On the average, the percentage improvements in score were about 5% in the IPD, 10% in the ICG, and 25% in the IBS; and the percentage improvements in rank were about 11% in the IPD, 38% in the ICG, and 33% in the IBS.

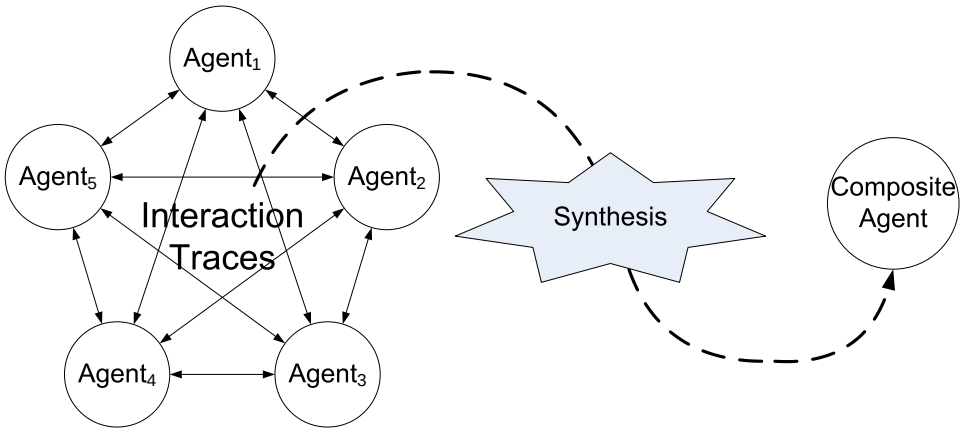

Figure 2.1: A composite strategy is synthesized from records of the interactions among many existing agents. An agent using this strategy can potentially outperform most of the agents from whom the interaction traces were obtained.

Our results show that given a collection agents, it is possible to do better than most of these agents, by combining some of the "best" behaviors (in terms of interaction traces) exhibited by these agents in the past, as illustrated in Figure 2.1.

2.2 Basic Definitions

Consider a two-player finite repeated game, such as the IPD. At any time point t, two agents λ_A and λ_B can choose actions a and b from finite sets of actions A and B, respectively. Neither agent learns what the other's action is until after choosing its own action. Throughout this paper, we will look at games from λ_A's point of view; i.e., λ_A is "our agent" and λ_B is the opponent.

We call the pair (a, b) an *interaction* between λ_A and λ_B. An *interaction trace* between λ_A and λ_B is a sequence of interactions $\tau = \langle (a_1, b_1), (a_2, b_2), \ldots (a_n, b_n) \rangle$, where

a_i and b_i are the actions of λ_A and λ_B, respectively.[1] The length of τ is $|\tau| = n$.

We assume interactions occur at a finite number of discrete time points t_1, t_2, ..., t_N, where N is the total number of interactions.[2] A *history* (or a *history* up to time t_k) is an interaction trace $\tau = \langle (a_1, b_1), (a_2, b_2), \ldots (a_k, b_k) \rangle$, where (1) a_i and b_i are the actions of λ_A and λ_B at time t_i, respectively, and (2) $0 \leq k \leq N$. τ is a *full* history if $|\tau| = N$. The histories of λ_A's and λ_B's actions are $\tau^A = \langle a_1, a_2, \ldots, a_n \rangle$ and $\tau^B = \langle b_1, b_2, \ldots, b_n \rangle$, respectively.

We often deal with the prefix of a history in this chapter. Let τ be a history $\langle (a_1, b_1), (a_2, b_2), \ldots (a_k, b_k) \rangle$. The *j'th prefix* of τ is the subsequence $\mathsf{prefix}_j(\tau) = \langle (a_1, b_1), (a_2, b_2), \ldots (a_j, b_j) \rangle$, whereas the *j'th suffix* of τ is $\mathsf{suffix}_j(\tau) = \langle (a_{j+1}, b_{j+1}), (a_{j+2}, b_{j+2}), \ldots (a_k, b_k) \rangle$.

An agent uses a *strategy* to determine what actions it should take in every time point. There are two types of strategies: *pure strategies* and *mixed strategies*. Let $\mathcal{H}$ be the set of all possible histories. A pure strategy is a function ϕ from histories $\mathcal{H}$ into actions A, that returns the next action to perform. A mixed strategy is a pair $\psi = (\Phi, \Delta_\Phi)$, where Φ is a nonempty set of pure strategies and Δ_Φ is a probability distribution over Φ. Each agent λ has exactly one strategy, which can be either pure or mixed. We denote the strategy of an agent λ by $\mathsf{strategy}(\lambda)$. A *deterministic* agent is one whose strategy is a pure strategy, and a *probabilistic* agent is one whose strategy is a mixed strategy.

[1] This formalism can easily model environments where the actions are interleaved rather than occurring simultaneously, by specifying that at odd time points λ_A's action must always be a "null" action, and similarly for λ_B at even time points.

[2] It is easy to modify our definitions and algorithms to handle games in which the number of interactions can vary. But to simply our discussion, we'll assume the number of interactions is constant.

An alternative definition of a mixed strategy is a mapping ψ' from histories $\mathcal{H}$ to the set $\mathbb{P}_A$ of all probability distributions over the set A of actions. Thus, $\psi'(\tau)$ returns a probability distribution Δ'_A such that $\Delta'_A(a)$ is the probability of choosing $a \in A$ at the situation $\mathsf{s}(\tau)$. For clarity, we define $\psi'(\tau, a) = \Delta'_A(a)$.

It is not hard to see that the two definitions of mixed strategies are mathematically equivalent. [3] Given $\psi = (\Phi, \Delta_\Phi)$, we construct ψ' as follows: for each $\tau \in \mathcal{T}$, let $\Phi' \subseteq \Phi$ be the set of all pure strategies that can possibly generate τ (i.e., $\Phi' = \{\tau : \forall k, \phi', (\tau = \mathsf{prefix}_k(\tau')) \wedge (\tau' \in \mathcal{H}(\phi', \cdot)) \wedge (k \geq 0) \wedge (\phi' \in \Phi)\}$. Let Φ'_a be the set of pure strategies in Φ' that will output $a \in A$ after generating τ (i.e., $\Phi'_a = \{\phi : (\phi(\tau^B) = a) \wedge (\phi \in \Phi')\}$). Then, $\Delta'_A(a) = \frac{\sum_{\phi \in \Phi'_a} \Delta_\Phi(\phi)}{\sum_{\phi \in \Phi'} \Delta_\Phi(\phi)}$. Conversely, given ψ', we can construct $\psi = (\Phi, \Delta_\Phi)$ where Φ is the set of all possible pure strategies and $\Delta_\Phi(\phi) = \prod_{\tau \in \mathcal{H}, a \in A} \{\delta(\tau, a)\}$, where $\delta(\tau, a) = \Delta'_A(a)$ if τ can be generated by $\phi \in \Phi$ and $\delta(\tau, a) = 0$ otherwise. However, these conversions may not be computationally feasible because it could take too much computational resources.

Suppose both our agent λ_A and the opponent λ_B are deterministic, and their pure strategies are $\phi_A = \mathsf{strategy}(\lambda_A)$ and $\phi_B = \mathsf{strategy}(\lambda_B)$, respectively. When the deterministic agents λ_A and λ_B interact, only one possible interaction trace can be generated, namely the interaction trace $\mathsf{trace}(\phi_A, \phi_B) = \tau_n$, where τ_i is defined recursively as follows: $\tau_0 = \langle\rangle$, and $\tau_{j+1} = \tau_j \circ \langle(\phi_A(\tau_j), \phi_B(\tau_j))\rangle$ for $i = 1, \ldots, n$. We assume λ_A's performance is measured by a *utility function* u_A, which is a function from interaction traces to non-negative real numbers. Thus, the reward for λ_A when its opponent is λ_B is

[3] More precisely, for each ψ there is a unique ψ' that is equivalent to ψ, but for each $mixed'$ there are one or more ψ that are equivalent to ψ'

$u_A(\mathsf{trace}(\phi_A, \phi_B))$.

Consider an agent λ that uses a mixed strategy $\psi = (\Phi, \Delta)$. We will call the pure strategies in Φ the *possible* strategies for λ. In each game, λ will choose one strategy $\phi \in \Phi$ according to the probability distribution Δ, and its opponent will not know which strategy was chosen. We call the chosen strategy λ's *actual* strategy in this game.

Suppose the opponent λ_B's strategy is a mixed strategy $(\Phi_B, \Delta_{\Phi_B})$. The overall performance of our agent λ_A using a pure strategy ϕ_A in its interactions with λ_B is λ_A's *expected utility*, which is

$$EU_A(\lambda_A, \lambda_B) = \sum_{\phi_B \in \Phi_B} \left\{ \Delta_{\Phi_B}(\phi_B) \times u_A(\mathsf{trace}(\phi_A, \phi_B)) \right\}.$$

2.3 Synthesis of Strategies from Interaction Traces

Section 2.3.1 presents an algorithm that can reconstruct the strategy of a single agent λ_A, given a collection of interaction traces generated by λ_A. Section 2.3.2 shows how to take a collection of traces that were generated by more than one agent, and construct a new strategy called a *composite* strategy, that combines parts of the strategies of all of those agents. Section 2.3.3 gives an algorithm to find the optimal one of these composite strategies.

2.3.1 Reconstructing an Agent's Strategy

Suppose we play a pure strategy ϕ against a mixed strategy $\psi = (\Phi, \Delta)$, where $\Phi = \{\phi_1, \phi_2, \phi_3\}$. Then the set of all possible interaction traces is

$$\mathcal{T} = \{\mathsf{trace}(\phi, \phi_j) : j \in \{1, 2, 3\}\}.$$

$\mathcal{T}$ contains enough information for us to construct a new strategy $\phi_{\mathcal{T}}$ whose interaction traces with ψ will be exactly the same as ϕ's. Figure 2.2 gives the pseudocode for an agent CA($\mathcal{T}$) that can generate and play the strategy $\lambda_{\mathcal{T}}$. We will call CA($\mathcal{T}$) a *composite agent*, and $\phi_{\mathcal{T}} = \mathsf{strategy}(\text{CA}(\mathcal{T}))$ a *composite strategy*. The strategy $\phi_{\mathcal{T}}$ is *partial*, i.e., it is only defined over some of the possible histories. However, as we will see in Theorem 1, $\phi_{\mathcal{T}}$ is ψ-*total*, i.e., it is defined over the set of all histories that are possible when playing it against ψ. In other words, in any history τ of interactions between CA($\mathcal{T}$) and the agent using ψ, CA($\mathcal{T}$) will always be able to determine what its next action $a_{|\tau|+1}$ should be.

To see how the CA agent works, suppose that the three interaction traces in $\mathcal{T}$ are

$$\begin{aligned}
\tau_1 &= \langle (C,C),(C,D),(D,C)\rangle, \\
\tau_2 &= \langle (C,C),(C,C),(C,C)\rangle, \\
\tau_3 &= \langle (C,D),(C,D),(D,C)\rangle.
\end{aligned}$$

Next, suppose we play CA($\mathcal{T}$) against an agent λ' using ψ. Since λ_A's first action is C in all three traces, we have $A_i = \{C\}$ at Line 5 of CA, so CA($\mathcal{T}$) returns $a_1 = C$ as its first action. Suppose λ''s first action is C. Then λ' cannot be using the strategy that produced τ_3, so Line 11 of the agent removes τ_3 from $\mathcal{T}_1'$, leaving τ_1 and τ_2 in $\mathcal{T}_2$. Hence in the next interaction with λ', CA($\mathcal{T}$) will choose $a_2 = C$. Suppose λ''s second action is $b_2 = D$. Then τ_2 is removed from $\mathcal{T}_2'$, and CA($\mathcal{T}$)'s next action is D. Hence that the composite agent CA($\mathcal{T}$) ended up "replaying" the interaction trace τ_1.

The following theorem provides a condition under which CA($\mathcal{T}$) will always generate the same action that λ_A would generate against an opponent λ_B using a mixed strategy ψ:

Agent CA($\mathcal{T}$) /* *a composite agent synthesized from* $\mathcal{T}$ */

1. $i := 1$; $\mathcal{T}_i := \mathcal{T}$
2. Loop until the end of the game
3. If $\mathcal{T}_i = \emptyset$, then exit with failure because $\mathcal{T}$ is insufficient
4. If $\mathcal{T}_i \neq \emptyset$, then
5. $A_i := \{a : (\exists\, \tau \in \mathcal{T}_i)\ a$ is the i'th action of $\tau^A\}$
6. If $|A_i| \neq 1$, then exit with failure because $\mathcal{T}$ is incompatible
7. If $|A_i| = 1$, then let a_i be the action in A_i
8. Output a_i and receive the other agent's action b_i
9. $\mathcal{T}_i' := \mathcal{T}_i$
10. For each $\tau \in \mathcal{T}_i$,
11. If the i'th interaction in τ isn't (a_i, b_i), remove τ from $\mathcal{T}_i'$
12. $\mathcal{T}_{i+1} := \mathcal{T}_i'$; $i := i + 1$

Figure 2.2: The pseudocode of a composite strategy.

Theorem 1 *Let ϕ_A and $\psi = (\Phi_B, \Delta_{\Phi_B})$ be pure and mixed strategies, respectively. If $\mathcal{T} = \{\mathsf{trace}(\phi_A, \phi_B) : \phi_B \in \Phi_B\}$, then $\mathsf{trace}(CA(\mathcal{T}), \phi_B) = \mathsf{trace}(\phi_A, \phi_B)$ for every $\phi_B \in \Phi_B$.*

Proof. Without loss of generality, let $\phi_B \in \Phi_B$ be the actual strategy of λ_B, and let $\tau = \mathsf{trace}(\phi_A, \phi_B)$ be the interaction trace between ϕ_A and ϕ_B. Suppose there exists i such that (a_i, b_i) at Line 8 of Figure 2.2 is not the i'th interaction (a'_i, b'_i) in τ, but, for $1 \leq j < i$, (a_j, b_j) is the j'th interaction in τ. First, all interaction traces in $\mathcal{T}_i$ have the same prefix up to the $(i-1)$'th iteration (any traces without this prefix were removed at Line 11 on a previous iteration) and λ_A is a deterministic agent. Therefore all traces in $\mathcal{T}_i$ (including τ) have the same i'th action a'_i, and the composite agent will certainly output a'_i at the iteration i. Thus, $a_i = a'_i$. Second, ϕ_B will certainly output b'_i at the i'th interaction, given the action sequence $a_1, a_2, \ldots, a_{i-1}$ as its inputs; that is, $b_i = b'_i$. Hence, a contradiction occurs. Therefore (a_i, b_i) at Line 8 of Figure 2.2 is always the i'th interaction in τ, and $\mathsf{trace}(\mathrm{CA}(\mathcal{T}), \phi_B) = \tau$. □

The notion of composite agents establishes the fact that the process of interaction trace generation is reversible: not only can agents generate interaction traces, but in addition, interaction traces can be used to construct agent strategies. This fact opens up opportunities to create better strategies. For example, we can first record the interaction traces generated by an agent, modify the set of interaction traces, and then synthesize a new strategy. By carefully adding or replacing interaction traces in the set of recorded interaction traces, the agent using the new strategy can outperform the original agent. In this chapter, we propose one such modification scheme for producing better agents. The

key idea is that by mixing interaction traces of *two or more* agents together, the composite strategy synthesized from these interaction traces often outperforms every agent contributed the interaction traces. The reason why this method works is simple: when different agents are good at dealing with different possible strategies of the opponent, we can combine the good strategies together so that the composite strategy can be more successful in dealing with the same opponent or similar opponents.

2.3.2 Constructing New Strategies

The previous section showed how to reconstruct a strategy of a single agent λ_A against an agent λ_B, given λ_A's interaction traces with λ_B. The same approach can be used to construct *new* strategies from a collection of traces generated by more than one agent, provided that two conditions, called *compatibility* and *sufficiency*, are satisfied.

To illustrate the notion of compatibility, consider two well-known strategies for the IPD: Tit-For-Tat (TFT) and Tit-For-Two-Tats (TFTT). In the usual setting, the optimal strategy against TFT is to cooperate in every iteration, and the optimal strategy against TFTT is to defect and cooperate alternatively. For an IPD game of five iterations, these can be represented by the following two interaction traces:

$$\begin{aligned} \tau_1 &= \langle (C,C),(C,C),(C,C),(C,C),(C,C)\rangle, \\ \tau_2 &= \langle (D,C),(C,C),(D,C),(C,C),(D,C)\rangle. \end{aligned}$$

However, no pure strategy can generate both of these interaction traces, because in the first interaction, no agent can choose both Cooperate and Defect simultaneously. Thus, there is no single agent that can act optimally against both TFT and TFTT, and we say

that τ_1 and τ_2 are *incompatible* with each other. If we run CA($\{\tau_1, \tau_2\}$), it will return an error message at Line 6 of Figure 2.2.

Now consider an agent that defects in the first iteration and then cooperates for the rest of a game. When this agent plays against TFT, the interaction trace is $\tau_3 = \langle (D, C), (C, D), (C, C), (C, C), (C, C) \rangle$. Although this strategy is not optimal against TFT, it is *compatible* with τ_2: both τ_2 and τ_3 produce D for a_1 and C for a_2; and the opponent's response at the end of the 2nd interaction gives us enough information to decide which of τ_2 and τ_3 to use thereafter. If the opponent's second action b_2 is C, then we discard τ_3 and continue with τ_2, and if b_2 is D, we discard τ_2 and continue with τ_3. This is exactly what CA($\{\tau_2, \tau_3\}$) does when it plays against a mixed strategy that includes TFT and TFTT.

We now formalize the concepts introduced in the above example, to provide necessary and sufficient condition on $\mathcal{T}$ for the synthesis of a composite strategy.

Definition 1 *The* longest common prefix *of two action sequences* $\alpha = \langle a_1, a_2, \ldots, a_n \rangle$ *and* $\alpha' = \langle a'_1, a'_2, \ldots, a'_n \rangle$ *is the longest action sequence* $\mathsf{lcp}(\alpha, \alpha')$ *that's a prefix of both* α *and* α'.

We now define a condition called *compatibility*, that (as we'll see in Theorem 2) is necessary for a set of interaction traces $\mathcal{T}$ to be used successfully by CA. The interaction traces do not all need to be generated by the same agent.

Definition 2 *Two interaction traces* τ_1 *and* τ_2 *are* compatible *if either (1)* $\tau_1^A = \tau_2^A$*, or (2)* $|\mathsf{lcp}(\tau_1^A, \tau_2^A)| > |\mathsf{lcp}(\tau_1^B, \tau_2^B)|$*. Otherwise,* τ_1 *and* τ_2 *are* incompatible.

Definition 3 *A set $\mathcal{T}$ of interaction traces is* compatible *if and only if there is no incompatible pair of interaction traces in $\mathcal{T}$.*

Even if a set $\mathcal{T}$ of interaction traces is compatible, CA($\mathcal{T}$) will not always be able to use them against the opponent λ_B using a mixed strategy ψ unless there is at least one interaction trace for each of λ_B's possible strategies. The following definition formalizes this notion.

Definition 4 *Let $\psi = (\Phi_B, \Delta_{\Phi_B})$ be a mixed strategy. A set $\mathcal{T}$ of interaction traces is* ψ-sufficient *if and only if for every strategy $\phi_B \in \Phi_B$, $\mathcal{T}$ contains an interaction trace $\tau = \langle (a_1, b_1), \ldots, (a_n, b_n) \rangle$ such that the action sequence $\langle b_1, \ldots, b_n \rangle$ can be generated by λ given $\langle a_1, \ldots, a_n \rangle$ as its inputs.*

The following theorem shows that compatibility and ψ-sufficiency are necessary and sufficient to guarantee that CA($\mathcal{T}$) will always be able to play an entire game against ψ.

Theorem 2 *The composite agent CA($\mathcal{T}$) will never exit with failure when it plays against an opponent whose mixed strategy is $\psi = (\Phi_B, \Delta_{\Phi_B})$, if and only if $\mathcal{T}$ is compatible and ψ-sufficient.*

Proof. The "only if" part is trivial. For the "if" part, suppose $\mathcal{T}$ is compatible and ψ-sufficient. Without loss of generality, let $\phi \in \Phi_B$ be the pure strategy that the opponent chooses to use at the start of the game. By induction on i, we prove that CA($\mathcal{T}$) does not exit with failure at the i'th iteration. More precisely, we prove (1) $|A_i| = 1$ at Line 6, and (2) there exist $\tau \in \mathcal{T}_i$ such that τ can be generated by ϕ (i.e., $\mathcal{T}_i \neq \emptyset$ at Line 3), for $1 \leq i \leq n$.

First, let us consider $i = 1$. For any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}$, the first actions in τ_1^A and τ_2^A must be the same since $\mathcal{T}$ is compatible. Therefore, $|A_1| = 1$. Since $\mathcal{T}$ is ψ-sufficient and Φ_B is nonempty, it follows that there is at least one interaction trace $\tau \in \mathcal{T}$ that can be generated by ϕ. Thus, $\mathcal{T}_1 \neq \emptyset$.

Now consider $i = k + 1$. Suppose $|A_k| = 1$ and there exist $\tau \in \mathcal{T}_k$ that can be generated by ϕ. First, since all traces in $\mathcal{T}_k$ have the same prefix up to the $(k-1)$'th iteration and ϕ is a pure strategy, ϕ will return a unique b_k at Line 8, which is the k'th action of τ (otherwise τ is not generated by ϕ). In addition, by Definition 2, for any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}_k$, the first k actions in τ_1^A and τ_2^A must be the same. Thus, the k'th action of τ^A must be a_k, the action generated at Line 8. Therefore, (a_k, b_k) is the i'th interaction of τ, and τ will not be removed from $\mathcal{T}'_k$ at Line 11. Hence $\tau \in \mathcal{T}_{k+1}$ since $\mathcal{T}_{k+1} = \mathcal{T}'_k$ at Line 12. Second, $|A_{k+1}| \geq 1$ because $|\mathcal{T}_{k+1}| \geq 1$. Third, at the start of each iteration $k + 1$, all interaction traces in $\mathcal{T}_{k+1}$ have the same prefix up to the k'th iteration (any traces without this prefix were removed at Line 11 on a previous iteration). By Definition 2, for any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}_{k+1}$, the first $k + 1$ actions in τ_1^A and τ_2^A must be the same; and consequently $|A_{k+1}| \leq 1$. Therefore, $|A_{k+1}| = 1$. By induction, $|A_i| = 1$ and $\mathcal{T}_i \neq \emptyset$, for $1 \leq i \leq n$. □

2.3.3 Finding the Best Composite Strategy

We now consider the problem of finding a strategy against a mixed strategy. It is not difficult to show by reduction from 3SAT that the problem of finding an *optimal* strategy—a strategy with the highest expected utility against a mixed strategy—is **NP**-

hard.

Theorem 3 *The problem of finding a strategy with an expected utility of at least K against a mixed strategy $\psi = (\Lambda, \Delta)$ is* **NP***-hard.*

Proof. We show that 3SAT is polynomial time reducible to this problem. Let f be a Boolean formula in conjunctive normal form with Boolean variables x_1, x_2, ..., x_n and clauses C_1, C_2, ... C_m. For each C_i, we construct an agent λ_i whose inputs are either True or False, and whose output is always the same action, namely Nil. The strategy $\phi_i = \mathsf{strategy}(\lambda_i)$ is a constant function that always returns Nil. The number of interaction is exactly N; thus any agent interacting with λ_i will terminate after N interactions. The utility of the interaction trace $\tau = \langle (v_j, \mathsf{Nil}) \rangle_{j=1..n}$ is 1 if and only if the assignment of v_j to x_j for $j = 1 \ldots n$ satisfies the clause C_i; otherwise $u_A(\tau)$ is 0.

We argue that f is satisfiable if and only if there exists an agent whose expected utility is at least 1 when it interacts with an opponent using a mixed strategy $\psi = (\Phi_B, \Delta_{\Phi_B})$, where $\Phi_B = \{\phi_1, \phi_2, \ldots \phi_m\}$ and Δ_{Φ_B} is a uniform probability distribution over Φ_B. Clearly, if f is satisfiable by an assignment of v_j to x_j for $j = 1 \ldots n$, we can construct an agent λ_A with strategy ϕ_A that chooses v_j in the j'th interaction. Then λ_A would have an expected utility of 1 because $u_A(\mathsf{trace}(\phi_A, \phi_i)) = 1$ for any $\phi_i \in \Phi_B$. If there exists an agent λ_A whose expected utility is at least 1, then (1) $u_A(\mathsf{trace}(\phi_A, \phi_i)) = 1$ for all $\phi_i \in \Phi_B$ since Δ_{Φ_B} is a uniform probability distribution, and (2) λ_A's sequence of action is the same when interacts with any two strategies $\phi_i, \phi_j \in \Phi_B$, since both ϕ_i and ϕ_j always return Nil, and λ_A, as a deterministic agent, cannot return two different actions at the same interaction for ϕ_i and ϕ_j. Thus, the sequence of actions generated by λ_A would

satisfy every clause in f. □

Therefore, instead of finding the optimal strategy, we find the best composite strategy that can be synthesized from a given set of interaction traces, and then experimentally show that the best composite strategy can perform pretty well in practice.

Let $\psi = (\Phi_B, \Delta_{\Phi_B})$ be a mixed strategy used by the opponent λ_B. Suppose we are given a set $\mathcal{T}_B$ of interaction traces that were played against λ_B; and suppose that for each interaction trace $\tau \in \mathcal{T}_B$, we know the utility $u_A(\tau)$ for the agent that played against λ_B. Let

$$\mathbb{T} = \{\mathcal{T} \subseteq \mathcal{T}_B : \mathcal{T} \text{ is compatible and } \psi\text{-sufficient}\};$$

and for each $\mathcal{T} \in \mathbb{T}$, let $\phi_{\mathcal{T}}$ be the composite strategy constructed from $\mathcal{T}$. Then the **optimal composite strategy problem** is the problem of finding a composite strategy $\phi_{\mathcal{T}^*}$ such that $\mathcal{T}^* \in \mathbb{T}$ and $EU_A(\phi_{\mathcal{T}^*}, \psi) \geq EU_A(\phi_{\mathcal{T}}, \psi)$ for every $\mathcal{T} \in \mathbb{T}$. We say that $\phi_{\mathcal{T}^*}$ is $(\mathcal{T}_B, \psi)$-*optimal*.

Here is another formulation of the optimal composite strategy problem that's equivalent to the above formulation. Suppose we are given sets of interaction traces $\mathcal{T}_1, \ldots, \mathcal{T}_m$ from games against several different agents $\lambda_1, \ldots, \lambda_m$, and for each $\tau_i \in \mathcal{T}_i$ we are given the utility $u_A(\tau_i)$ for the agent that played against λ_i. Furthermore, suppose we are given numbers $p_1, \ldots, p_m$ such that p_i is the probability that we'll need to play against λ_i. Now, consider the problem of finding a strategy with an optimal expected utility against these agents. This is equivalent to an optimal composite strategy problem in which $\mathcal{T}_B = \{\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m\}$ and $\psi = (\{\phi_1, \ldots, \phi_m\}, \Delta)$, where $\phi_i = \mathsf{strategy}(\lambda_i)$ and $\Delta(\phi_i) = p_i$ for each i.

Procedure CIT($k, \{\mathcal{T}_i\}_{i=1,\ldots,m}, \{p_i\}_{i=1,\ldots,m}$)

1. If $k > n$, then ** n is the maximum number of interactions */*
2. For $i = 1$ to m, choose any one trace, namely τ_i, in $\mathcal{T}_i$
3. Return $(\{\tau_i\}_{i=1..m},\ \sum_{i=1..m}\{p_i \times u_A(\tau_i)\})$
4. Else
5. $A_i := \{a_k : \langle (a_j, b_j)\rangle_{j=1..n} \in \mathcal{T}_i\}$, for $i := 1..m$
6. $B_i := \{b_k : \langle (a_j, b_j)\rangle_{j=1..n} \in \mathcal{T}_i\}$, for $i := 1..m$
7. $A' := A_1 \cap A_2 \cap \ldots \cap A_m$; $B' := B_1 \cup B_2 \cup \ldots \cup B_m$
8. If $A' = \emptyset$, then return $(\emptyset, -1)$ */* incompatibility detected */*
9. If $|B_i| \neq 1$ for some i, then exit with failure.
10. For $i := 1$ to m
11. Partition $\mathcal{T}_i$ into $\mathcal{T}_i^{ab}$ for each pair $(a, b) \in A' \times B'$ such that
12. $\mathcal{T}_i^{ab} := \{\tau \in \mathcal{T}_i$: the k'th interaction in τ is $(a, b)\}$
13. For each $(a, b) \in A' \times B'$
14. $\mathbb{T}^{ab} := \{\mathcal{T}_i^{ab} : 1 \leq i \leq m, \mathcal{T}_i^{ab} \neq \emptyset\}$
15. $\mathcal{P}^{ab} := \{p_i : 1 \leq i \leq m, \mathcal{T}_i^{ab} \in \mathcal{T}^{ab}\}$
16. $(\mathcal{T}_*^{ab}, U_*^{ab}) :=$ CIT$(k + 1, \mathbb{T}^{ab}, \mathcal{P}^{ab})$ */* call CIT itself */*
17. For each $a \in A'$
18. If $U_*^{ab} \geq 0$ for all $b \in B'$, then
19. $\hat{\mathcal{T}}_a := \bigcup_{b\in B'}\{\mathcal{T}_*^{ab}\}$; $\hat{U}_a := \sum_{b\in B'} U_*^{ab}$
20. Else */* i.e., $\mathcal{T}_*^{ab}$ is not a solution for some $b \in B'$ */*
21. $\hat{\mathcal{T}}_a := \emptyset$; $\hat{U}_a := -1$ */* i.e., no solution */*
22. $a^{max} := \arg\max_{a\in A'}\{\hat{U}_a\}$
23. Return $(\hat{\mathcal{T}}_{a^{max}}, \hat{U}_{a^{max}})$

Figure 2.3: The pseudocode of the CIT algorithm.

As an example, suppose we want to play the IPD against two agents λ_1 = TFT and λ_2 = TFTT, who will be our opponents with probabilities $p_1 = 0.7$ and $p_2 = 0.3$, respectively. Suppose we are given $\mathcal{T}_1 = \{\tau_1, \tau_2, \tau_3\}$, where

$$\tau_1 = \langle (C,C),(C,C),(D,C)\rangle; \quad u_A(\tau_1) = 11.0;$$

$$\tau_2 = \langle (D,C),(C,D),(C,C)\rangle; \quad u_A(\tau_2) = 8.0;$$

$$\tau_3 = \langle (D,C),(D,D),(D,D)\rangle; \quad u_A(\tau_3) = 7.0;$$

and $\mathcal{T}_2 = \{\tau'_1, \tau'_2, \tau'_3\}$, where

$$\tau'_1 = \langle (C,C),(C,C),(C,C)\rangle; \quad u_A(\tau'_1) = 9.0;$$

$$\tau'_2 = \langle (D,C),(C,C),(D,C)\rangle; \quad u_A(\tau'_2) = 13.0;$$

$$\tau'_3 = \langle (D,C),(D,C),(D,D)\rangle; \quad u_A(\tau'_3) = 11.0.$$

We can map this into the optimal composite agent problem by letting $\psi = (\Phi_B, \Delta_{\Phi_B})$ and $\mathcal{T}_{\Phi_B} = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\Phi_B = \{\phi_1, \phi_2\}$, $\Delta_{\Phi_B}(\phi_1) = 0.7$, and $\Delta_{\Phi_B}(\phi_2) = 0.3$.

There are nine subsets of $\mathcal{T}$ that contain one trace for each of λ_1 and λ_2 (and hence are ψ-sufficient), namely $\{\tau_j, \tau'_k\}$ for $j = 1,2,3$ and $k = 1,2,3$. Only two of these nine sets are compatible: $\{\tau_2, \tau'_2\}$ and $\{\tau_3, \tau'_3\}$. Of the two sets, $\{\tau_2, \tau'_2\}$ is the (Φ_B, ψ)-optimal one, because $EU_A(\text{CA}(\{\tau_2, \tau'_2\}), \psi) = 9.5 > 8.2 = EU_A(\text{CA}(\{\tau_3, \tau'_3\}), \psi)$. Hence, our (Φ_B, ψ)-optimal strategy is to choose D and C in the first two iterations, and then choose C (or D) in the third iteration if the opponent chooses D (or C) in the second iteration, respectively.

Figure 2.3 shows an algorithm, CIT, that can be used to solve the above problem. CIT is a recursive algorithm that works by analyzing interaction traces played against a set of agents $\{\lambda_1, \ldots, \lambda_m\}$. Its inputs include, for each λ_i, a set of interaction traces $\mathcal{T}_i$ and a probability p_i that we'll have λ_i as our opponent; and a number k that represents

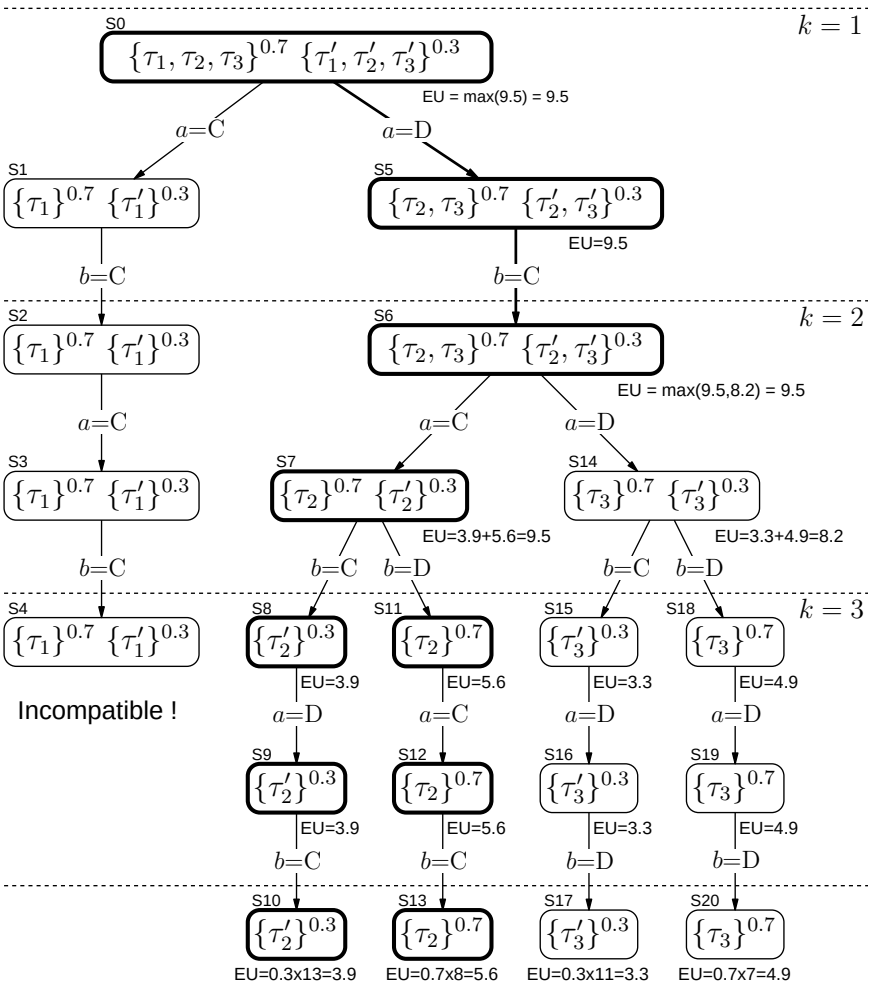

Figure 2.4: The search space of the CIT algorithm.

how many moves deep we have gone in our analysis of the interaction traces.

Example. We'll now illustrate CIT's operation on the same example that we discussed earlier. In Figure 2.4, the node S0 represents the initial call to CIT. The two sets of traces are $\mathcal{T}_1$ and $\mathcal{T}_2$ described earlier, and the superscripts on these traces are the probabilities $p_1 = 0.7$ and $p_2 = 0.3$ described earlier.

Each path from S0 to the bottom of the tree is one of the interaction traces; and the value $k = 1$ at S0 indicates that we're currently looking at the first interaction in each

trace.

At the k'th interaction for each k, we need to consider both our possible moves and the opponent's possible moves. Although these moves are made simultaneously, we can separate the computation into two sequential choices: for each value of k, the higher layer of nodes corresponds to *our* possible moves, and the lower layer of nodes corresponds to the opponent's possible moves. In the higher layer, the expected utility of each node can be computed by taking the maximum of the expected utilities of the child nodes; hence we call these nodes *max nodes*. In the lower layer, the expected utility of each node can be computed by adding the expected utilities of the child nodes; hence we call these nodes *sum nodes*.[4]

In our example, the max node at $k = 1$ is S0, and the sum nodes at $k = 1$ are S1 and S5. The edges between the max node and the sum nodes correspond to our actions that can lead from the max node to the sum nodes. For instance, if our action is C at S0, the next sum node is S1; otherwise, the next sum node is S5. The edges between the sum nodes at $k = 1$ and the max nodes at $k = 2$ corresponds to the actions that can be chosen by the opponent. At $k = 1$, the opponent can only choose C, but at S7 at $k = 2$, the opponent can choose either C or D, thus leading to two different max nodes S8 and S11. A terminal node corresponds to the set of interaction traces that are consistent with the actions on the path from S0 to the terminal node. The expected utility of a terminal node is the sum of the probability of the set of interaction traces (denoted by the superscripts in Figure 2.4) times the utility of any interaction trace in the set. For instance, at S10, the

[4]Mathematically, what we're computing here is a weighted average; but each number has already been multiplied by the appropriate weight, so we just need to add the numbers together.

expected utility is $\sum_{i=1..m}\{p_i \times U_A(\tau_i)\} = 0.3 \times 13 = 3.9$. Notice that all interaction traces in a set of interaction trace of a terminal node are the same; thus the algorithm chooses any τ_i from $\mathcal{T}_i$ at Line 2 since they all have the same utility.

The CIT algorithm basically does a depth-first search on a tree as shown in Figure 2.4, and propagates the expected utilities of the terminal nodes to S0 together with the compatible set of interaction traces that gives the expected utility. The expected utility of a max node is the maximum of the expected utilities of its child nodes, whereas the expected utility of a sum node is the sum of the expected utilities of its child nodes. Notice that at each max node the CIT algorithm will check the compatibility of the set of interaction traces of the node at Line 8. For example, at S4 the algorithm discovers that τ_1 and τ_1' are incompatible. Then a failure signal (the expected utility -1) is passed from S4 to the ancestor nodes. The max nodes and the sum nodes would then ignore the solution with a failure signal, thus eliminate the incompatibility. This is one of the key difference between the CIT algorithm and other search algorithms for game trees or MDPs—the CIT algorithm directly operates with interaction traces and checks the incompatibility among them as the search process proceeds.

Running time. To achieve efficient performance in CIT we have used some indexing schemes that we will not describe here, due to lack of space. CIT's running time is $O(nM)$, where n is the number of iterations and $M = \sum_{i=1}^{m} |\mathcal{T}_i|$. In practice, the CIT algorithm is very efficient—it can find the optimal solution from a database of $500,000$ interaction traces in less than 15 minutes on a computer with a 3.8GHz Pentium 4 CPU and 2GB RAM.

Discussion. At first glance, CIT's search space (see Figure 2.4) looks somewhat like a game tree; but there are several important differences. First, our purpose is to compute an agent's entire strategy offline, rather than having the agent do an online game-tree search at every move of a game. Second, we are not searching an entire game tree, but are just searching through a set of interaction traces; hence the number of nodes in our tree is no greater than the total number of interactions in the interaction traces. Third, game-tree search always assumes, either explicitly or tacitly, a model of the opponent's behavior.[5] Rather than assuming any particular model of the opponent, we instead proceed from observations of agents' actual interactions.

2.3.4 Using a Base Strategy

Recall that CA($\mathcal{T}$)'s strategy is partial. In particular, although CA($\mathcal{T}$) will never exit with failure when playing against ψ when $\mathcal{T}$ is compatible and ψ-sufficient, it might do so if we play it against another opponent λ'_B with a mixed strategy $\psi' = (\Phi'_B, \Delta_{\Phi'_B})$ for which $\mathcal{T}$ is not ψ'-sufficient. However, in iterated games such as the IPD, there is enough overlap in the strategies of different agents that even if λ'_B was not used to construct any of the traces in $\mathcal{T}$, $\mathcal{T}$ may still be ψ'-sufficient.

Even if $\mathcal{T}$ is not ψ'-sufficient, CA($\mathcal{T}$) may still be able to play against λ'_B most of

[5] For example, minimax game-tree search assumes that the opponent will always use its dominant strategy. In perfect-information games such as chess, this opponent model has worked so well that it is taken more-or-less for granted. But in an imperfect-information variant of chess, it has been shown experimentally [52] that this model is not the best one—instead, it is better to use a model that assumes the opponent has very little idea of what its dominant strategy might be.

Agent MCA($\mathcal{T}_B, \Delta, \lambda_{base}$) /* *a modified composite agent* */

1. $\mathcal{T}^* :=$ CIT$(0, \mathcal{T}_B, \Delta)$ /* $\mathcal{T}^*$ *is* $(\mathcal{T}_B, \psi)$*-optimal* */
2. $\lambda_A :=$ CA$(\mathcal{T}^*)$ /* λ_A *is a composite strategy* */
3. Loop until the end of the game
4. Get an action a from λ_A
5. If λ_A fails, then get an action a from λ_{base} and $\lambda_A := \lambda_{base}$
6. Output a and receive the other agent's action b
7. Give b to λ_A as its input

Figure 2.5: Algorithm for a composite agent with a base strategy.

the time without exiting with failure. To handle cases where CA$(\mathcal{T})$ does exit with failure, we can modify the CA algorithm so that instead of exiting with failure, it instead uses the output produced by a "base strategy" λ_{base}, which may be TFT or any other strategy. We call this modified algorithm the *modified composite agent* (MCA), and its pseudo-code is shown in Figure 2.5. The strategy of modified composite agents is called the *modified composite strategy*. A modified composite agent may be viewed in either of two ways:

- As a composite strategy that can use λ_{base} when the composite strategy is insufficient.
- As an enhanced version of λ_{base}, in which we modify λ_{base}'s moves in cases where the set of traces $\mathcal{T}_B$ might yield a better move. From this viewpoint, $\mathcal{T}_B$ is a case base that is processed by the CIT algorithm so that cases can be selected quickly when they are appropriate.

2.4 Experimental Evaluation

We evaluated our technique in three well-known games: the Iterated Prisoner's Dilemma (IPD), the Iterated Chicken Game (ICG), and the Iterated Battle of the Sexes (IBS). The IPD and ICG are the iterated versions of the Prisoner's Dilemma [3] and the Game of Chicken [23], whose payoff matrices are shown in Figure 2.1 and Figure 2.2.

The IBS is the iterated version of the Battle of the Sexes [42], whose payoff matrix is shown below. To allow arbitrary agents to play against each other without having to take on different roles (hence different strategies), we needed to reformulate the IBS to make the roles of Husband and Wife interchangeable. This was quite easy to do, as shown in Figure 2.3: for the Wife we renamed Football to C and Opera to D; and for the Husband we renamed Football to D and Opera to C.

2.4.1 Experimental Design

To obtain a large collection of agents for the games, we asked the students in several advanced-level AI classes to contribute agents. We did not tell the students the exact number of iterations in each game, but did tell them that it would be at least 50 (in all of our experiments we used 200 iterations). The students contributed 43 IPD agents, 37 ICG agents, and 37 IBS agents. For each game, we also contributed 9 more agents that used the following well-known strategies: ALLC, ALLD, GRIM, NEG, PAVLOV, RAND, STFT, TFT, and TFTT.[6] This gave us a total of 52 IPD agents, 46 ICG agents, and 46 IBS agents.

[6]These are often used as standard strategies; e.g., they were used as standard entries in the 2005 IPD tournament [33].

Table 2.1: The payoff matrix of the Prisoner's Dilemma.

Prisoner's Dilemma		Player B	
		Cooperate (C)	Defect (D)
Player A	Cooperate (C)	$(3, 3)$	$(0, 5)$
	Defect (D)	$(5, 0)$	$(1, 1)$

Table 2.2: The payoff matrix of the Chicken Game.

Chicken Game		Player B	
		Cooperate (C)	Defect (D)
Player A	Cooperate (C)	$(4, 4)$	$(3, 5)$
	Defect (D)	$(5, 3)$	$(0, 0)$

Table 2.3: The payoff matrix of the Battle of the Sexes.

Battle of the Sexes		Husband	
		D (was Football)	C (was Opera)
Wife	C (was Football)	$(1, 2)$	$(0, 0)$
	D (was Opera)	$(0, 0)$	$(2, 1)$

In the rest of this section, we'll call these the *original* agents, to distinguish them from the composite agents generated by our algorithms. In each game, each player's utility is the sum of that player's payoff in each of the 200 iterations in an interaction trace.

For each agent λ, we wanted to find out how much improvement we could get by replacing λ with a modified composite agent whose base strategy is λ. We investigated this by doing a 5-fold cross-validation experiment that worked as follows.

For each of the three games (IPD, ICG, and IBS), we took our original set of agents for the game, and randomly and evenly partitioned it into five subsets $\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4, \Lambda_5$. Then we repeated the following steps five times, once for each Λ_i:

1. For the test set Λ_{test}, we chose Λ_i. For the training set Λ_{train}, we chose $\bigcup_{j \neq i} \Lambda_j$.

2. We ran a 200-iteration tournament (the details are described below) among the agents in Λ_{train} (with one modification, also described below), and let $\mathcal{T}_{train}$ be the set of all interaction traces recorded in this tournament.

3. For each agent $\lambda \in \Lambda_{train}$, we played 100 tournaments, each 200 iterations long, involving λ and all of the agents in Λ_{test}. We calculated the agent's *average score*, which is equal to $\frac{1}{100 \times |\Lambda_{test}|} \sum_{1 \leq l \leq 100} \sum_{\lambda_k \in \Lambda_{test}} \{$ the score of λ when it plays against λ_k in the l'th tournament$\}$.

4. Likewise, for each agent $\lambda \in \Lambda_{train}$, we played 100 tournaments, each 200 iterations long, involving a modified composite agent $\lambda' = \text{MCA}(\mathcal{T}_{train}, \Delta, \lambda)$ and all of the agents in Λ_{test}, where Δ is a uniform probability distribution over Λ_{train}. Apart from the average score of λ', we also recorded the frequency with which λ' used its base strategy λ.

Table 2.4: Among the original agents, how many of each type.

	IPD	ICG	IBS
Deterministic agents	34	22	17
Probabilistic agents	18	24	29

All of our tournaments were similar to Axelrod's IPD tournaments [3] and the 2005 IPD tournament [33]. Each participant played against every participant including itself (thus a tournament among n agents consists of n^2 iterated games).

One problem in Step 2 is that MCA's input needs to come from deterministic agents, but many of our students' agents were probabilistic (see Table 2.4). We handled this problem as follows. Each probabilistic agent λ used a random number generator for which one can set a starting seed. By using ten different starting seeds, we created ten determinized versions of λ; and we generated interaction traces using the determinized agents rather than λ.

Note that we used the determinized agents *only during training*. The modified composite agents generated from the training data are quite capable of being played against probabilistic agents, so we played them against the probabilistic agents in our tests.

2.4.2 Experimental Results

Table 2.5 tells how many traces, on average, were collected for each type of game, and how many traces were in the composite strategies generated by CIT.

In each experiment, we calculated 4 average scores for each agent (one for each test

Table 2.5: Average number of interaction traces collected during the training sessions, before and after removing duplicate traces, and average number of interaction traces in the composite strategies generated by the CIT algorithm.

	IPD	ICG	IBS
Before removing duplicates	29466.0	44117.4	60470.5
After removing duplicates	7452.0	25391.5	29700.8
Composite strategies	171.2	209.6	245.6

sets that did not contain the agent) and 4 average scores for each MCA agent. We repeat the above experiment 100 times using different partitions and random seeds. Figure 2.6–2.8 show the agents' *overall average scores*, each of which is an average of 400 average scores. Since each average score is an average of $100 \times |\Lambda_{test}|$ scores, each data point in the figures is computed from the scores of $40000 \times n$ games, where n is the average number of agents in the test sets. Hence in the IPD, each data point is computed from the scores of approximately 415769.2 games, and in both the ICG and the IBS each data point is computed from the scores of approximately 367826.1 games. The data file of the experiments can be downloaded at `http://www.cs.umd.edu/~chiu/papers/Au08synthesis_data.tar.gz`.

In each graph in Figure 2.6–2.8, the x axis shows the agents and the y axis shows their scores. The lower line shows the performance of each original agent λ (the agents are sorted in order of decreasing overall performance). The upper line shows the average

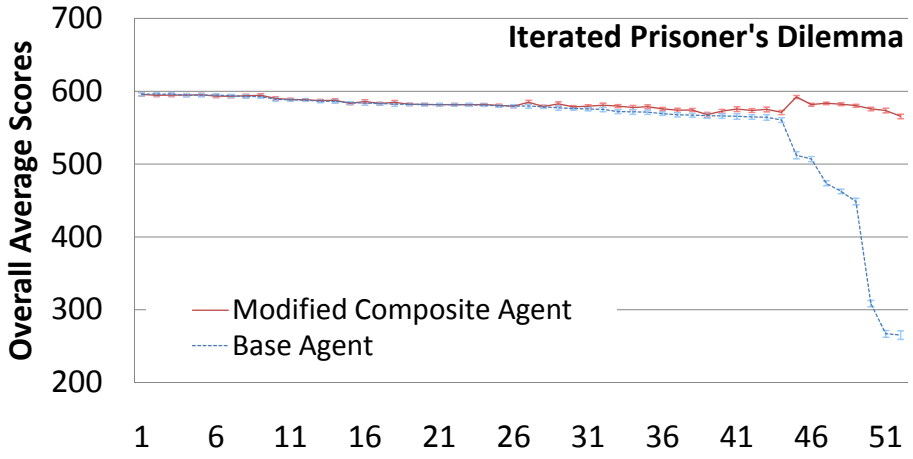

Figure 2.6: Overall average scores of the base agents and the MCA agents in the IPD. The agents are displayed on the x axis in order of decreasing score of the base agent. The error bars denote the 95% confidence intervals of the overall average scores.

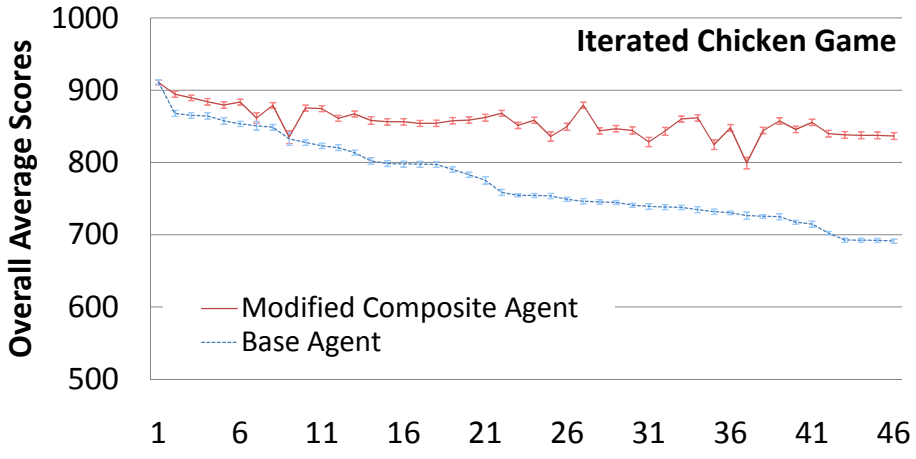

Figure 2.7: Overall average scores of the base agents and the MCA agents in the ICG.

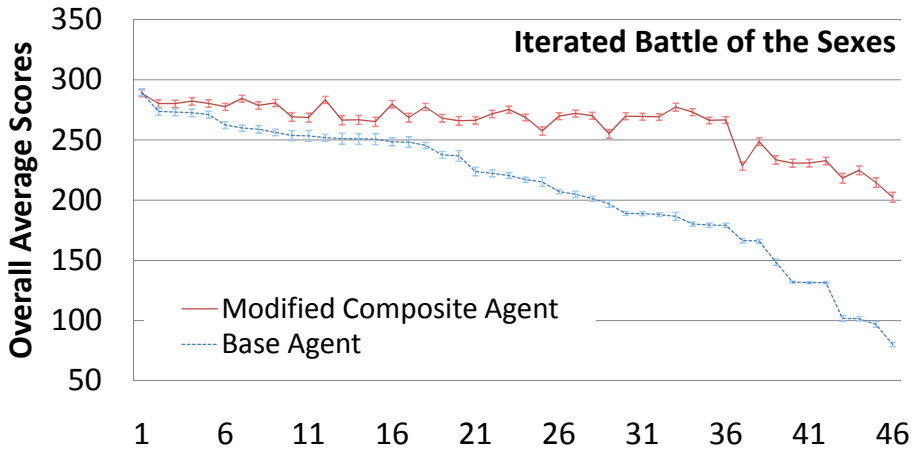

Figure 2.8: Overall average scores of the base agents and the MCA agents in the IBS.

performance of the corresponding MCA agents.

From the graphs, we note the following. In all three games, the average scores of the MCA agents were as good or better than the corresponding base agent. In the IPD, the differences in performance were usually small; and we believe this is because most IPD agents have similar behaviors and therefore leave little room for improvement. In the ICG and IBS, the differences were usually quite large. Finally, in all three games, the MCA agents performed well even when their base agents performed poorly. For example, in the IPD and the IBS, when we incorporated our composite strategy into the weakest of the existing strategies, it more than doubled that strategy's score.

Figure 2.9 shows the increase in rank of an agent after incorporating the composite strategy into it, while all other agents did not incorporate the composite strategy. We can see that the ranks of most agents increased after the modification. We conducted sign tests to see whether the overall average scores of MCAagents are greater than that of the original agents. The p-values of one-sided sign tests are less than 0.00001 in all games.

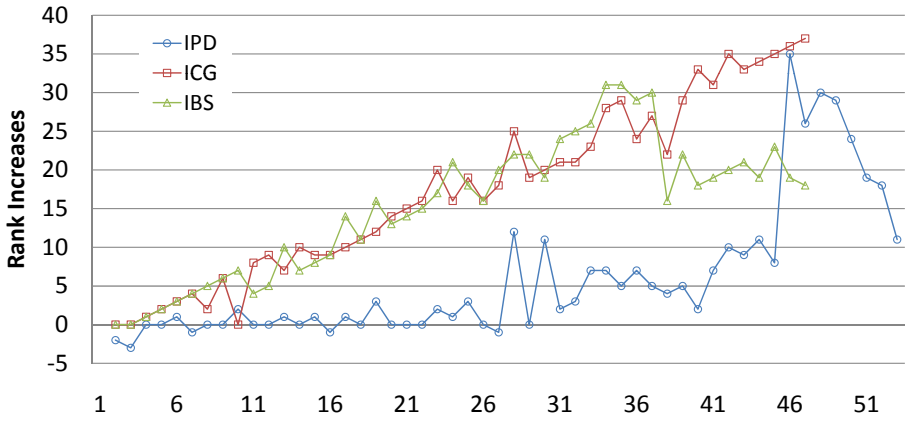

Figure 2.9: Increase in rank of each enhanced agent, relative to the corresponding base agent. The x axis is as in Figure 2.6–2.8.

Therefore, the MCA agents are significantly better at the 99.9% level.

Table 2.6 shows the average improvement that each MCA agents provided relative to the corresponding original agent. On the average, the percentage improvements in score were about 5% in the IPD, 11% in the ICG, and 26% in the IBS; and the percentage improvements in rank were about 12% in the IPD, 38% in the ICG, and 33% in the IBS. Hence, the use of composite strategies greatly enhance the performance of most of the agent.

Table 2.7 compares the average scores of the best agents, with and without the composite strategy. We can see that the average scores are more or less the same. Thus, the best strategies in the tournaments does not benefit from the use of composite strategies. We believe the reason for that is that the best strategies have already had a decent set of behavior that are similar to the interaction traces in the composite strategies.

Table 2.8 shows how frequently the MCA agents invoked their base agents. We can

Table 2.6: Average increases in score and rank of the MCA agents, relative to the corresponding original agents.

	IPD	ICG	IBS
Average MCA agent score	582.69	856.37	262.47
Average original agent score	553.62	774.38	208.73
Average difference in score	29.08	81.99	53.74
Average % difference in score	5.3%	10.6%	25.7%
Average increase in rank	6.0	17.4	15.2
Average % increase in rank	11.5%	37.8%	33.0%

Table 2.7: Average scores of the best original agent, and the MCA agent whose base strategy is that agent.

	IPD	ICG	IBS
MCA agents	477.23	729.47	232.61
Original agents	477.46	729.50	232.46

Table 2.8: Average frequency of invocation of base strategies.

	IPD	ICG	IBS
Average percentage of games	15.4%	52.7%	54.4%

make the following observations.

- In more than 84% of the IPD games, the MCA agents did not need to use their base strategies at all. In other words, the MCA agents' composite strategies worked successfully throughout those games, even though the games were played with opponents other than the ones used to build the composite strategies. One possible the reason for this high reusability of interaction traces is that the IPD is a well known game and thus most of the original strategies are similar to certain well-known strategies such as Tit-for-Tat.

- In the ICG and IBS, the MCA agents invoked their base strategies a little more than half of the time. We think the reason for this is that there was more diversity among the original strategies, perhaps because these games are not as well-known as IPD. But even though the MCA agents used their composite strategies less frequently than in the IPD, the composite strategies provided much more improvement, relative to the base strategy, than in the IPD. In other words, the places where MCA was used its composite strategy provided a much bigger enhancement to the base strategy's performance.

2.5 Comparisons with Other Work

Reinforcement learning. One similarity between our approach and reinforcement learning is that our technique improves an agent's performance by incorporating previous experiences. But reinforcement learning agents usually use on-line learning (e.g., Q-learning agents learn and improve their policy during acting), while our composite agent uses off-line learning to produce a composite strategy that can then be incorporated into an agent.

A more important difference is that our technique does not require us to know the set of all possible states of the other agents, as opposed to most existing work on POMDPs or learning automata, in which the set of all possible states must be known beforehand (e.g., [28] and [41]) In open environments such as IPD tournaments in which the opponents' strategies are not known, it would be difficult, if not impossible, to identify all possible states of the opponent's strategy. Moreover, the Markovian assumption intrinsic to a policy or automaton does not always hold because an agent's decision can depend on the entire history in a game. Our paper demonstrates that it is possible to perform well in certain open environments such as the IPD without knowing the set of the opponents' internal states. To the best of our knowledge, contemporary reinforcement learning techniques are, unlike our techniques, not yet efficient enough to compete with strategies such as TFT and Pavlov in the IPD. In future, we would like to run a much wider variety of experiments to see whether our technique can be applicable in other open environments.

Modeling other agents. One approach for playing against a given opponent is to develop an *opponent model* that predicts the opponent's likely behavior [14, 22, 27]. This model is then used to aid in developing strategies against that opponent. In contrast, we

focus on generating strategies directly from observations, without constructing an explicit opponent model.

Case-based reasoning. Our technique has some similarities to Derivational Analogy [63] and the reuse of macro actions/operators [34], in which records of a problem-solver's decisions or actions are used to solve new problems. Most work on derivational analogy and macro actions focuses on problems in which there is no need to interact with the environment during problem solving. But there are some works on using derivational analogy or macro actions in interactive environments [13, 43, 54]. In these domains, it would be beneficial not to discard the observation sequences generated by the environments, since the observation sequences capture important information that can be used to determine whether an agent can utilize two different action sequences in the same problem (see Definition 2). Our work pushes this idea further by showing how to construct a strategy from interaction traces.

Case-based reasoning techniques have been used to improve existing reinforcement learning techniques [25, 65]. But so far case-based reasoning played a supporting role only in these work. In contrast, our technique generates fully-functional agents out of previous problem solving experiences, without the help of existing reinforcement learning techniques.

The winner of the 2005 IPD tournament is based on some sort of case-based reasoning technique [40]. Similarly, the winning strategy of our ICG tournament is a combination of three different ways to deal with three different type of opponents. The success of these strategies compels us to believe that one way to dominate monolithic strategies

such as Tit-for-Tat is to combine the winning strategies for different type of opponents together.

One problem with the case-based reasoning strategies mentioned in the above paragraph is that the ways they combine strategies together are quite ad hoc, and only manage to consider a handful of possible opponents' strategies. Our work shows a systematic way to combine strategies that can be scaled up to handle a large number of different opponent's strategies.

2.6 Summary

The idea that an agent can improve its performance by observing interactions among other agents is not new. What is new in this paper is how to do it without the knowledge of the set of opponents' internal states. In open environments such as IPD tournaments, we know little about the opponents' strategy, let alone the current state of the opponents' strategy. Hence methods that do not require us to know the opponents' states can be quite valuable in such domains.

To avoid using the notion of states or belief states as in policies or automata, our approach directly selects and combines the records of the other agents' interactions to form a partial strategy called a composite strategy, which maximizes the expected utility with respect to an estimated probability distribution of opponents that can possibly generate those interactions, without knowing the states or other details of the strategies of these opponents. The composite strategy can be used to decide how an agent should respond to typical behaviors of opponents in an environment. Out of a set of 126 agents in three

iterated games (the IPD, ICG, and IBS), our technique significantly improved the agent's rank (compared to the other agents) by 12% in the IPD, 38% in the ICG, and 33% in the IBS, after incorporating the partial strategy.

In future, we would like to address the following issues:

- In iterated games such as the IPD, ICG, and IBS, players frequently encounter the game situations that have been seen before; hence combining the interaction traces was sufficient to provide large increases in an agent's performance. By itself, this would not be suitable for large-scale non-zero-sum games such as chess, where games among experts usually lead to situations that no player has ever seen before. We would like to develop a way to combine our technique with game-tree search, to see how much that extends the scope of the technique.

- Composite strategies in our experiments usually do not fail even if some agents in the test sets are probabilistic. The reason is that agents in non-zero-games we considered exhibit relatively small number of different behaviors, such that the sizes of the sizes of the composite strategies are large enough to cover most of these behaviors. However, in other domains agents may exhibit so many different behavior that cannot be covered by a small number of interaction traces. In the future, we would like to address this problem when we extend this technique to other domains.

- We've done preliminary tests on some nondeterministic partially-observable planning domains (e.g., transportation domains with unknown traffic patterns). Our approach worked quite well in our tests, but we need to do larger cross-validated experiments before reporting the results. In future, we intend to generalize our

method for domains that are more complicated than two-player repeated games.

- A significant limitation of our technique is that it requires an estimate of the probability with which we'll encounter each of opponents we have observed. In the future, we would like to study how to estimate the probability from a database of interaction traces and modify our techniques to alleviate this requirement.

- Our current formalization cannot handle problems whose horizon is infinite. We would like to see how to extend our algorithm to deal with interaction trace of potentially infinite length.

- The CIT technique can be considered as a technique for *transfer learning*, the transfer of learned knowledge from one situation to another situation (e.g., [6]). The idea is that the composite strategy learned in one non-zero-sum games should enable the system to learn more quickly how to play another non-zero-sum game. In future, we would like to see how to transfer interaction traces recorded in one situation to another in order to speed up the learning process.

Chapter 3

Task-Completion Problems with Goal Uncertainty

In Chapter 2, we presented the Combining Interaction Traces (CIT) technique for synthesizing a strategy from a database of interaction traces. An interesting question is whether this technique is applicable to domains other than repeated games. There is a large class of problems in which an agent must perform a sequence of interactions with an environment that responds non-deterministically to these actions, in order to achieve a goal. The environment can be any environment with contingencies, including an opponent in a game or a city with traffic lights and accidental events. In this chapter, we consider the task of adapting the CIT technique to these problems.

It turns out that we cannot directly apply the CIT technique to all kind of problems in which an agent has to interact with an environment. The issue is that some problems are inherently *unsolvable*—there does not exist a strategy that can guarantee the success of an agent in accomplishing the given task. For these problems, it is impossible to synthesize a sure-win strategy, no matter how clever the strategy synthesis algorithm is.

The concept of unsolvable problems in agent-environment interaction resembles impossibility theorems in distributed systems [29] and the concepts of observability and controllability in control systems. In distributed systems, researchers know that some problems (e.g., distributed consensus with one faulty process) are fundamentally unsolvable. Remedying these problems is far from trivial, because it requires a modification to

the fundamental assumptions of the distributed system model people have been used. In control theory, controllability is about the possibility of steering a system to a particular state by using an appropriate control signal—if a state is not controllable, no control signal can control the state. In short, it is impossible to design a controller to achieve certain desired properties in a system. But our focus in this chapter is not the solvability of all kinds of systems, but the solvability of problems in which an agent has to interact with an environment in order to accomplish certain tasks. Repeated games such as the IPD, of course, belongs to this class of problems.

We will study *task-completion problems*, in which an agent has to interact with an environment in order to accomplish a goal set by a given task. There are many problems in which an agent needs to complete a given task by interacting with a partially observable and/or non-deterministic environment. In these problems, the challenge for the agent is to explore an unknown world while solving the problem. However, not every environment is safely explorable [57, page 125]; The *exploration-exploitation dilemma* [19, 20] implies that it is not always possible for the agent to do this successfully (or optimally), because during exploration the agent may reach a state in which it is no longer possible to complete the task. The problem is compounded by the *uncertainty about goals*—the goals in the problem can vary from one situation to another, and the agent is uncertain about the goals it needs to achieve at the beginning of the interaction. Thus, the strategy synthesis algorithm must take both the nondeterministic behavior of the environment and the goal uncertainty into account when creating a new strategy for solving the problems.

First, we characterize problems that are unsolvable by using the compatibility of interaction traces. We provide theorems giving conditions under which it is possible or

impossible to construct strongly successful strategies (i.e., strategies that are guaranteed to be successful). For cases where no strongly successful agent is possible, we provide mathematical results giving the probability that an agent can be successful. Second, we present provably correct algorithms to analyze a database of interaction traces from previous problem-solving episodes, in order to construct a strongly successful strategies if one exists, or construct a strategy that has the highest possible probability of success given the interaction traces. These algorithms can be considered as the adaptations of the CIT algorithm in Chapter 2 for task-completion problems. Finally, we provide theoretical and experimental results demonstrating the algorithm's performance.

To summarize, the contributions of this chapter are:

- We provide necessary and sufficient conditions (on an environment's set of possible interaction traces) for there to exist an agent that can *always* successfully accomplish a task. If a task-completion problem satisfies this condition, we say the problem is *strongly solvable*.

- We provide an algorithm that takes a collection of interaction traces from previous problem-solving episodes, and constructs an agent with the highest probability of success among all combinations of the interaction traces.

- We present experimental results showing that even with a small number of interaction traces, the agent constructed by our algorithm performs well.

3.1 Basic Definitions

Our definitions of agents and environments are based on Chapter 2 of [57]. An agent interacts with an environment by performing *actions* in some finite set A and receiving *percepts* in some finite set B. In repeated games, an environment is the opponent and percepts are opponent's actions.

We assume the interactions between an agent λ and an environment e occur at discrete time points t_1, t_2, ... At each time t_i, an agent performs an action $a_i \in A$ and receives a percept $b_i \in B$. An *interaction trace* of length N is a sequence of interactions $\langle (a_1, b_1), (a_2, b_2), \ldots, (a_N, b_N) \rangle$. For simplicity, we also denote an interaction trace as a pair $\tau = (\alpha, \beta)$, where $\alpha = \langle a_1, \ldots, a_N \rangle$ is an action sequence, and $\beta = \langle b_1, b_2, \ldots, b_N \rangle$ is a percept sequence.[1] Notice that the length of α and β must be equal. Let $\mathcal{T}^*$ be the set of all interaction traces, i.e., $\mathcal{T}^* = \{(\alpha, \beta) : \alpha \in A^*, \beta \in B^*, |\alpha| = |\beta|\}$, where A^* is the set of all action sequences, and B^* is the set of all percept sequences.

Let $\tau = (\alpha, \beta)$. Then we let

$$\mathsf{actions}(\tau) = \tau^A = \alpha;$$

$$\mathsf{percepts}(\tau) = \tau^B = \beta.$$

By extension, if $\mathcal{T}$ is a set of interaction traces, then we let

$$\mathsf{actions}(\mathcal{T}) = \{\tau^A : \tau \in \mathcal{T}\};$$

$$\mathsf{percepts}(\mathcal{T}) = \{\tau^B : \tau \in \mathcal{T}\}.$$

[1] Although we assume that a_i and b_i occur simultaneously, our formalism is general enough to model environments where they are interleaved. This can be done by specifying that odd time points the actions have no effect, and that at even time points the environment always returns a "null" percept.

Let μ be any sequence such as action sequences, percept sequences, or interaction sequences. Then $[\mu]_k$ denotes the k'th element in μ, $\mathsf{prefix}_k(\mu)$ be the k-*prefix* of μ, and $\mathsf{suffix}_k(\mu)$ be the k-*suffix* of μ.

More precisely, we define a k-prefix and a k-suffix as follows: Let $\mu = \langle m_1, m_2, \ldots, m_n \rangle$, where m_i can be either an action or a percept. Then $\mathsf{prefix}_k(\mu) = \langle m_1, \ldots, m_{min(k,n)} \rangle$ and $\mathsf{suffix}_k(\mu) = \langle m_{min(k,n)+1}, \ldots, m_n \rangle$. If μ is an interaction trace τ, then $\mathsf{prefix}_k(\tau) = (\mathsf{prefix}_k(\tau^A), \mathsf{prefix}_k(\tau^B))$, and $\mathsf{suffix}_k(\tau) = (\mathsf{suffix}_k(\tau^A), \mathsf{suffix}_k(\tau^B))$.

3.1.1 Agents and Environments

We consider three types of environments: deterministic, nondeterministic, and probabilistic:

- A *deterministic environment* is a function $\mathsf{e} : A^* \rightarrow B$. If α is an action sequence of length k, then $\mathsf{e}(\alpha)$ is the percept returned by e at time t_{k+1}.

- A *nondeterministic environment* is a function $\bar{\mathsf{e}} : \mathcal{T}^* \rightarrow 2^B$. If τ is an interaction sequence of length k, then at time t_{k+1}, $\bar{\mathsf{e}}$ may return any of the percepts in $\bar{\mathsf{e}}(\tau)$.

- A *probabilistic environment* is a pair $\hat{\mathsf{e}} = (\bar{\mathsf{e}}, \delta)$, where $\bar{\mathsf{e}}$ is as defined above and $\delta : \mathcal{T}^* \times B \rightarrow [0, 1]$ is a function such that for every $\tau \in \mathcal{T}^*$, the function $\delta_\tau(b) = \delta(\tau, b)$ is a probability distribution over B.

In Chapter 2, we distinguish agents from their strategies, in spite of the one-one correspondence between an agent and the strategy the agent uses. But in the literature, an

agent is sometimes considered as an *action function*, which actually is the strategy of the agent rather than the agent itself. In this chapter, action functions and strategies mean the same thing.

A *deterministic agent function* ϕ is a function $\phi : B^* \to A$; a *nondeterministic agent function* is a function $\bar{\phi} : \mathcal{T}^* \to 2^A$; and a *probabilistic agent function* is a pair $\hat{\phi} = (\bar{\phi}, \delta)$, where δ_τ is a probability distribution over $\bar{\phi}(\tau)$.

A *deterministic agent* is an agent whose action function is deterministic; a *nondeterministic agent* is an agent whose action function is nondeterministic; and a *probabilistic agent* is an agent whose action function is probabilistic.

3.1.2 Equivalences

Mathematically, deterministic agent functions and deterministic environments are pure strategies, whereas probabilistic agent functions and probabilistic environments are mixed strategies. Nondeterministic agent functions is equivalent to a set of pure strategies, so does nondeterministic environments.

To see why it is the case, consider a nondeterministic environment $\bar{\mathsf{e}}$. One can construct a set $\mathbb{E}^{\bar{\mathsf{e}}}$ of deterministic environments recursively by (1) $\mathbb{E}_0 = \{\{\langle\rangle \to b\} : b \in \bar{\mathsf{e}}(\langle\rangle)\}$, (2) for $k \geq 0$, $\mathbb{E}_{k+1} = \{e_k \cup \{\alpha \to b\} : \mathsf{e} \in \mathbb{E}_k, \alpha \in \mathsf{dom}(\mathsf{e}), a \in A, b \in \bar{\mathsf{e}}((\alpha \circ \langle a\rangle, \mathsf{e}(\alpha)))\}$, and (3) $\mathbb{E}^{\bar{\mathsf{e}}} = \cup_{k\geq 0}\mathbb{E}_k$. We call each of the deterministic environments in $\mathbb{E}^{\bar{\mathsf{e}}}$ a *configuration* of $\bar{\mathsf{e}}$. When an agent interacts with $\bar{\mathsf{e}}$, the agent can imagine that it is interacting with one of the configuration in $\mathbb{E}^{\bar{\mathsf{e}}}$, but it does not know which one it is. We call this configuration the *actual* configuration. According to this viewpoint,

all the contingencies in a nondeterministic environment can be summarized by a single contingency, which is about the ignorance about which configuration is the actual one. That's why we consider $\bar{\mathsf{e}}$ is *equivalent* to $\mathbb{E}^{\bar{\mathsf{e}}}$. Every nondeterministic environment has an equivalent set of deterministic environments. [2].

Let $\hat{\mathsf{e}} = (\bar{\mathsf{e}}, \delta)$ be a probabilistic environment, let $\mathbb{E}^{\bar{\mathsf{e}}}$ be as above, and let $\mathbb{E}^{\hat{\mathsf{e}}} = \mathbb{E}^{\bar{\mathsf{e}}}$. Then there is a probability distribution $\Delta^{\hat{\mathsf{e}}}$ over $\mathbb{E}^{\bar{\mathsf{e}}}$ that is *equivalent* to $\hat{\mathsf{e}}$ in the sense that $\delta_\tau(b) = \sum\{\Delta^{\hat{\mathsf{e}}}(\mathsf{e}) : \mathsf{e} \in \mathbb{E}^{\bar{\mathsf{e}}}, \mathsf{e}(\mathsf{prefix}_j(\tau^A)) = [\tau^B]_{j+1}, 0 \leq j < k, \mathsf{e}(\tau^A) = b\}$. Then $\hat{\mathsf{e}}$ is said to be equivalent to the mixed strategy $(\mathbb{E}^{\hat{\mathsf{e}}}, \Delta^{\hat{\mathsf{e}}})$.

Similarly, every nondeterministic agent $\bar{\phi}$ has an equivalent set of deterministic agents $\Lambda^{\bar{\phi}}$, and every probabilistic agent $\hat{\phi} = (\bar{\phi}, \delta)$ has an probability distribution $\Delta^{\hat{\phi}}$ over the set $\Lambda^{\bar{\phi}}$ of deterministic agents such that $\Delta^{\hat{\phi}}$ is equivalent to δ. Thus, we write $\bar{\phi}$ and $\hat{\phi}$ as $\Lambda^{\bar{\phi}}$ and $(\Lambda^{\bar{\phi}}, \Delta^{\hat{\phi}})$, respectively.

In Chapter 2, mixed strategies are denoted by ψ and sets of pure strategies are denoted by Φ. But for clarity, we should use $\hat{\phi}$ and $\hat{e}$ in place of ψ, while $\bar{\phi}$ and $\bar{e}$ in place of Φ.

The above equivalences will allow us to derive properties of nondeterministic or probabilistic environments by referring to the equivalent sets of deterministic environments. For the rest of the paper, we assume agents and environments are deterministic unless we state otherwise.

[2]However, not every set of deterministic environments $\mathbb{E}$ has an nondeterministic environment whose equivalent set of deterministic environments is exactly $\mathbb{E}$.

3.1.3 Agent-Environment Interaction

When a deterministic agent interacts with a deterministic environment e, there is exactly one interaction trace (α, β) such that $\phi(\mathsf{prefix}_k(\beta)) = a_{k+1}$ and $\mathsf{e}(\mathsf{prefix}_k(\alpha)) = b_{k+1}$ for any $k \geq 0$, where ϕ is the agent function. We denote this interaction trace by $\mathsf{trace}(\phi, \mathsf{e})$.

If we are given just an action sequence α generated by some agent function ϕ, we write $\mathsf{trace}(\alpha, \mathsf{e}) = \mathsf{trace}(\phi, \mathsf{e})$. If we are given just a percept sequence β generated by e, we write $\mathsf{trace}(\phi, \beta) = \mathsf{trace}(\phi, \mathsf{e})$.

When an deterministic agent function ϕ interacts with a nondeterministic environment $\bar{\mathsf{e}}$, the set of all possible interaction traces is denoted by

$$\mathsf{traces}(\phi, \bar{\mathsf{e}}) = \left\{\mathsf{trace}(\phi, \mathsf{e}) : \mathsf{e} \in \mathbb{E}^{\bar{\mathsf{e}}}\right\}.$$

Since a probabilistic environment $\hat{\mathsf{e}}$ is also a nondeterministic environment, we denote the set of all possible interaction traces between a deterministic agent function ϕ and $\hat{\mathsf{e}}$ by

$$\mathsf{traces}(\phi, \hat{\mathsf{e}}) = \left\{\mathsf{trace}(\phi, \mathsf{e}) : \mathsf{e} \in \mathbb{E}^{\hat{\mathsf{e}}}\right\}.$$

3.2 Task-Completion Problems

Agent-based problems are often defined using states. For *goal-based agents*, the objective is to reach a goal state by interacting with the environment. For example, in classical planning problems, an agent must, by means of its actions, take the environment to a state in some set S of goal states. Likewise, problems with *extended goals* require an agent to take the environment through a sequence in some set $\mathcal{S}$ of *sequences* of states.

We generally call these problems *task-completion problems*.

One major drawback of the state-based problem formulation is that in some problems it is difficult, if not impossible, to define the set of states. For example, in repeated games such as the IPD, a player usually does not know the internal states of the other player. The state-based problem formulation may not be appropriate for these problems.

To avoid the reliance on the notion of states, we propose a new problem formulation called the *interaction-based* problem formulation that uses interaction traces in place of states. There is a large class of problems in which the success of an agent in solving a problem depends *solely* on how it interacts with the environment. Thus, we can formulate these problems based on how successful an interaction is, without referring the states of the world. The interaction-based problem formulations is expressive enough to formulate many kind of problems including nondeterministic or probabilistic planning problems.

3.2.1 Interaction-based Problem Formulations

In our interaction-based problem formulations, a state is replaced by the set of interaction traces that can reach the state, and goals are defined by the condition over the set of all possible interaction traces. More generally, a goal is a Boolean function over the set of all possible histories. Let $\mathcal{T}_{success}$ be the set of all possible histories that satisfies the goal. We say the interaction traces in $\mathcal{T}_{success}$ are *successful*. Mathematically, a goal is equivalence to the membership function of $\mathcal{T}_{success}$. Thus, we can also use $\mathcal{T}_{success}$ to define a goal.

For task-completion problems, we put certain restriction of the set of successful

interaction traces:

Definition 5 *A* goal *for task-completion problems is a set $\mathcal{T}_{success}$ of interaction traces such that for every $\tau \in \mathcal{T}_{success}$, τ is finite and all extensions of τ are in $\mathcal{T}_{success}$ (i.e., $\tau \circ \tau' \in \mathcal{T}_{success}$ for any $\tau' \in \mathcal{T}^*$).*

The condition that all extensions of successful interaction traces are successful is specific to task-completion problems: this condition ensures that once an agent completes a task, no further interaction with the environment could alter this accomplishment.

We define three versions of task-completion problems based on the types of the environments:

- A *deterministic task-completion problem* is a 4-tuple $\mathcal{P} = (A, B, \mathsf{e}, \mathcal{T}_{success})$.
- A *nondeterministic task-completion problem* is a 4-tuple $\bar{\mathcal{P}} = (A, B, \bar{\mathsf{e}}, \mathcal{T}_{success})$.
- A *probabilistic task-completion problem* is a 4-tuple $\hat{\mathcal{P}} = (A, B, \hat{\mathsf{e}}, \mathcal{T}_{success})$.

In the above definitions,

- A is a finite set of actions;
- B is a finite set of percepts;
- $\mathcal{T}_{success}$ is a goal;
- The environments e, $\bar{\mathsf{e}}$, and $\hat{\mathsf{e}}$ are deterministic, nondeterministic, and probabilistic, respectively; and
- e, $\bar{\mathsf{e}}$, $\hat{\mathsf{e}}$, and the membership of $\mathcal{T}_{success}$ are computable functions.

Roughly speaking, the objective of these problems is to find a deterministic agent function ϕ such that if an agent uses ϕ to interact with the environment, the interaction trace between the agent and the environment is successful. But the objective are slightly different in different versions of the problem.

In deterministic task-completion problems, it is sufficient to find an action sequence α such that $\mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success}$, without finding a deterministic agent function ϕ, since all an agent needs to succeed is to generate α. We call α a *solution* of $\mathcal{P}$.

In nondeterministic task-completion problems, the interaction between the agent and the environment can vary from time to time. As discussed in Section 3.1.2, every nondeterministic environment $\bar{\mathsf{e}}$ has an equivalent set $\mathbb{E}^{\bar{\mathsf{e}}}$ of deterministic environments, each of them is called a configuration. For each configuration there is an action sequence that can reach the goal. But unfortunately, the agent does not know which configuration is the actual one, thus it does not know which action sequence is the right one.

Due to this problem, there are several ways to define the objectives for nondeterministic task-completion problems. One objective is to require that the agent always succeeds no matter which configuration is actual. More precisely, the objective is to find a deterministic agent function ϕ such that if the agent uses ϕ to interact with $\bar{\mathsf{e}}$, *all* interaction traces generated by the agent are successful (i.e., $\mathsf{traces}(\phi, \bar{\mathsf{e}}) \subseteq \mathcal{T}_{success}$). ϕ is called a *strong solution*.

Not every nondeterministic task-completion problem has strong solutions. But if strong solutions exist, we say the nondeterministic task-completion problem is *strongly solvable*.

Strong solvability is a rather strong requirement. When there is no strong solution,

we would like to look for suboptimal solutions instead. A deterministic agent function ϕ is a *weak solution* if at least one successful interaction trace can possibly be generated when ϕ interacts with $\bar{\mathsf{e}}$. More precisely, ϕ is a weak solution for $\bar{\mathcal{P}}$ if *at least* one interaction trace generated by the agent is successful (i.e., $\mathsf{traces}(\phi, \bar{\mathsf{e}}) \cap \mathcal{T}_{success} \neq \emptyset$). $\bar{\mathcal{P}}$ is *weakly solvable* if and only if a weak solution exists.

The concepts of strong solutions and weak solutions can be directly applied to probabilistic task-completion problems. But the solvability of probabilistic task-completion problems can be more fine-grained. The *probability of success* of an agent function ϕ in $\hat{\mathcal{P}}$ is the sum of the probabilities of the configurations that ϕ succeeds in it. Mathematically, the probability of success of ϕ is

$$P(\phi) = \sum \left\{ \Delta^{\hat{\mathsf{e}}}(\mathsf{e}) : \forall \mathsf{e} \in \mathbb{E}^{\hat{\mathsf{e}}} \text{ such that } \mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success} \right\}$$

$\hat{\mathcal{P}}$ is *p-solvable*, where $0 \leq p \leq 1$, iff there is an agent function ϕ that $P(\phi) \geq p$. An *optimal agent function* ϕ^* for $\hat{\mathcal{P}}$ is one for which there is no other agent function ϕ such that $P(\phi) > P(\phi^*)$. $\hat{\mathcal{P}}$ is strongly solvable iff $P(\phi^*) = 1$; $\hat{\mathcal{P}}$ is *unsolvable* iff $P(\phi^*) = 0$; If $\hat{\mathcal{P}}$ is not unsolvable (i.e., $0 < P(\phi^*)$), $\hat{\mathcal{P}}$ is *weakly solvable*.

3.2.2 Goal Uncertainty

So far we assume the goal of a task-completion problem is fixed—the agent knows ahead of time the set $\mathcal{T}_{success}$ of successful interaction traces. But what if the agent is not sure about which interaction traces are successful? If it is the case, the agent may face the kind of dilemma as shown in Figure 3.1. In this figure, there are two deterministic environments (configurations) that has different goals, and the behavior of the environ-

ments are depicted by the state diagrams. The states in the diagrams denote the percepts generated by the environment, while the edges denote the actions the agent can choose. Actions and percepts are not made simultaneously, but interleaved with each other. First the environment generates a percept, and then the agent chooses an action. Depended on the action the agent chooses, the environment will advance to the next states according to the edges in the diagrams. The objective is to reach the goal states, which is denoted by the double circles in the figure.

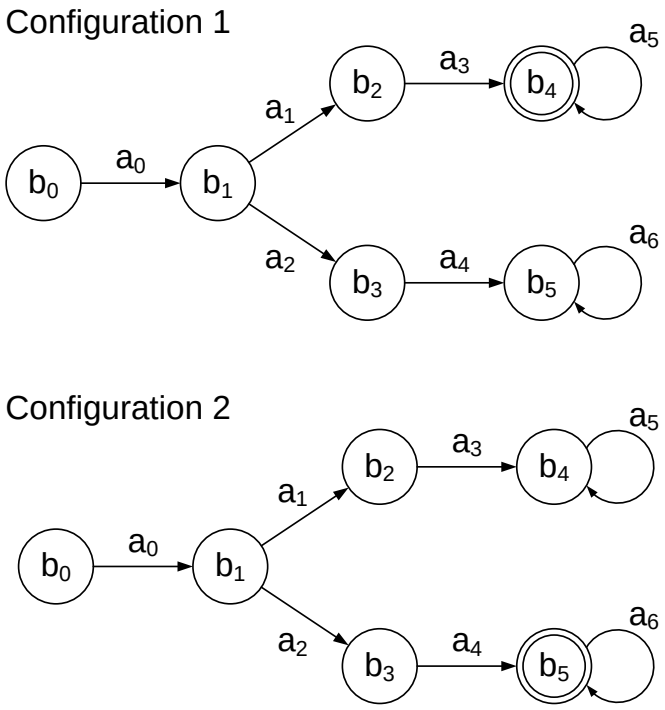

Figure 3.1: The state diagrams of two configurations with different goals. The double circles are the goal states. This figure is adapted from Figure 4.19(a) on page 124 in Russell and Norvig [57].

There are two possible goal states: b_4 and b_5. If the goal is b_4, a successful interac-

tion trace is

$$\langle \tau_1 = (\mathsf{Nil}, b_0), (a_0, \mathsf{Nil}), (\mathsf{Nil}, b_1), (a_1, \mathsf{Nil}), (\mathsf{Nil}, b_2), (a_3, \mathsf{Nil}), (\mathsf{Nil}, b_4) \rangle$$

If the goal is b_5, a successful interaction trace is

$$\langle \tau_2 = (\mathsf{Nil}, b_0), (a_0, \mathsf{Nil}), (\mathsf{Nil}, b_1), (a_2, \mathsf{Nil}), (\mathsf{Nil}, b_3), (a_4, \mathsf{Nil}), (\mathsf{Nil}, b_5) \rangle$$

But since the agent is uncertain about which goal state is the true one, the agent cannot determine which action it should choose at the fourth interaction in order to reach the goal states. This dilemma is caused by the uncertainty about the goal states.

In some problems, the success of an agent can depend on the configurations. Let us consider the example in Section 2.3.2. In this example, there are two possible opponents: TFT and TFTT. Thus, the environment has two different configurations: e_{TFT} and e_{TFTT}, and the probability of occurrence of e_{TFT} is 0.7 while that of e_{TFTT} is 0.3. Suppose we define the notion of success in such a way that when playing against TFT, the successful interaction trace has to be $\tau_3 = \langle (C, C), (C, C), \ldots \rangle$, while the successful interaction trace for playing against TFTT has to be $\tau_4 = \langle (D, C), (C, C), (D, C), (C, C), \ldots \rangle$. According to the analysis in Section 2.3.2, no agent can always succeed according this criteria.

At first glance, one possible way to define goal uncertainty is to replace the goal $\mathcal{T}_{success}$ with a set of goals or a probability distributions of a set of goals. These definitions implicitly assume that goals are independent of the random events in the environment. However, this independence assumption is not always true—the goal can depend on the environment. For example, suppose there are two possible configurations: e_1 and e_2. When e_1 is the actual configuration, the goals are both g_1 and g_2. When e_2 is the actual

configuration, the goal is g_1 only. Then the goal uncertainty in this example cannot be modeled as a set of goals.

A natural way to express the associations between goals and the environment is to consider task-completion problems with goal uncertainty as a set of task-completion problems without goal uncertainty, or a probability distribution over a set of task-completion problems.

Definition 6 *A* nondeterministic task-completion problem with goal uncertainty *is a set of nondeterministic task-completion problems, all of them have the same set of actions and percepts.*

Definition 7 *A* probabilistic task-completion problem with goal uncertainty *is a pair* (P^{gu}, Δ^{gu})*, where* P^{gu} *is a set of probabilistic task-completion problems, all of them have the same set of actions and percepts, and* Δ^{gu} *is a probabilistic distribution over* P^{gu}*.*

For example, let e_1 and e_2 be the deterministic environments in Figure 3.1. Let $A = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6\}$, $B = \{b_0, b_1, b_2, b_3, b_4, b_5\}$, $\mathcal{T}^1_{success} = \{\tau_1\}$ and $\mathcal{T}^2_{success} = \{\tau_2\}$. Then the problem in Figure 3.1 can be defined as $\{\bar{\mathcal{P}}_1, \bar{\mathcal{P}}_2\}$, where $\bar{\mathcal{P}}_i = (A, B, \mathsf{e}_i, \mathcal{T}^i_{success})$ for $i = 1$ or 2. Similarly, the problem in Section 2.3.2 can be defined as a probability distribution Δ over $\{\hat{\mathcal{P}}_3, \hat{\mathcal{P}}_4\}$, where $\hat{\mathcal{P}}_3 = \{\{C, D\}, \{C, D\}, (\{\mathsf{e}_{TFT}\}, \Delta_{uniform}), \{\tau_3\}\}$, $\hat{\mathcal{P}}_4 = \{\{C, D\}, \{C, D\}, (\{\mathsf{e}_{TFTT}\}, \Delta_{uniform}), \{\tau_4\}\}$, $\Delta(\hat{\mathcal{P}}_3) = 0.7$, $\Delta(\hat{\mathcal{P}}_4) = 0.3$, and $\Delta_{uniform}$ is a uniform probability distribution.

3.2.3 Simplifying the Problem Definitions

One way to simplify the problem definitions is to combine the nondeterministic choices or the random variables in the problem definitions together to form a single nondeterministic choice or random variable.

Let $\{(A, B, \bar{\mathsf{e}}_i, \mathcal{T}^i_{success})\}_{i=1..n}$ be a nondeterministic task-completion problem with goal uncertainty. Let $\mathbb{E}$ is the union of the equivalent sets of configurations of $\bar{\mathsf{e}}_i$ (i.e., $\mathbb{E} = \bigcup_{i=1..n} \mathbb{E}^{\bar{\mathsf{e}}_i}$). If e is a common configuration of n nondeterministic environments, there will be n copies of e in $\mathbb{E}$, one for each nondeterministic environment containing e. [3] We define $\mathcal{T}_{success}$ as a mapping from $\mathbb{E}$ to $\mathcal{T}^*$, such that $\mathcal{T}_{success}(\mathsf{e}) = \mathcal{T}^i_{success}$ if e is a configuration for $\bar{\mathsf{e}}_i$. We say $\mathcal{T}_{success}(\mathsf{e})$ is the goal of the configuration e. The nondeterministic task-completion problem with goal uncertainty can be defined as a 4-tuple $(A, B, \mathbb{E}, \mathcal{T}_{success})$. According to this definition, a nondeterministic task-completion problem without goal uncertainty is a special case of nondeterministic task-completion problem with goal uncertainty in which $\mathcal{T}_{success}$ is a constant function.

Definition 8 *A* nondeterministic task-completion problem $\bar{\mathcal{P}}$ *(with or without goal uncertainty) is a* 4*-tuple* $(A, B, \mathbb{E}, \mathcal{T}_{success})$*, where* A *is a set of actions,* B *is a set of percepts,* $\mathbb{E}$ *is a set of configurations (i.e., deterministic environments), and* $\mathcal{T}_{success}$ *is a mapping from* $\mathbb{E}$ *to* $\mathcal{T}^*$*.*

Similarly, let (P^{gu}, Δ^{gu}) be a probabilistic task-completion problem with goal uncertainty, where (1) $P^{gu} = \{(A, B, \hat{\mathsf{e}}_i, \mathcal{T}^i_{success})\}_{i=1..n}$, (2) Δ^{gu} is a probability distribution over P^{gu}, and (3) $\hat{\mathsf{e}}_i = (\bar{\mathsf{e}}_i, \Delta_i)$ for $1 \leq i \leq n$. Then the problem can be defined as 5-tuple

[3] This implies that a configuration in $\mathbb{E}$ can have more than one set of successful interaction traces.

$(A, B, \mathbb{E}, \Delta, \mathcal{T}_{success})$, where

- $\mathbb{E}$ is the union of the equivalent sets of configurations of $\bar{\mathsf{e}}_i$ (i.e., $\mathbb{E} = \bigcup_{i=1..n} \mathbb{E}^{\bar{\mathsf{e}}_i}$);
- Δ is a probability distribution over $\mathbb{E}$, such that $\Delta(\mathsf{e}) = \Delta^{gu}(\hat{\mathcal{P}}_i) \times \Delta_i(\mathsf{e})$ where $\hat{\mathcal{P}}_i = (A, B, \hat{\mathsf{e}}_i, \mathcal{T}^i_{success})$ and $\mathsf{e} \in \mathbb{E}^{\hat{\mathsf{e}}_i}$; and
- $\mathcal{T}_{success}$ as a mapping from $\mathbb{E}$ to $\mathcal{T}^*$, such that $\mathcal{T}_{success}(\mathsf{e}) = \mathcal{T}^i_{success}$ if e is a configuration for $\hat{\mathsf{e}}_i$.

According to this definition, a probabilistic task-completion problem without goal uncertainty is a special case of probabilistic task-completion problem with goal uncertainty in which $\mathcal{T}_{success}$ is a constant function.

Definition 9 *A* probabilistic task-completion problem $\hat{\mathcal{P}}$ *(with or without goal uncertainty) is a* 5*-tuple* $(A, B, \mathbb{E}, \Delta, \mathcal{T}_{success})$*, where* A *is a set of actions,* B *is a set of percepts,* $\mathbb{E}$ *is a set of configurations (i.e., deterministic environments),* Δ *is a probabilistic distribution over* $\mathbb{E}$*, and* $\mathcal{T}_{success}$ *is a mapping from* $\mathbb{E}$ *to* $\mathcal{T}^*$*.*

With the help of these simplified definitions, we can easily adapt the concept of strong solutions and weak solutions for task-completion problems without goal uncertainty to task-completion problems with goal uncertainty. A *strong solution* for a nondeterministic task-completion problem $\bar{\mathcal{P}}$ (with or without goal uncertainty) is a deterministic agent function ϕ whose interaction traces generated by interacting with the environment are always successful (i.e., $\mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})$ *for all* $\mathsf{e} \in \mathbb{E}$). A *weak solution* for $\bar{\mathcal{P}}$ is ϕ such that some of the interaction traces of ϕ are successful (i.e., there exists $\mathsf{e} \in \mathbb{E}$ such that $\mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})$). We say the agent using ϕ *strongly succeeds*

in $\bar{\mathcal{P}}$ if ϕ is a strong solution. Likewise, the agent *weakly succeeds* in $\bar{\mathcal{P}}$ if ϕ is a weak solution.

The *probability of success* of an agent function ϕ in a probabilistic task-completion problem $\hat{\mathcal{P}}$ is the sum of the probabilities of the configurations that ϕ succeeds in it:

$$P(\phi) = \sum \{\Delta(\mathsf{e}) : \forall \mathsf{e} \in \mathbb{E} \text{ such that } \mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})\}$$

$\hat{\mathcal{P}}$ is *p-solvable*, where $0 \leq p \leq 1$, iff there is an agent function ϕ that $P(\phi) \geq p$. An *optimal agent function* ϕ^* for $\hat{\mathcal{P}}$ is one for which there is no other agent function ϕ such that $P(\phi) > P(\phi^*)$. $\hat{\mathcal{P}}$ is strongly solvable iff $P(\phi^*) = 1$; $\hat{\mathcal{P}}$ is *unsolvable* iff $P(\phi^*) = 0$; If $\hat{\mathcal{P}}$ is not unsolvable (i.e., $0 < P(\phi^*)$), $\hat{\mathcal{P}}$ is *weakly solvable*.

3.3 Strong Solvability

This section gives necessary and sufficient conditions for strong solvability of nondeterministic task-completion problems. The conditions are also applicable to probabilistic task-completion problems, since every probabilistic task-completion problem has a nondeterministic task-completion problem whose strong solutions are the strong solutions for the probabilistic task-completion problem.

First, we look at the case where a nondeterministic task-completion problem $\bar{\mathcal{P}}$ contains exactly two configurations. For this case, Theorem 4 gives necessary and sufficient conditions for strong solvability. If the conditions in Theorem 4 are not satisfied, every agent for $\bar{\mathcal{P}}$ will inevitably face the kind of dilemma shown in Figure 3.1, hence will fail in at least one of the configurations.

Definition 10 *The* longest common prefix *of two (finite or infinite) sequences* $\mu_1 =$

$\langle m_1, m_2, \ldots\rangle$ *and* $\mu_2 = \langle m'_1, m'_2, \ldots\rangle$ *is* $\mathsf{lcp}(\mu_1, \mu_2) = \langle m_1, m_2, \ldots, m_k\rangle$*, where* $m_i = m'_i$ *for* $1 \leq i \leq k$*, and either (1)* $k = |s_1|$*, (2)* $k = |s_2|$*, or (3)* $m_{k+1} \neq m'_{k+1}$.

In other words, suppose k is the largest number such that $\mathsf{prefix}_k(\mu_1) = \mathsf{prefix}_k(\mu_2)$. Then the longest common prefix of μ_1 and μ_2 is

$$\mathsf{lcp}(\mu_1, \mu_2) = \mathsf{prefix}_k(\mu_1) = \mathsf{prefix}_k(\mu_2).$$

If $k < |\mu_1|$ and $k < |\mu_2|$, then $[\mu_1]_j = [\mu_2]_j$ for $1 \leq j \leq k$, and $[\mu_1]_j \neq [\mu_2]_j$ for $k < j \leq min(|\mu_1|, |\mu_2|)$.

Theorem 4 $\bar{\mathcal{P}} = \langle A, B, \mathbb{E}, \mathcal{T}_{success}\rangle$ *where* $\mathbb{E} = \{\mathsf{e}_1, \mathsf{e}_2\}$ *is strongly solvable if and only if either (1)* $\mathbb{A}_1 \cap \mathbb{A}_2 \neq \emptyset$*, where* $\mathbb{A}_1 = \mathsf{actions}(\mathcal{T}_{success}(\mathsf{e}_1))$ *and* $\mathbb{A}_2 = \mathsf{actions}(\mathcal{T}_{success}(\mathsf{e}_2))$*, or (2) there exist action sequences* $\alpha_1 \in \mathbb{A}_1$ *and* $\alpha_2 \in \mathbb{A}_2$ *such that* $|\mathsf{lcp}(\mathsf{percepts}(\mathsf{trace}(\alpha_1, \mathsf{e}_1)), \mathsf{percepts}(\mathsf{trace}(\alpha_2, \mathsf{e}_2)))| < |\mathsf{lcp}(\alpha_1, \alpha_2)|$.

Proof. If $\mathbb{A}_1 \cap \mathbb{A}_2$ is not an empty set, we can construct a strongly successful agent function ϕ for $\bar{\mathcal{P}}$ by choosing any action sequence in $\mathbb{A}_1 \cap \mathbb{A}_2$ and then constructing ϕ which generates the chosen action sequence. Otherwise, suppose there exist $\alpha_1 \in \mathbb{A}_1$ and $\alpha_2 \in \mathbb{A}_2$ such that $|\mathsf{lcp}(\mathsf{percepts}(\mathsf{trace}(\alpha_1, \mathsf{e}_1)), \mathsf{percepts}(\mathsf{trace}(\alpha_2, \mathsf{e}_2)))| < |\mathsf{lcp}(\alpha_1, \alpha_2)|$. Then $\mathsf{e}_1(\alpha') \neq \mathsf{e}_2(\alpha')$, where $\alpha' = \mathsf{prefix}_k(\mathsf{lcp}(\alpha_1, \alpha_2))$ with $k = |\mathsf{lcp}(\mathsf{percepts}(\mathsf{trace}(\alpha_1, \mathsf{e}_1)), \mathsf{percepts}(\mathsf{trace}(\alpha_2, \mathsf{e}_2)))|$. Thus, we construct a strongly successful agent function ϕ such that (1) ϕ generates α' on or before t_{k+1}, and (2) according to the percept b_{k+1} at t_{k+1}, it generates either $\mathsf{suffix}_{k+1}(\alpha_1)$ (if $b_{k+1} = \mathsf{e}_1(\alpha')$) or $\mathsf{suffix}_{k+1}(\alpha_2)$ (if $b_{k+1} = \mathsf{e}_2(\alpha')$) after t_{k+1}.

Conversely, suppose λ strongly succeeds in $\mathcal{P}$. Let $\tau_1 = (\alpha_1, \beta_1) = \mathsf{trace}(\lambda, \mathsf{e}_1) \in \mathcal{T}_{success}(\mathsf{e}_1)$ and $\tau_2 = (\alpha_2, \beta_2) = \mathsf{trace}(\lambda, \mathsf{e}_2) \in \mathcal{T}_{success}(\mathsf{e}_2)$. If $\tau_1 = \tau_2$, then

$\alpha_1 = \alpha_2 \in \mathbb{A}_1 \cap \mathbb{A}_2$. If $\tau_1 \neq \tau_2$, without loss of generality we can assume that $k = |\mathsf{lcp}(\alpha_1, \alpha_2)| \leq |\mathsf{lcp}(\beta_1, \beta_2)|$. At t_{k+1}, the agent chose a different action for each environment. But $\mathsf{prefix}_k(\mathsf{lcp}(\beta_1, \beta_2))$ is the same no matter which environment the agent is in. This contradicts the fact that since λ is deterministic, $\lambda(\mathsf{prefix}_k(\mathsf{lcp}(\beta_1, \beta_2)))$ cannot map to two different actions. Therefore, $|\mathsf{lcp}(\beta_1, \beta_2)| = |\mathsf{lcp}(\mathsf{percepts}(\mathsf{trace}(\alpha_1, \mathsf{e}_1)), \mathsf{percepts}(\mathsf{trace}(\alpha_2, \mathsf{e}_2)))| < |\mathsf{lcp}(\alpha_1, \alpha_2)|$. □

To extend necessary and sufficient conditions in Theorem 4 to nondeterministic environments that involve more than two configurations, we first of all define several relationships among interaction traces that resemble the conditions in Theorem 4.

Definition 11 *Two interaction traces τ_1 and τ_2 are* compatible *if and only if either (1)* $\mathsf{actions}(\tau_1)$ *is a prefix of* $\mathsf{actions}(\tau_2)$*, (2)* $\mathsf{actions}(\tau_2)$ *is a prefix of* $\mathsf{actions}(\tau_1)$*, or (3)* $|\mathsf{lcp}(\tau_1^A, \tau_2^A)| > |\mathsf{lcp}(\tau_1^B, \tau_2^B)|$.

Definition 12 *A set $\mathcal{T}$ of interaction traces is* compatible *if and only if every pair of interaction traces in $\mathcal{T}$ are compatible.*

It is interesting to compare Definition 11 and Definition 12 with Definition 2 and Definition 3: in Chapter 2, we assume all interaction traces are equal-length; thus an interaction trace τ_1 is a prefix of another interaction trace τ_2 if and only if $\tau_1 = \tau_2$. But here interaction traces can have different-length.

One may attempt to define the sufficiency of a set of interaction traces as in Definition 4 as follows:

Definition 13 *A set $\mathcal{T}$ of interaction traces is* $\mathbb{E}$-sufficient *if and only if for all* $\mathsf{e} \in \mathbb{E}$ *there exists $\tau \in \mathcal{T}$ such that $\tau = \mathsf{trace}(\alpha, \mathsf{e})$ for some action sequence α.*

But $\mathbb{E}$-sufficiency, together with compatibility, does not guarantee the success of an agent in task-completion problems, unless the interaction traces in $\mathcal{T}$ are successful. Thus, we extend the definition of $\mathbb{E}$-sufficiency to require that interaction traces are compatible, $\mathbb{E}$-sufficient, and successful.

Definition 14 *A set $\mathcal{T}$ of interaction traces is* $\bar{\mathcal{P}}$-successful *if and only if (1) $\mathcal{T}$ is compatible and (2) for all $\mathsf{e} \in \mathbb{E}$ there exists $\tau \in \mathcal{T}$ such that $\tau \in \mathcal{T}_{success}(\mathsf{e})$.*

If $\mathcal{T}$ is $\bar{\mathcal{P}}$-successful, there exists a bipartite matching between $\mathbb{E}$ and $\mathcal{T}$, such that for every configuration in $\mathbb{E}$, the corresponding interaction trace according to the mapping is successful in the configuration. The bipartite matching, called an *EI-mapping* (which stands for an *environment-interaction mapping*) and denoted by ξ, is an surjection (i.e., onto) mapping from $\mathbb{E}$ to $\mathcal{T}$. We modify Definition 14 based on EI-mappings as follows:

Definition 15 *A set $\mathcal{T}$ of interaction traces is* $\bar{\mathcal{P}}$-successful *if and only if (1) $\mathcal{T}$ is compatible and (2) there exists an EI-mapping ξ such that $\xi(\mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})$ for all $\mathsf{e} \in \mathbb{E}$.*

We now extend Theorem 4 to problems that involve more than two environments.

Theorem 5 *A nondeterministic task-completion problem $\bar{\mathcal{P}} = \langle A, B, \mathbb{E}, \mathcal{T}_{success} \rangle$ is strongly solvable if and only if there exists a $\bar{\mathcal{P}}$-successful set of interaction traces.*

Proof. If $\bar{\mathcal{P}}$ is strongly solvable by an agent function ϕ, let $\mathcal{T} = \{\tau_i = \mathsf{trace}(\phi, \mathsf{e}_i) : \mathsf{e}_i \in \mathbb{E}\}$ be the set of successful interaction traces. Clearly, there is an EI-mapping $\xi : \mathbb{E} \rightarrow \mathcal{T}$ such that $\xi(\mathsf{e}_i) = \tau_i$ for $\mathsf{e}_i \in \mathbb{E}$. The remaining question is whether $\mathcal{T}$ is compatible. If $\mathcal{T}$ is not compatible, there exist $\mathsf{e}_1, \mathsf{e}_2 \in \mathbb{E}$ such that τ_1 and τ_2 are not compatible. But this contradicts to the fact that ϕ is deterministic. Thus, $\mathcal{T}$ is $\mathbb{E}$-compatible.

Conversely, if there exists a $\bar{\mathcal{P}}$-successful set $\mathcal{T}$ of interaction traces, we claim that an agent using the deterministic agent function in Figure 3.2 with $\mathcal{T}$ as the input will strongly succeed in $\bar{\mathcal{P}}$. The agent function is called a *composite agent function* and is denoted by $\text{CA}(\mathcal{T})$. To see why $\text{CA}(\mathcal{T})$ is a strong solution for $\bar{\mathcal{P}}$, we need to check whether $\mathsf{trace}(\text{CA}(\mathcal{T}), \mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})$ for all $\mathsf{e} \in \mathbb{E}$. In other words, we need to check the composite agent function always terminates with success (Line 14) when interacting with any configuration in $\mathbb{E}$.

Without loss of generality, let $\mathsf{e}^* \in \mathbb{E}$ be the actual configuration. By induction on i, we prove that $\text{CA}(\mathcal{T})$ terminates with success when interacting with e^*. Since all interaction traces in $\mathcal{T}$ are finite (because they are all successful), if $\text{CA}(\mathcal{T})$ does not exist with failure at Line 3 or Line 6 at the i'th iteration, then $\text{CA}(\mathcal{T})$ will terminate with success at Line 14. Thus, all we need is to prove by induction on i that $\text{CA}(\mathcal{T})$ does not exist with failure at the i'th iteration. More precisely, we prove (1) $|A_i| = 1$ at Line 6, and (2) there exist $\tau \in \mathcal{T}_i$ such that τ can be generated by e^* (i.e., $\mathcal{T}_i \neq \emptyset$ at Line 3), for $1 \leq i \leq n$.

First, let us consider $i = 1$. For any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}$, the first actions in τ_1^A and τ_2^A must be the same since $\mathcal{T}$ is compatible. Therefore, $|A_1| = 1$. Since $\mathcal{T}$ is $\mathbb{E}$-sufficient and $\mathbb{E}$ is nonempty, it follows that there is at least one interaction trace $\tau \in \mathcal{T}$ that can be generated by e^*. Thus, $\mathcal{T}_1 \neq \emptyset$.

Now consider $i = k + 1$. Suppose $|A_k| = 1$ and there exist $\tau \in \mathcal{T}_k$ that can be generated by e^*. First, since all traces in $\mathcal{T}_k$ have the same prefix up to the $(k-1)$'th iteration and e^* is a deterministic environment, e^* will return a unique percept b_k at Line 8, which is the k'th action of τ (otherwise τ is not generated by ϕ). In addition,

by Definition 2, for any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}_k$, the first k actions in τ_1^A and τ_2^A must be the same. Thus, the k'th action of τ^A must be a_k, the action generated at Line 8. Therefore, (a_k, b_k) is the i'th interaction of τ, and τ will not be removed from $\mathcal{T}'_k$ at Line 11. Hence $\tau \in \mathcal{T}_{k+1}$ since $\mathcal{T}_{k+1} = \mathcal{T}'_k$ at Line 12. Second, $|A_{k+1}| \geq 1$ because $|\mathcal{T}_{k+1}| \geq 1$. Third, at the start of each iteration $k+1$, all interaction traces in $\mathcal{T}_{k+1}$ have the same prefix up to the k'th iteration (any traces without this prefix were removed at Line 11 on a previous iteration). By Definition 2, for any two interaction traces $\tau_1, \tau_2 \in \mathcal{T}_{k+1}$, the first $k+1$ actions in τ_1^A and τ_2^A must be the same; and consequently $|A_{k+1}| \leq 1$. Therefore, $|A_{k+1}| = 1$. By induction, $|A_i| = 1$ and $\mathcal{T}_i \neq \emptyset$, for $1 \leq i \leq n$. □

The significance of Theorem 4 is that it implies that if we can find a set of mutually compatible interaction sequences that relate to the set $\mathbb{E}$ of configurations by an EI-mapping, then we know $\bar{\mathcal{P}}$ is strongly solvable without needing to know any information about the structure of the environments in the configurations in $\mathbb{E}$. Hence, we can ignore a large part of the structures of the environments when we constructs an agent function against the environment. Theorem 4 also suggests that one way to decide whether $\bar{\mathcal{P}}$ is not strongly solvable is to find two configurations $\mathsf{e}_1, \mathsf{e}_2 \in \mathbb{E}$ such that no successful interaction sequences of these environments are $(\mathsf{e}_1, \mathsf{e}_2)$-compatible.

The difference between the composite agent function in Figure 3.2 and the composite agent in Figure 2.2 is that this procedure continues until the agent succeeds, while the procedure in Figure 2.2 continues the end of the game. If the input $\mathcal{T}$ is compatible, the procedure would never fail at Line 6. If $\mathcal{T}$ is $\mathbb{E}$-compatible, the procedure would never fail at Line 3. If $\mathcal{T}$ is $\bar{\mathcal{P}}$-successful, the procedure will terminate and succeed according

Agent CA($\mathcal{T}$) /* *a composite agent synthesized from* $\mathcal{T}$ */

1. $i := 1$; $\mathcal{T}_i := \mathcal{T}$
2. Loop until the agent succeeds
3. If $\mathcal{T}_i = \emptyset$, then exit with failure because $\mathcal{T}$ is insufficient
4. If $\mathcal{T}_i \neq \emptyset$, then
5. $A_i := \{a : (\exists\, \tau \in \mathcal{T}_i)\ a$ is the i'th action of $\tau^A\}$
6. If $|A_i| \neq 1$, then exit with failure because $\mathcal{T}$ is incompatible
7. If $|A_i| = 1$, then let a_i be the action in A_i
8. Output a_i and receive a percept b_i
9. $\mathcal{T}_i' := \mathcal{T}_i$
10. For each $\tau \in \mathcal{T}_i$,
11. If the i'th interaction in τ isn't (a_i, b_i), remove τ from $\mathcal{T}_i'$
12. $\mathcal{T}_{i+1} := \mathcal{T}_i'$; $i := i + 1$
13. End loop
14. Terminate with success

Figure 3.2: The pseudocode of a composite agent function (also known as a composite strategy) for task-completion problems (with or without goal uncertainty).

to Theorem 4.

The running time of the procedure depends on the size of $\mathcal{T}$. We assume every interaction trace in $\mathcal{T}$ has a finite length. The number of iterations in the main loop of the procedure is at most $K = \max\{|\tau| : \tau \in \mathcal{T}\}$. In each iteration, there is one enumeration of $\mathcal{T}_k$, which takes at most $|\mathcal{T}|$ computation. Each enumeration involves a look up of the i'th entry of τ, which we assume they take a constant time. In summary, the running time of the CIT-agent procedure in Figure 3.2 is $O(K|\mathcal{T}|)$, where $K = \max\{|\tau| : \tau \in \mathcal{T}\}$. In practice, the efficiency of the procedure depends on the data structure (i.e., the prefix tree) we used to store $\mathcal{T}$.

3.4 p-Solvability and Optimal Solutions

If a task-completion problem is not strongly solvable, we opt for weak solutions instead. Let us consider a probabilistic task-completion problem $\hat{\mathcal{P}} = (A, B, \mathbb{E}, \Delta, \mathcal{T}_{success})$. The probability of success of an agent function ϕ in $\hat{\mathcal{P}}$ is

$$P(\phi) = \sum \{\Delta(\mathsf{e}) : \forall \mathsf{e} \in \mathbb{E} \text{ such that } \mathsf{trace}(\phi, \mathsf{e}) \in \mathcal{T}_{success}(\mathsf{e})\}.$$

If $\hat{\mathcal{P}}$ is p-solvable, there is an agent function ϕ such that $P(\phi) \geq p$. If $\hat{\mathcal{P}}$ is not strongly solvable, $\hat{\mathcal{P}}$ is not 1.0-solvable. But $\hat{\mathcal{P}}$ may still be weakly solvable. Our objective is to find an optimal agent function ϕ^* such that there is no other agent function ϕ such that $P(\phi) > P(\phi^*)$.

One might wonder whether a probabilistic agent function can do better than the best deterministic agent function. The answer is no. For every probabilistic agent, there is a deterministic agent function that has an equal or higher probability of success:

Theorem 6 *For any probabilistic agent function $\hat{\phi}$, there exists a deterministic agent function ϕ such that $P(\phi) \geq P(\hat{\phi})$ in any probabilistic task-completion problem $\hat{\mathcal{P}} = (A, B, \mathbb{E}, \Delta, \mathcal{T}_{success})$.*

Proof. Let $\hat{\phi}$ be $(\Lambda^{\hat{\phi}}, \Delta^{\hat{\phi}})$, where $\Lambda^{\hat{\phi}}$ is the set of deterministic agent functions equivalent to $\hat{\phi}$ and $\Delta^{\hat{\phi}}$ is the probability distribution over $\Lambda^{\hat{\phi}}$. Let $\phi_{max} = \arg\max\{P(\phi) : \phi \in \Lambda^{\hat{\phi}}\}$. Then

$$\begin{aligned} P(\hat{\phi}) &= \sum_{\phi \in \Lambda^{\hat{\phi}}} \left\{ \Delta^{\hat{\phi}}(\phi) \times P(\phi) \right\} \\ &\leq \sum_{\phi \in \Lambda^{\hat{\phi}}} \left\{ \Delta^{\hat{\phi}}(\phi) \times P(\phi_{max}) \right\} \\ &= P(\phi_{max}) \times \sum_{\phi \in \Lambda^{\hat{\phi}}} \left\{ \Delta^{\hat{\phi}}(\phi) \right\} \\ &= P(\phi_{max}) \end{aligned}$$

Thus, the probability of success of the deterministic agent function ϕ_{max} is equal to or larger than that of $\hat{\phi}$. □

Theorem 6 means that at least one of the optimal agents for $\hat{\mathcal{P}}$ is deterministic. Therefore, we want to find one of the optimal agents, it is sufficient to look at deterministic agents.

Moreover, among all optimal deterministic agent functions, it is sufficient to consider just composite agent functions, because every deterministic agent function has a "companion" composite agent function that has the same behavior:

Theorem 7 *For every deterministic agent function ϕ there is a composite agent function $\phi' = CA(\mathcal{T})$ for some set $\mathcal{T}$ of interaction traces such that $\phi(\beta) = \phi'(\beta)$ for any percept sequence β.*

Proof. Without loss of generality, let $\bar{\mathsf{e}}$ be a nondeterministic environment. Let $\mathcal{T} = \{\mathsf{trace}(\phi, \mathsf{e}_i) : \mathsf{e}_i \in \mathbb{E}^{\bar{\mathsf{e}}}\}$ be the set of all possible interaction traces that can be generated by ϕ. $\mathcal{T}$ is compatible set of interaction traces because ϕ is deterministic. Therefore the composite agent function $\phi' = \mathrm{CA}(\mathcal{T})$ would not produce any error message. Moreover, both ϕ' and λ generate the same sequence of actions no matter which configuration (among $\mathbb{E}^{\bar{\mathsf{e}}}$) is actual. □

In short, for any probabilistic task-completion problem there exists at least one composite agent function that is optimal.

3.5 Solving Weakly Solvable Problems

The results in the last section, together with Theorem 5, suggest an algorithm to construct an optimal composite agent function for a (strongly or weakly solvable) problem $\hat{\mathcal{P}}$, given a set of interaction traces $\mathcal{T}$. If $\mathcal{T}$ is exhaustive then the agent will have the highest possible probability of success, but it may be close to this probability even when $\mathcal{T}$ is not exhaustive. We call this algorithm the *CIT-search algorithm*, where CIT stands for "Compatible Interaction Traces". The major difference between CIT-search and the CIT algorithm in Section 2 is that the CIT algorithm in Section 2 is not solving task-completion problems, and if we translate the iterated games we studied in Section 2 into task-completion problems, the CIT algorithm will find the strongly solvable solutions only. But the CIT-search algorithm we present in this section can also find weakly solvable solutions.

In this section, we present how the CIT-search algorithm works, and an analysis of

the CIT-search algorithm. In next section, we present the experimental results to demonstrate its performance.

An optimal deterministic agent function ϕ^* for $\hat{\mathcal{P}}$ is one that has the maximum probability of success. To see what this means, let $\Omega_{strong} \subseteq 2^{\mathbb{E}}$ be the set of all strongly solvable sets of configurations. Here, a set $\mathbb{E}'$ of environments is strongly solvable if the subproblem $(A, B, \mathbb{E}', \mathcal{T}_{success})$ of $\hat{\mathcal{P}}$ is strongly solvable. Then ϕ^* is strongly successful on some set $\mathbb{E}_{max} \in \Omega_{strong}$ such that $\sum\{\Delta(\mathsf{e}) : \mathsf{e} \in \mathbb{E}_{max}\} \geq \sum\{\Delta(\mathsf{e}) : \mathsf{e} \in \mathbb{E}'\}$ for every $\mathbb{E}' \in \Omega_{strong}$. This analysis suggests a two-step procedure to construct ϕ^*: (1) identify a set $\mathbb{E}_{max} \subseteq \mathbb{E}$ that maximizes the weighted sum $\sum\{\Delta(\mathsf{e}) : \mathsf{e} \in \mathbb{E}_{max}\}$ subject to the constraint that $(A, B, \mathbb{E}', \mathcal{T}_{success})$ is still strongly solvable, and (2) find an agent function that strongly succeeds on $\mathbb{E}_{max}$.

The CIT-search algorithm handle these two steps at the same time. Suppose we have a data base $\mathcal{T}$ of interaction traces, where each interaction trace is indexed by the environment in which they are successful. Mathematically, let $\mathbb{E}_{case} = \{\mathsf{e}_1, \mathsf{e}_2, \ldots, \mathsf{e}_m\} \subseteq \mathbb{E}$, and $C = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$ be a collection of interaction traces, such that $\mathcal{T}_i \subseteq \mathcal{T}$ is a set of successful interaction traces for e_i, for $1 \leq i \leq m$. Then we can search for (1) a subset $\mathbb{E}'$ of $\mathbb{E}_{case}$ and (2) a set $\mathcal{T}' = \{\tau_1, \tau_2, \ldots, \tau_{|\mathbb{E}'|}\}$ of interaction traces such that $\tau_i \in \mathcal{T}_i$ for each $\mathsf{e}_i \in \mathbb{E}'$ and for any $\mathsf{e}_i, \mathsf{e}_j \in \mathbb{E}'$, τ_i and τ_j are compatible. If we can find $\mathbb{E}'$ and $\{\tau_1, \tau_2, \ldots, \tau_{|\mathbb{E}'|}\}$ that satisfy these conditions, then we can show that $\mathbb{E}'$ is strongly solvable, according to Theorem 5. Furthermore, the composite agent function $\text{CA}(\mathcal{T}')$ will strongly succeed in the subproblem $(A, B, \mathbb{E}', \mathcal{T}_{success})$. If $\mathbb{E}'$ maximizes $\sum\{\Delta(\mathsf{e}) : \mathsf{e} \in \mathbb{E}'\}$, then the composite agent function is an optimal solution.

The CIT-search algorithm is a procedure that does a branch-and-bound search to

construct a CIT-agent whose success probability is p or greater. The procedure is called CIT-search and the pseudo-code of the procedure is shown in Figure 3.3. The initial inputs of CIT-search are as follows. Each $\mathcal{T}_i$ is a set of successful interaction traces for a configuration e_i whose probability is $p_i = \bar{\mathsf{e}}$. ξ, η, and η^+ are for CIT-search's internal bookkeeping, and their initial values should be $\xi = \emptyset$, $\eta = 0$, and $\eta^+ = 1$.

CIT-search automatically creates a composite agent function that has either (i) a success probability of p or greater, or (ii) the highest probability of success ($P(\lambda) = \eta^*$) among all combinations of the given interaction traces. Here is how it works: in this procedure, $\mathbb{E}'$ and $\mathcal{T}'$ are denoted by a EI-mapping $\xi : \mathbb{E}' \rightarrow \mathcal{T}'$, such that $\xi(\mathsf{e}_i) = \tau_i$ for each $\tau_i \in \mathcal{T}'$ that succeeds in $\mathsf{e}_i \in \mathbb{E}'$. CIT-search goes through the sets of interaction traces $C = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_m\}$ one by one. For each $\mathcal{T}_k$, it looks for a successful interaction trace $\tau_k \in \mathcal{T}_k$ that is compatible with *every* interaction trace in $\mathsf{range}(\xi)$. If such an interaction trace exists, CIT-search sets $\xi(\mathsf{e}) = \tau$. Otherwise, it skips $\mathcal{T}_k$ and considers the next set of interaction trace $\mathcal{T}_{k-1}$. Once it has looked at all sets of interaction traces in C, it checks to see if $\mathrm{CA}(\mathsf{range}(\xi))$ can succeed with a probability $\eta \geq p$, where $\eta = \sum\{p_i : e_i \in \mathsf{dom}(\xi)\}$, and $p_i = \Delta(e_i)$. If the answer is yes, then CIT-search terminates and returns the composite agent function $\mathrm{CA}(\mathsf{range}(\xi))$. Otherwise, it backtracks and searches for another mutually compatible set of interaction traces. The procedure maintains the EI-mapping ξ^* with the highest probability η^* that has seemed so far, and this can be used with a upper-bound η^+ of η to prune unpromising search branches.

The way the procedure constructs ξ guarantees that $\mathcal{T}'$ is a compatible set of interaction traces. Moreover, when $\mathbb{E}_{case} = \mathbb{E}$, $\mathcal{T}_i = \mathcal{T}_{success}(\mathsf{e}_i)$ for all $\mathsf{e}_i \in \mathbb{E}_{case}$, and

Global variables: $\xi^* := \emptyset$; $\eta^* := 0$ *// the best EI-mapping*

Procedure CIT-search$(\{\mathcal{T}_1, ..., \mathcal{T}_k\}, \{p_1, ..., p_k\}, p, \xi, \eta, \eta^+)$

1. If $\eta^+ < \eta^*$, then return CA(range(ξ^*)) *// stop unpromising search*
2. If $k = 0$ and $\eta > \eta^*$, then
3. $\xi^* := \xi$; $\eta^* := \eta$; If $\eta \geq p$, then terminate and return CA(range(ξ)).
4. If $k > 0$, then
5. $C' := \{\mathcal{T}_1, ..., \mathcal{T}_{k-1}\}$; $\mathbb{P}' := \{p_1, ..., p_{k-1}\}$
6. For each $\tau \in \mathcal{T}_k$
7. If τ is compatible with every $\tau' \in$ range(ξ), then
8. CIT-search$(C', \mathbb{P}', p, \xi \cup \{(\mathsf{e} \rightarrow \tau)\}, \eta + p_k, \eta^+)$
9. CIT-search$(\{\mathcal{T}_1, ..., \mathcal{T}_k\}, \{p_1, ..., p_k\}, p, \xi, \eta, (\eta^+ - p_k))$

Figure 3.3: The Compatible Interaction Traces Search algorithm (CIT-search).

$p = 1$, CIT-search will correctly return an optimal composite agent function, because it exhaustively searches all possible EI-mappings. More generally, if we begin with a small case base and gradually add additional cases, then the probability of success of the agent returned by CIT-search will also increase and will approach the optimal probability:

Theorem 8 *Let $n = |\mathbb{E}_{case}|$ and $m_i = |\mathcal{T}_i|$ for any $\mathsf{e}_i \in \mathbb{E}_{case}$. As $n \rightarrow |\mathbb{E}|$ and $m_i \rightarrow |\mathcal{T}_{success}(\mathsf{e}_i)|$, then $P(CA(\mathcal{T}')) \rightarrow p^*$, where p^* is the probability of success for an optimal solution, and $CA(\mathcal{T}')$ is the composite agent function returned by CIT-search with $\mathbb{E}_{case}$, C, $\{p_1, p_2, \ldots, p_m\}$, and $p = 1.0$ as inputs.*

The running time of CIT-search is $O(l^{m+1})$, where $l = \max\{|\mathcal{T}_i| : \mathcal{T}_i \in C\}$ and $m = |C|$. Thus, the running time depends on the size of the case base. Although the CIT-search procedure (Figure 3.3) of this method is computationally intractable (because

$\mathbb{E}$ and $\mathcal{T}_{success}$ can be infinite), we found that even if this procedure takes just a finite case base of successful interaction traces for a finite number of environments as inputs, it would still return an agent that performs reasonably well. In practice, we are likely to obtain the successful interaction traces for some instances of the nondeterministic environments though our past successful interaction with the environments. So we can construct such a case base of successful interaction traces. Furthermore, the probability Δ of environments is available, for example, by multiplying the probabilities of the outcomes of the actions (e.g., the domain description is written in a probabilistic STRIPS-style operators.)

If every e_i has an equal probability and $|\mathcal{T}_i| = 1$ for each e_i, the problem of finding the maximal set of compatible successful interaction traces is NP-hard (by reduction from the maximum clique problem). On the other hand, the running time is mostly determined by how easily the successful interaction traces can be combined together. In some problems, there are lots of successful interaction traces for every environment (hence both $|\mathcal{T}_{success}(\mathsf{e}_i)|$ and $|\mathcal{T}_i|$ is large), and in such cases it is relatively easy for CIT-search to find a good solution (i.e., one whose success probability is greater than the input parameter p).

3.6 Experimental Results

Now we experimentally evaluate how good the solutions CIT-search produces from a small number of interaction traces. Our test domain is a *pizza delivery domain* in which a delivery person needs to deliver a pizza to a customer's home within some time limit. Traffic congestion may hamper his/her job, but the only way for him/her to know whether a road is congested is by going onto that road.

We assume that the road map is a grid (e.g., like downtown Manhattan) of size $N \times N$, with $N + 1$ north-south roads and $N + 1$ east-west roads, each N blocks long. The pizza shop and the customer's home are at $(0, 0)$ and (N, N), respectively. Each road has a probability of 0.5 of being congested, and either all of it is congested or none of it is congested. The delivery person only needs to go one block on a road to find out whether it is congested, because it takes one time unit to traverse a block without congestion, and 5 time units with congestion. The actions are the directions the delivery person should go at each junction, and the percepts are whether congestion has occurred on the block that he/she has just traversed.

Note that there are $2^{2(N+1)}$ possible deterministic environments. The difficulty of each of them depends partly on the amount of time available to the delivery person. With $10N$ time units or more, the problem is strongly solvable; but with less than $2N$ time units it is unsolvable. For our experiments we used a time limit of $2N + 8Nd$ with $0 \leq d < 1$, in order to focus on weakly solvable problems.

For each combination of $N \in \{4, 5, 6, 7, 8\}$ and $d \in \{0.1, 0.3\}$, we did the following steps 25 times:

We created 63 distinct configurations $\mathsf{e}_1, \ldots, \mathsf{e}_{63}$ at random, each with congestion on different sets of roads, with a 50% chance of congestion for each road. For each e_i, we created a set of successful interaction traces $\mathcal{T}_i$ as follows. First, we found a path τ_i that went from $(0, 0)$ and (N, N) as quickly as possible in e_i. Then for each τ_i and each j, we created the set of all paths that followed τ_i for the first $j - 1$ steps, chose a different action step j, and followed the quickest path from the current location to (N, N). Out of all of these paths, $\mathcal{T}_i$ contained the ones whose total time was within e_i's time limit.

Then we ran CIT-search with $C_0, \ldots, C_{63}$, where $C_k = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$. Note that C_0 is empty. Since CIT-search is NP-hard, we usually could not wait for CIT-search to find the optimal agent: instead, we terminated it after $100,000$ recursive calls, and retrieve the best composite agent function it had found during that time.

In the literature on case-based problem solving, the problem solver will often revert to using a conventional search algorithm if it cannot solve the problem using just the case base. To model this, for each agent λ_k we also created a modified agent λ'_k that used a *base procedure* if the replay of interaction traces failed. Our base procedure was a simple one: it told the delivery person to move toward to destination randomly, as long as it did not increase the distance between the delivery person and the destination.

We ran each of the agents λ_k and λ'_k 1000 times on a simulator and recorded their success rates. We did this for all combinations of $N \in \{4, 5, 6, 7, 8\}$ and $d \in \{0.1, 0.3\}$, averaged the success rates, and graphed them as shown in Figure 3.4. Hence in Figure 3.4, each data point is an average of 250,000 runs.

For the agents that use the base procedure, Figure 3.4 shows that the success rate increases quickly with the number of configurations covered by CIT-search's database of interaction traces. It takes less than 16 configurations in order to attain a success rate of 0.33, and after that point there is no improvement as the number of configurations increases. We suspect the reason for convergence with a small number of environments was because of the base procedure, which has a success rate of 0.17 even without any help from the composite agent function.

For the agents that do not use the base procedure, the success rate also grows quickly with the number of environments. The agent generated from $k = 3$ performs as well as the

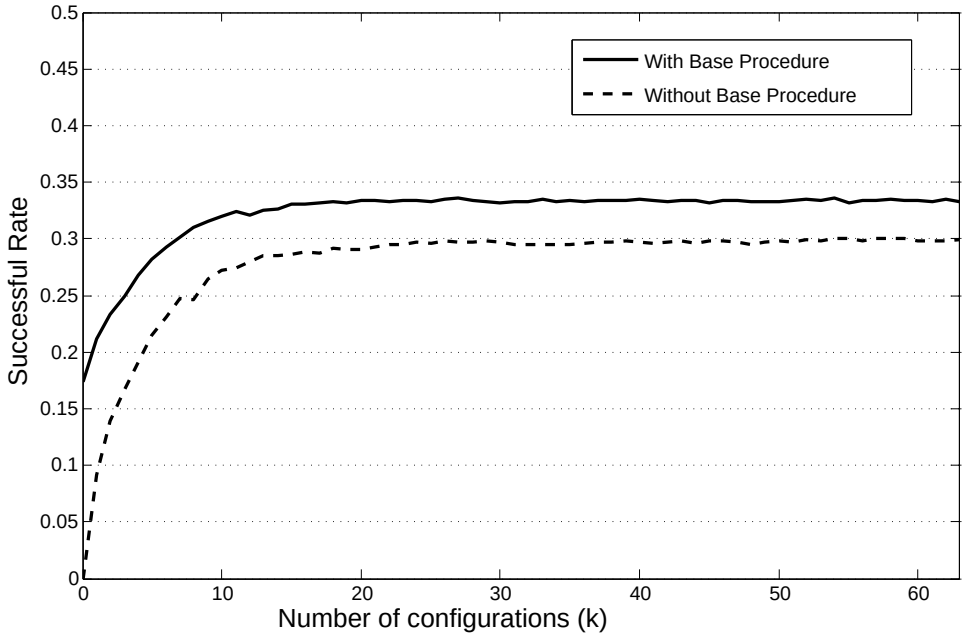

Figure 3.4: Success rates for composite agent functions constructed by running CIT-search with a database that covers k of $\bar{e}$'s 64 configurations, for $k = 1, \ldots, 64$. Each data point is an average of 250,000 runs.

base procedure with $k = 0$, and the difference in performance decreases as k increases. When all of the successful interaction traces are used without the base procedure, the performance is 94% higher than the base procedure without the interaction traces.

Just as in machine learning techniques, there is a risk of overfitting of the data. In fact, when d=0.1, we observed a small decrease of probability of success from 0.27 to 0.26 as the as the number of configurations increases from 23 to 64. However, we do not observe the same tendency when d is large. Our hypothesis to this is that for difficult problems, there is only a small number of successful interaction traces for each environment. Therefore it is possible that the CIT agent may lead the delivery person to some "remote" locations and then fail, and the base procedure cannot complete the

task starting from these locations. This also explains why we did not observe a decrease of probability of success when problems are easy. In future, we will conduct a more thorough study to investigate this issue.

3.7 Related Work

We know of no previous work on the use of case-based reasoning in stochastic environments, especially when there is goal uncertainty. A rough analogy can be made between each of $\bar{\mathsf{e}}$'s configurations e_i and a case in a traditional case-based reasoning problem such as those in [39, 63]. But in a traditional case-based reasoning problem, the domain is totally observable, hence one knows what the "new" problem is that needs solving. In our case, all we have is a probability distribution over the domain's possible behaviors.

Our method is different from existing instance-based learning techniques (such as the k-nearest neighber algorithms [44]), in that our agent classifies the environment during execution, not before execution. Unlike the decision trees in decision tree learning, our prefix tree generates actions during classification, and this influences the agent's probability of success—particularly if the environment is not safely explorable. This difficulty is similar to the exploration-exploitation dilemma that occurs in reinforcement learning [30], game playing [19], and real-time search algorithms [10]. But instead of using learning and exploration techniques, we reuse previously successful interactions to guide the exploration.

3.8 Summary

We have described a new technique to construct an agent for interacting with non-deterministic and stochastic environments with goal uncertainty. Our technique reuses previous successful interactions to maximize the agent's chance of success. Our contributions are summarized below.

We formulate the problems using interaction traces instead of states of the world. Based on this new interaction-based problem formulation of task-completion problems, we have given necessary and sufficient conditions for there to exist an agent that can always successfully accomplish a task. If there exists a set of mutually compatible successful interaction traces for each configuration of an environment, then the problem is *strongly solvable*, i.e., there exists an agent that is guaranteed to solve the problem. One class of such agent is the *composite agent function*, which take the set of compatible interaction traces to make decisions.

On problems that are not strongly solvable but where we have a database of successful interaction traces, our CIT-search algorithm constructs a agent function having the highest probability of success among all combinations of the interaction traces.

Even the database covers just a small number of configurations of an environment, our experiments show that the agents produced by CIT-search can still perform well, and that the reuse of successful interaction traces can improve the success rate of an existing problem solving strategy.

In future, we would like to conduct more extensive experiments to find out the relationship between successful interaction traces in the database and the quality of the

agent function produced by CIT-search. Moreover, we are interested to see how to speed up the CIT-search algorithm.

Chapter 4

Noise Detection in the Iterated Prisoner's Dilemma

The Iterated Prisoner's Dilemma (IPD) has become well known as an abstract model of a class of multi-agent environments in which agents accumulate payoffs that depend on how successful they are in their repeated interactions with other agents. An important variant of the IPD is the *Noisy* IPD, in which there is a small probability, called the *noise level*, that accidents will occur. In other words, the noise level is the probability of executing "cooperate" when "defect" was the intended move, or vice versa.

Accidents can cause difficulty in cooperation with others in real-life situations, and the same is true in the Noisy IPD. Strategies that do quite well in the ordinary (non-noisy) IPD may do quite badly in the Noisy IPD [5, 11, 12, 45, 46, 47]. For example, if two players both use the well-known Tit-For-Tat (TFT) strategy, then an accidental defection may cause a long series of defections by both players as each of them punishes the other for defecting.

This chapter reports on a strategy called the Derived Belief Strategy (DBS), which was the best-performing non-master-slave strategy in Category 2 (noisy environments) of the 2005 Iterated Prisoner's Dilemma competition (see Table 4.1).

Like most opponent-modeling techniques, DBS attempts to learn a model of the other player's strategy (i.e., the *opponent model*[1]) during the games. Our main innovation

[1]The term "opponent model" appears to be the most common term for a model of the other player, even

Table 4.1: Scores of the best programs in Competition 2 (IPD with Noise). The table shows each program's average score for each run and its overall average over all five runs. The competition included 165 programs, but we have listed only the top 15.

Rank	Program	Author	Average Score					
			Run1	Run2	Run3	Run4	Run5	Overall
1	BWIN	P. Vytelingum	441.7	431.7	427.1	434.8	433.5	433.8
2	IMM01	J.W. Li	424.7	414.6	414.7	409.1	407.5	414.1
3	DBSz	T.C. Au	411.7	405.0	406.5	407.7	409.2	408.0
4	DBSy	T.C. Au	411.9	407.5	407.9	407.0	405.5	408.0
5	DBSpl	T.C. Au	409.5	403.8	411.4	403.9	409.1	407.5
6	DBSx	T.C. Au	401.9	410.5	407.7	408.4	404.4	406.6
7	DBSf	T.C. Au	399.2	402.2	405.2	398.9	404.4	402.0
8	DBStft	T.C. Au	398.4	394.3	402.1	406.7	407.3	401.8
9	DBSd	T.C. Au	406.0	396.0	399.1	401.8	401.5	400.9
10	lowES-TFT_classic	M. Filzmoser	391.6	395.8	405.9	393.2	399.4	397.2
11	TFTIm	T.C. Au	399.0	398.8	395.0	396.7	395.3	397.0
12	Mod	P. Hingston	394.8	394.2	407.8	394.1	393.7	396.9
13	TFTIz	T.C. Au	397.7	396.1	390.7	392.1	400.6	395.5
14	TFTIc	T.C. Au	400.1	401.0	389.5	388.9	389.2	393.7
15	DBSe	T.C. Au	396.9	386.8	396.7	394.5	393.7	393.7

involves how to reason about noise using the opponent model.

The key idea used in DBS is something that we call *symbolic noise detection*—the use of the other player's deterministic behavior to tell whether an action has been affected by noise. More precisely, DBS builds a symbolic model of how the other player behaves, and watches for any deviation from this model. If the other player's next move is inconsistent with its past behavior, this inconsistency can be due either to noise or to a genuine change in its behavior; and DBS can often distinguish between these two cases by waiting to see whether this inconsistency persists in the next few iterations of the game.[2]

Of the nine different version of DBS that we entered into the competition, all of them placed in the top 15, and seven of them placed among top ten (see Table 4.1). Our best version, DBSz, placed third; and the two players that placed higher were both masters of master-and-slave teams.

DBS operates in a distinctly different way from the master-and-slaves strategy used by several other entrants in the competition. Each participant in the competition was allowed to submit up to 20 programs as contestants. Some participants took advantage of this to submit collections of programs that worked together in a conspiracy in which 19 of their 20 programs (the "slaves") worked to give as many points as possible to the 20th program (the "master"). DBS does not use a master-and-slaves strategy, nor does it conspire with other programs in any other way. Nonetheless, DBS remained competitive with the master-and-slaves strategies in the competition, and performed much better than the master-and-slaves strategies if the score of each master is averaged with the scores

though this player is not necessarily an "opponent" (since the IPD is not zero-sum).

[2] An iteration has also been called a period or a round by some authors.

of its slaves. Furthermore, a more extensive analysis in Section 4.7.2 shows that if each master-and-slaves team had been limited to 10 programs or less, DBS would have placed first in the competition.

4.1 Motivation and Approach

The techniques used in DBS are motivated by a British army officer's story that was quoted in [4, page 40]:

> I was having tea with A Company when we heard a lot of shouting and went out to investigate. We found our men and the Germans standing on their respective parapets. Suddenly a salvo arrived but did no damage. Naturally both sides got down and our men started swearing at the Germans, when all at once a brave German got onto his parapet and shouted out: "We are very sorry about that; we hope no one was hurt. It is not our fault. It is that damned Prussian artillery." (Rutter 1934, 29)

Such an apology was an effective way of resolving the conflict and preventing a retaliation because it told the British that the salvo was not the intention of the German infantry, but instead was an unfortunate accident that the German infantry did not expect nor desire. The reason why the apology was convincing was because it was consistent with the German infantry's past behavior. The British had was ample evidence to believe that the German infantry wanted to keep the peace just as much as the British infantry did.

More generally, an important question for conflict prevention in noisy environments is *whether a misconduct is intentional or accidental.* A deviation from the usual course

of action in a noisy environment can be explained in either way. If we form the wrong belief about which explanation is correct, our response may potentially destroy our long-term relationship with the other player. If we ground our belief on evidence accumulated before and after the incident, we should be in a better position to identify the true cause and prescribe an appropriate solution. To accomplish this, DBS uses the following key techniques:

1. **Learning about the other player's strategy.** DBS uses an induction technique to identify a set of rules that model the other player's recent behavior. The rules give the probability that the player will cooperate under different situations. As DBS learns these probabilities during the game, it identifies a set of *deterministic* rules that have either 0 or 1 as the probability of cooperation.

2. **Detecting noise**. DBS uses the above rules to detect anomalies that may be due either to noise or a genuine change in the other player's behavior. If a move is different from what the deterministic rules predict, this inconsistency triggers an *evidence collection process* that will monitor the persistence of the inconsistency in the next few iterations of the game. The purpose of the evidence-collection process is to determine whether the violation is likely to be due to noise or to a change in the other player's policy. If the inconsistency does not persist, DBS asserts that the derivation is due to noise; if the inconsistency persists, DBS assumes there is a change in the other player's behavior.

3. **Temporarily tolerating possible misbehaviors by the other player.** Until the evidence-collection process finishes, DBS assumes that the other player's behavior

is still as described by the deterministic rules. Once the evidence collection process has finished, DBS decides whether to believe the other player's behavior has changed, and updates the deterministic rules accordingly.

Since DBS emphasizes the use of deterministic behaviors to distinguish noise from the change of the other player's behavior, it works well when the other player uses a pure (i.e., deterministic) strategy or a strategy that makes decisions deterministically most of the time. Fortunately, deterministic behaviors are abundant in the Iterated Prisoner's Dilemma. Many well-known strategies, such as TFT and GRIM, are pure strategies. Some strategies such as Pavlov or Win-Stay, Lose-Shift strategy (WSLS) [35, 36, 37, 48] are not pure strategies, but a large part of their behavior is still deterministic. The reason for the prevalence of determinism is discussed by Axelrod in [3]: clarity of behavior is an important ingredient of long-term cooperation. A strategy such as TFT benefits from its clarity of behavior, because it allows other players to make credible predictions of TFT's responses to their actions. We believe the success of our strategy in the competition is because this clarity of behavior also helps us to fend off noise.

The results of the competition show that the techniques used in DBS are indeed an effective way to fend off noise and maintain cooperation in noisy environments. When DBS defers judgment about whether the other player's behavior has changed, the potential cost is that DBS may not be able to respond to a genuine change of the other player's behavior as quickly as possible, thus losing a few points by not retaliating immediately. But this delay is only temporary, and after it DBS will adapt to the new behavior. More importantly, the techniques used in DBS greatly reduce the probability that noise will

cause it to end a cooperation and fall into a mutual-defect situation. Our experience has been that it is hard to re-establish cooperation from a mutual-defection situation, so it is better avoid getting into mutual defection situations in the first place. When compared with the potential cost of ending an cooperation, the cost of temporarily tolerating some defections is worthwhile.

Temporary tolerance also benefits us in another way. In the noisy Iterated Prisoner's Dilemma, there are two types of noise: one that affects the other player's move, and the other affects our move. While our method effectively handles the first type of noise, it is the other player's job to deal with the second type of noise. Some players such as TFT are easily provoked by the second type of noise and retaliate immediately. Fortunately, if the retaliation is not a permanent one, our method will treat the retaliation in the same way as the first type of noise, thus minimizing its effect.

4.2 Iterated Prisoner's Dilemma with Noise

In the Iterated Prisoner's Dilemma, two players play a finite sequence of classical prisoner's dilemma games, whose payoff matrix is:

		Player 2	
		Cooperate	Defect
Player 1	Cooperate	(u_{CC}, u_{CC})	(u_{CD}, u_{DC})
	Defect	(u_{DC}, u_{CD})	(u_{DD}, u_{DD})

where $u_{DC} > u_{CC} > u_{DD} > u_{CD}$ and $2u_{CC} > u_{DC} + u_{CD}$. In the competition, u_{DC}, u_{CC}, u_{DD} and u_{CD} are 5, 3, 1 and 0, respectively.

At the beginning of the game, each player knows nothing about the other player and does not know how many iterations it will play. In each iteration, each player chooses either to cooperate (C) or defect (D), and their payoffs in that iteration are as shown in the payoff matrix. We call this decision a *move* or an *action*. After both players choose a move, they will each be informed of the other player's move before the next iteration begins.

If $a_k, b_k \in \{C, D\}$ are the moves of Player 1 and Player 2 in iteration k, then we say that (a_k, b_k) is the *interaction* of iteration k. If there are N iterations in a game, then the total scores for Player 1 and Player 2 are $\sum_{1 \leq k \leq N} u_{a_k b_k}$ and $\sum_{1 \leq k \leq N} u_{b_k a_k}$, respectively.

The *Noisy Iterated Prisoner's Dilemma* is a variant of the Iterated Prisoner's Dilemma in which there is a small probability that a player's moves will be mis-implemented. The probability is called the *noise level*.[3] In other words, the noise level is the probability of executing C when D was the intended move, or vice versa. The incorrect move is recorded as the player's move, and determines the interaction of the iteration.[4] Furthermore, neither player has any way of knowing whether the other player's move was executed correctly or incorrectly.[5]

For example, suppose Player 1 chooses C and Player 2 chooses D in iteration k, and noise occurs and affects the Player 1's move. Then the interaction of iteration k is (D, D). However, since both players do not know that the Player 1's move has been

[3]The noise level in the competition was 0.1.

[4]Hence, a mis-implementation is different from a misperception, which would not change the interaction of the iteration. The competition included mis-implementations but no misperceptions.

[5]As far as we know, the definitions of "mis-implementation" used in the existing literature are ambiguous about whether either of the players should know that an action has been mis-executed.

changed by noise, Player 1 and Player 2 perceive the interaction differently: for Player 1, the interaction is (C, D), but for Player 2, the interaction is (D, D). As in real life, this misunderstanding would become an obstacle in establishing and maintaining cooperation between the players.

4.3 Strategies, Policies, and Hypothesized Policies

A *history* τ of length k is the sequence of interactions of all iterations up to and including iteration k. We write $\tau = \langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$. Let $\mathcal{H} = \langle (C, C), (C, D), (D, C), (D, D) \rangle^*$ be the set of all possible histories. A *mixed strategy* is a $\psi = (\Phi, \Delta)$

A *strategy* $\psi : \mathcal{H} \to [0, 1]$ associates with each history τ a real number called the *degree of cooperation*. $\psi(\tau)$ is the probability that ψ chooses to cooperate at iteration $k + 1$, where $k = |\tau|$ is τ's length. The definition of a strategy in this chapter is in fact a shorthand way of writing a mixed strategy ψ' that maps histories into probability distributions over $A = \{C, D\}$. ψ' can be obtained from ψ by using the following equations: $\psi'(\tau, C) = \psi(\tau)$ and $\psi'(\tau, D) = 1 - \psi(\tau)$.

For examples, TFT can be considered as a function ψ_{TFT}, such that (1) $\psi_{TFT}(\tau) = 1.0$ if $k = 0$ or $a_k = C$ (where $k = |\tau|$), and (2) $\psi_{TFT}(\tau) = 0.0$ otherwise; Tit-for-Two-Tats (TFTT), which is like TFT except it defects only after it receives two consecutive defections, can be considered as a function ψ_{TFTT}, such that (1) $\psi_{TFTT}(\tau) = 0.0$ if $k \geq 2$ and $a_{k-1} = a_k = D$, and (2) $\psi_{TFTT}(\tau) = 1.0$ otherwise.

We can model a strategy as a *policy*. A *condition* $\mathsf{Cond} : \mathcal{H} \to \{\mathsf{True}, \mathsf{False}\}$ is a

mapping from histories to Boolean values. A history τ *satisfies* a condition Cond if and only if $\mathsf{Cond}(\tau) = \mathsf{True}$. A *rule* is a pair (Cond, p), which we will write as $\mathsf{Cond} \rightarrow p$, where Cond is a condition and p is a degree of cooperation (a real number in $[0, 1]$). A rule is *deterministic* if p is either 0.0 or 1.0; otherwise, the rule is *probabilistic*. A *policy schema* $\mathcal{C}$ is a set of conditions such that each history in $\mathcal{H}$ satisfies exactly one of the conditions in $\mathcal{C}$. In this chapter, we define a policy to be a set of rules whose conditions constitute a policy schema.

ψ_{TFT} can be modeled as a policy as follows: we define $\mathsf{Cond}_{a,b}$ to be a condition about the interactions of the last iteration of a history, such that $\mathsf{Cond}_{a,b}(\tau) = \mathsf{True}$ if and only if (1) $k \geq 1$, $a_k = a$ and $b_k = b$, (where $k = |\tau|$), or (2) $k = 0$ and $a = b = C$. For simplicity, we also write $\mathsf{Cond}_{a,b}$ as (a, b). The policy for ψ_{TFT} is $\pi_{TFT} = \{(C, C) \rightarrow 1.0, (C, D) \rightarrow 1.0, (D, C) \rightarrow 0.0, (D, D) \rightarrow 0.0\}$. Notice that the policy schema for π_{TFT} is $\mathcal{C} = \{(C, C), (C, D), (D, C), (D, D)\}$.

Given a policy π and a history τ, there is one and only one rule $\mathsf{Cond} \rightarrow p$ in π such that $\mathsf{Cond}(\tau) = \mathsf{True}$. We write p as $\pi(\tau)$. A policy π is *complete* for a strategy ψ if and only if $\pi(H) = \psi(H)$ for any $\tau \in \mathcal{H}$. In other words, a complete policy for a strategy is one that completely models the strategy. For instance, π_{TFT} is a complete policy for ψ_{TFT}.

Some strategies are much more complicated than TFT—a large number of rules is needed in order to completely model these strategies. If the number of iterations is small and the strategy is complicated enough, it is difficult or impossible to obtain a complete model of the other player's strategy. Therefore, an agent should not aim at obtaining a complete policy of the other player's strategy; instead, all an agent can do is to learn an

approximation of the other player's strategy during a game, using a small number of rules. In order to distinguish this approximation from the complete policies for a strategy, we call this approximation a *hypothesized policy*.

Given a policy schema $\mathcal{C}$, an agent constructs a hypothesized policy π whose policy schema is $\mathcal{C}$. The degrees of cooperation of the rules in π are estimated by a *learning function*, which computes the degrees of cooperation according to the current history. Mathematically, a learning function is a mapping from $\mathcal{C} \times \mathcal{H}$ to $[0, 1]$. For example, consider a learning function L_{avg} that computes the degrees of cooperation by averaging the number of time the next action is C when a condition Cond_i holds in the current history τ:

$$L_{avg}(\mathsf{Cond}_i, \tau) = \frac{|\{k : (0 \leq k < |\tau|) \wedge (\mathsf{Cond}_i(\mathsf{prefix}_k(\tau)) = \mathsf{True}) \wedge ([\tau^B]_{k+1} = C)\}|}{|\{k : (0 \leq k < |\tau|) \wedge (\mathsf{Cond}_i(\mathsf{prefix}_k(\tau)) = \mathsf{True})\}|} \tag{4.1}$$

where τ^B is the sequence of actions of the agent we are trying to model, and $[\tau^B]_{k+1}$ is the $(k+1)$'th action in the sequence. If the denominator of $L_{avg}(\mathsf{Cond}_i, \tau)$ is zero, $L_{avg}(\mathsf{Cond}_i, \tau) = 0$. L_{avg} is undefined if $|\tau| = 0$.

Suppose the other player's strategy is ψ_{TFTT}, the given policy schema is $\mathcal{C} = \{(C, C), (C, D), (D, C), (D, D)\}$, and the current history is $\tau = \{(C, C), (D, C), (C, C), (D, C), (D, C), (D, D), (C, D), (C, C)\}$. Then the hypothesized policy is $\pi = \{(C, C) \rightarrow 1.0, (C, D) \rightarrow 1.0, (D, C) \rightarrow 0.66, (D, D) \rightarrow 0.0\}$. Notice that the rule $(D, C) \rightarrow 0.66$ does not accurately model ψ_{TFTT}; this probabilistic rule is just an approximation of what ψ_{TFTT} does when the condition (D, C) holds. In fact, this approximation is inaccurate as long as the policy schema contains (D, C)—

there is no complete policy for ψ_{TFTT} whose policy schema contains (D, C). If we want to model ψ_{TFTT} correctly, we need a different policy schema that allows us to specify more complicated rules.

4.3.1 Discussion

When we use a hypothesized policy to model a strategy, the difference between the hypothesized policy and the strategy is due to the choice of the policy schema and the learning function. But we usually do not know how to choose a good policy schema and a good learning function. In the literature, the difficulty is choosing a right model is called *model uncertainty*, and the problem is called the problem of *model selection*. If we consider the difference between the hypothesized policy and the strategy as an error, the error is largely due to model uncertainty.

We interpret a hypothesized policy as a belief of what the other player will do in the next few iterations in response to our next few moves. This belief does not necessarily hold in the long run, since the other player can behave differently at different time in a game. Even worse, there is no guarantee that this belief is true in the next few iterations. Nonetheless, hypothesized policies constructed by an agent in certain non-zero-sum games such as the IPD usually have a high degree of accuracy in predicting what the other player will do.

This belief is subjective—it depends on the choice of the policy schema and the learning function. We formally define this subjective viewpoint as follows. The *hypothesized policy space* spanned by a policy schema $\mathcal{C}$ and a learning function $L : \mathcal{C} \times \mathcal{H} \rightarrow$

$[0, 1]$ is a set of policies $\Pi = \{\pi(H) : \tau \in \mathcal{H}\}$, where $\pi(\tau) = \{\mathsf{Cond} \rightarrow L(\mathsf{Cond}, \tau) : \mathsf{Cond} \in \mathcal{C}\}$. Let τ be a history of a game in which the other player's strategy is ψ. The set of all possible hypothesized policies for ψ in this game is $\{\pi(\tau_k) : \tau_k \in \mathsf{prefixes}(\tau)\} \subseteq \Pi$, where $\mathsf{prefixes}(\tau)$ is the set of all prefixes of τ, and τ_k is the prefix of length k of τ. We say $\pi(\tau_k)$ is the *current* hypothesized policy of ψ in the iteration k. A rule $\mathsf{Cond} \rightarrow p$ in $\pi(\tau_k)$ describes a particular *behavior* of the other player's strategy in the iteration k. The behavior is *deterministic* if p is either zero or one; otherwise, the behavior is *random* or *probabilistic*. If $\pi(\tau_k) \neq \pi(\tau_{k+1})$, we say there is a change of the hypothesized policy in the iteration $k + 1$, and the behaviors described by the rules in $(\pi(\tau_k) \setminus \pi(\tau_{k+1}))$ have changed.

4.4 Derived Belief Strategy

In the ordinary Iterated Prisoner's Dilemma (i.e., without any noise), if we know the other player's strategy and how many iterations in a game, we can compute an optimal strategy against the other player by trying every possible sequence of moves to see which sequence yields the highest score, assuming we have sufficient computational power. However, we are missing both pieces of information. So it is impossible for us to compute an optimal strategy, even with sufficient computing resource. Therefore, we can at most predict the other player's moves based on the history of a game, subject to the fact that the game may terminate any time.

Some strategies for the Iterated Prisoner's Dilemma do not predict the other player's moves at all. For example, Tit-for-Tat and GRIM react deterministically to the other

player's previous moves according to fixed sets of rules, no matter how the other player actually plays. Many strategies adapt to the other player's strategy over the course of the game: for example, Pavlov [35] adjusts its degree of cooperation according to the history of a game. However, these strategies do not take any prior information about the other player's strategy as an input; thus they are unable to make use of this important piece of information even when it is available.

Let us consider a class of strategies that make use of a model of the other player's strategy to make decisions. Figure 4.1 shows an abstract representation of these strategies. Initially, these strategies start out by assuming that the other player's strategy is TFT or some other strategy. In every iteration of the game, the model is updated according to the current history (using UpdateModel). These strategies decide which move it should make in each iteration using a move generator (GenerateMove), which depends on the current model of the other player's strategy of the iteration.

DBS belongs to this class of strategies. DBS maintains a model of the other player in form of a hypothesized policy throughout a game, and makes decisions based on this hypothesized policy. The key issue for DBS in this process is how to maintain a good approximation of the other player's strategy, despite that some actions in the history are affected by noise. A good approximation will increase the quality of moves generated by DBS, since the move generator in DBS depends on an accurate model of the other player's behavior.

The approach DBS uses to minimize the effect of noise on the hypothesized policy has been discussed in Section 4.1: temporarily tolerate possible misbehavior by the other

```
Procedure StrategyUsingModelOfTheOtherPlayer()
    π ← InitialModel()          // the current model of the other player
    τ ← ⟨⟩                      // the current history
    a ← GenerateMove(π, τ)      // the initial move
    Loop until the end of the game
        Output our move a and obtain the other player's move b
        τ := τ ∘ ⟨(a, b)⟩
        π := UpdateModel(π, τ)
        a := GenerateMove(π, τ)
    End Loop
```

Figure 4.1: An abstract representation of a class of strategies that generate moves using a model of the other player.

player, and then update the hypothesized policy only if DBS believes that the misbehavior is due to a genuine change of behaviors. Figure 4.2 shows an outline of the implementation of this approach in DBS. As we can see, DBS does not maintain the hypothesized policy explicitly; instead, DBS maintains three sets of rules: the default rule set (R_d), the current rule set (R_c), and the probabilistic rule set (R_p). DBS combines these rule sets to form a hypothesized policy for move generation. In addition, DBS maintains several auxiliary variables (promotion counts and violation counts) to facilitate the update of these rule sets. We will explain every line in Figure 4.2 in detail in the next section.

4.4.1 Discussion

When using a policy to express the other player's strategy, an important question is how large a policy schema to use for the hypothesized policy. If the other player's strategy is complicated and the policy schema is too small, the policy schema won't provide enough detail to give useful predictions of the other player's behavior. But if the policy schema is too large, DBS will be unable to compute an accurate approximation of each rule's degree of cooperation, because the number of iterations in the game is too small for learning all the rules precisely. After all, a large policy does not necessarily outperform a small one when it is used to project the behavior of the other player in future; in fact, it is worse if the policy schema is too large due to the first reason. For these reasons we shall use a small policy schema. We found that a simple policy is sufficient for our noise detection technique to work. In the competition, we used a policy schema of size 4: $\{(C,C),(C,D),(D,C),(D,D)\}$. We have found this to be good enough for modeling a

Procedure $\mathbf{DerivedBeliefStrategy}()$

1. $R_d := \pi_{TFT}$ // *the default rule set*
2. $R_c := \emptyset$ // *the current rule set*
3. $a_0 := C$; $b_0 := C$; $\tau := \langle (a_0, b_0) \rangle$; $\pi := R_d$; $k := 1$; $v := 0$
4. $a_1 := \mathsf{MoveGen}(\pi, \tau)$
5. Loop until the end of the game
6. Output a_k and obtain the other player's move b_k
7. $r^+ := ((a_{k-1}, b_{k-1}) \rightarrow b_k)$
8. $r^- := ((a_{k-1}, b_{k-1}) \rightarrow (\{C, D\} \setminus \{b_k\}))$
9. If $r^+, r^- \notin R_c$, then
10. If $\mathsf{ShouldPromote}(r^+) = \mathsf{true}$, then insert r^+ into R_c.
11. If $r^+ \in R_c$, then set the violation count of r^+ to zero
12. If $r^- \in R_c$ and $\mathsf{ShouldDemote}(r^-) = \mathsf{true}$, then
13. $R_d := R_c \cup R_d$; $R_c := \emptyset$; $v := 0$
14. If $r^- \in R_d$, then $v := v + 1$
15. If $v > RejectThreshold$, or ($r^+ \in R_c$ and $r^- \in R_d$), then
16. $R_d := \emptyset$; $v := 0$
17. $R_p := \{(\mathsf{Cond} \rightarrow p') \in R_{k+1}^{prob} : \mathsf{Cond}$ not appear in R_c or $R_d\}$
18. $\pi := R_c \cup R_d \cup R_p$ // *construct a hypothesized policy*
19. $\tau := \tau \circ \langle (a_k, b_k) \rangle$; $a_{k+1} := \mathsf{MoveGen}(\pi, \tau)$; $k := k + 1$
20. End Loop

Figure 4.2: An outline of the DBS strategy. ShouldPromote first increases r^+'s promotion count, and then if r^+'s promotion count exceeds the promotion threshold, ShouldPromote returns true and resets r^+'s promotion count. Likewise, ShouldDemote first increases r^-'s violation count, and then if r^-'s violation count exceeds the violation threshold,

large number of strategies for the purpose of noise detection.

4.5 Learning Hypothesized Policies in Noisy Environments

We will describe how DBS learns and maintains a hypothesized policy for the other player's strategy in this section. Section 4.5.1 describes how DBS uses *discounted frequencies* for each behavior to estimate the degree of cooperation of each rule in the hypothesized policy. Section 4.5.2 explains why using discounted frequencies alone are not sufficient for constructing an accurate model of the other player's strategy in the presence of noise, and how symbolic noise detection and temporary tolerance can help overcome the difficulty in using discounted frequencies alone. Section 4.5.3 presents the induction technique DBS uses to identify deterministic behaviors in the other player. Section 4.5.4 illustrates how DBS defers judgment about whether an anomaly is due to noise. Section 4.5.5 discusses how DBS updates the hypothesized policy when it detects a change of behavior.

4.5.1 Learning by Discounted Frequencies

In Section 4.3, we introduced the learning function L_{avg}, which estimate the probabilities of cooperation by averaging the number of "firings" of rules in the current history. L_{avg} does not distinguish early moves from recent moves. However, agents in the IPD often changes its behavior—their behavior can change from time to time. It is essentially true in noisy environments, since noise can trigger the change of the behavior. Thus, early moves should be less useful in the estimation of cooperation than recent moves in

the current history.

We now describe a better learning function to estimate the degree of cooperation of the rules in the hypothesized policy. The idea is to maintain a *discounted frequency* for each behavior: instead of keeping an ordinary frequency count of how often the other player cooperates under a condition in the past, DBS applies discount factors based on how recent each occurrence of the behavior was.

Given a history $\tau = \{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$, a real number α between 0 and 1 (called the *discount factor*), and an initial hypothesized policy $\pi_0 = \{\mathsf{Cond}_1 \rightarrow p_1^0, \mathsf{Cond}_2 \rightarrow p_2^0, \ldots, \mathsf{Cond}_n \rightarrow p_n^0\}$ whose policy schema is $\mathcal{C} = \{\mathsf{Cond}_1, \mathsf{Cond}_2, \ldots, \mathsf{Cond}_n\}$, the *probabilistic policy* at iteration $k+1$ is $R_{k+1}^{prob} = \{\mathsf{Cond}_1 \rightarrow p_1^{k+1}, \mathsf{Cond}_2 \rightarrow p_2^{k+1}, \mathsf{Cond}_n \rightarrow p_n^{k+1}\}$, where p_i^{k+1} is computed by the following equation:

$$p_i^{k+1} = \frac{\sum_{0 \leq j \leq k} \left(\alpha^{k-j} g_j\right)}{\sum_{0 \leq j \leq k} \left(\alpha^{k-j} f_j\right)} \tag{4.2}$$

and where

$$g_j = \begin{cases} p_i^0 & \text{if } j = 0, \\ 1 & \text{if } 1 \leq j \leq k, \mathsf{Cond}_i(\tau_{j-1}) = \mathsf{True} \text{ and } b_j = C, \\ 0 & \text{otherwise;} \end{cases}$$

$$f_j = \begin{cases} p_i^0 & \text{if } j = 0, \\ 1 & \text{if } 1 \leq j \leq k, \mathsf{Cond}_i(\tau_{j-1}) = \mathsf{True}, \\ 0 & \text{otherwise;} \end{cases}$$

$$\tau_{j-1} = \begin{cases} \emptyset & \text{if } j = 1, \\ \langle (a_1, b_1), (a_2, b_2), \ldots, (a_{j-1}, b_{j-1}) \rangle & \text{otherwise.} \end{cases}$$

In short, the current history τ has $k+1$ possible prefixes, and f_j is basically a Boolean function indicating whether the prefix of τ up to the $j-1$'th iteration satisfies Cond_i. g_j is a restricted version of f_j.

When $\alpha = 1$, p_i is approximately equal to the frequency of the occurrence of $\mathsf{Cond}_i \rightarrow p_i$. When α is less than 1, p_i becomes a weighted sum of the frequencies that gives more weight to recent events than earlier ones. For our purposes, it is important to use $\alpha < 1$, because it may happen that the other player changes its behavior suddenly, and therefore we should forget about its past behavior and adapt to its new behavior (for instance, when GRIM is triggered). In the competition, we used $\alpha = 0.75$.

An important question is how large a policy schema to use for the hypothesized policy. If the policy schema is too small, the policy schema won't provide enough detail to give useful predictions of the other player's behavior. But if the policy schema is too large, DBS will be unable to compute an accurate approximation of each rule's degree of cooperation, because the number of iterations in the game will be too small. In the competition, we used a policy schema of size 4: $\{(C,C),(C,D),(D,C),(D,D)\}$. We have found this to be good enough for modeling a large number of strategies.

It is essential to have a good initial hypothesized strategy because at the beginning of the game the history is not long enough for us to derive any meaningful information about the other player's strategy. Due to the past success of TFT, we believe many strategies in the competition follows or mimics TFT. Thus, in the competition the initial hypothesized policy of DBS is $\pi_{TFT} = \{(C,C) \rightarrow 1.0, (C,D) \rightarrow 1.0, (D,C) \rightarrow 0.0, (D,D) \rightarrow 0.0\}$.

4.5.2 Deficiencies of Discounted Frequencies in Noisy Environments

It may appear that the probabilistic policy learned by the discounted-frequency learning technique should be inherently capable of tolerating noise, because it takes many, if not all, moves in the history into account: if the number of terms in the calculation of the average or weighted average is large enough, the effect of noise should be small. However, there is a problem with this reasoning: it neglects the effect of multiple occurrences of noise within a small time interval.

A mis-implementation that alters the move of one player would distort an established pattern of behavior observed by the other player. The general effect of such distortion to the Equation 4.2 is hard to tell—it varies with the value of the parameters and the history. But if several distortions occur within a small time interval, the distortion may be big enough to alter the probabilistic policy and hence change our decision about what move to make. This change of decision may potentially destroy an established pattern of mutual cooperation between the players.

At first glance, it might seem rare for several noise events to occur at nearly the same time. But if the game is long enough, the probability of it happening can be quite high. The probability of getting two noise events in two consecutive iterations out of a sequence of i iterations can be computed recursively as $X_i = p(p + qX_{i-2}) + qX_{i-1}$, providing that $X_0 = X_1 = 0$, where p is the probability of a noise event and $q = 1 - p$. In the competition, the noise level was $p = 0.1$ and $i = 200$, which gives $X_{200} = 0.84$. Similarly, the probabilities of getting three and four noises in consecutive iterations are 0.16 and 0.018, respectively.

In the 2005 competition, there were 165 players, and each player played each of the other players five times. This means every player played 825 games. On average, there were 693 games having two noises in two consecutive iterations, 132 games having three noises in three consecutive iterations, and 15 games having four noises in four consecutive iterations. Clearly, we did not want to ignore situations in which several noises occur nearly at the same time.

Symbolic noise detection and temporary tolerance outlined in Section 4.1 provide a way to reduce the amount of susceptibility to multiple occurrences of noise in a small time interval. Deterministic rules enable DBS to detect anomalies in the observed behavior of the other player. DBS temporarily ignores the anomalies which may or may not be due to noise, until a better conclusion about the cause of the anomalies can be drawn. This temporary tolerance prevents DBS from learning from the moves that may be affected by noise, and hence protects the hypothesized policy from the influence of errors due to noise. Since the amount of tolerance (and the accuracy of noise detection) can be controlled by adjusting parameters in DBS, we can reduce the amount of susceptibility to multiple occurrences of noise by increasing the amount of tolerance, at the expense of a higher cost of noise detection—losing more points when a change of behavior occurs.

4.5.3 Identifying Deterministic Rules Using Induction

As we discussed in Section 4.1, deterministic behaviors are abundant in the Iterated Prisoner's Dilemma. Deterministic behaviors can be modeled by deterministic rules, whereas random behavior would require probabilistic rules.

A nice feature about deterministic rules is that they have only two possible degrees of cooperation: zero or one, as opposed to an infinite set of possible degrees of cooperation of the probabilistic rules. Therefore, there should be ways to learn deterministic rules that are much faster than the discounted frequency method described earlier. For example, if we knew at the outset which rules were deterministic, it would take only one occurrence to learn each of them: each time the condition of a deterministic rule was satisfied, we could assign a degree of cooperation of 1 or 0 depending on whether the player's move was C or D.

The trick, of course, is to determine which rules are deterministic. We have developed an inductive-reasoning method to distinguish deterministic rules from probabilistic rules during learning and to learn the correct degree of cooperation for the deterministic rules.

In general, induction is the process of deriving general principles from particular facts or instances. To learn deterministic rules, the idea of induction can be used as follows. If a certain kind of behavior occurs repeatedly several times, and during this period of time there is no other behavior that contradicts to this kind of behavior, then we will hypothesize that the chance of the same kind of behavior occurring in the next few iterations is pretty high, regardless of how the other player behaved in the remote past.

More precisely, let $K \geq 1$ be a number which we will call the *promotion threshold*. Let $\tau = \langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$ be the current history. For each condition $\mathsf{Cond}_j \in \mathcal{C}$, let I_j be the set of indexes such that for all $i \in I_j$, $i < k$ and $\mathsf{Cond}_j(\langle (a_1, b_1), (a_2, b_2), \ldots, (a_i, b_i) \rangle) = \mathsf{True}$. Let $\hat{I}_j$ be the set of the largest K indexes in I_j. If $|I_j| \geq K$ and for all $i \in \hat{I}_j$, $b_{i+1} = C$ (i.e., the other player chose C when the

previous history up to the i'th iteration satisfies Cond_j), then we will hypothesize that the other player will choose C whenever Cond_j is satisfied; hence we will use $\mathsf{Cond}_j \rightarrow 1$ as a deterministic rule. Likewise, if $|I_j| \geq K$ and for all $i \in \hat{I}_j$, $b_{i+1} = D$, we will use $\mathsf{Cond}_j \rightarrow 0$ as a deterministic rule. See Line 7 to Line 10 in Figure 4.2 for an outline of the induction method we use in DBS.

The induction method can be faster at learning deterministic rules than the discounted frequency method that regards a rule as deterministic when the degree of cooperation estimated by discounted frequencies is above or below certain thresholds. As can be seen in Figure 4.3, the induction method takes only three iterations to infer the other player's moves correctly, whereas the discounted frequency technique takes six iterations to obtain a 95% degree of cooperation, and it never becomes 100%.[6] We may want to set the threshold in the discounted frequency method to be less than 0.8 to make it faster than the induction method. However, this will increase the chance of incorrectly identifying a random behavior as deterministic.

A faster learning speed allows us to infer deterministic rules with a shorter history, and hence increase the effectiveness of symbolic noise detection by having more deterministic rules at any time, especially when a change of the other player's behavior occurs. The promotion threshold K controls the speed of the identification of deterministic rules. The larger the value of K, the slower the speed of identification, but the less likely we will mistakenly hypothesize that the other player's behavior is deterministic.

[6] If we modify Equation 4.2 to discard the early interactions of a game, the degree of cooperation of a probabilistic rule can attain 100%.

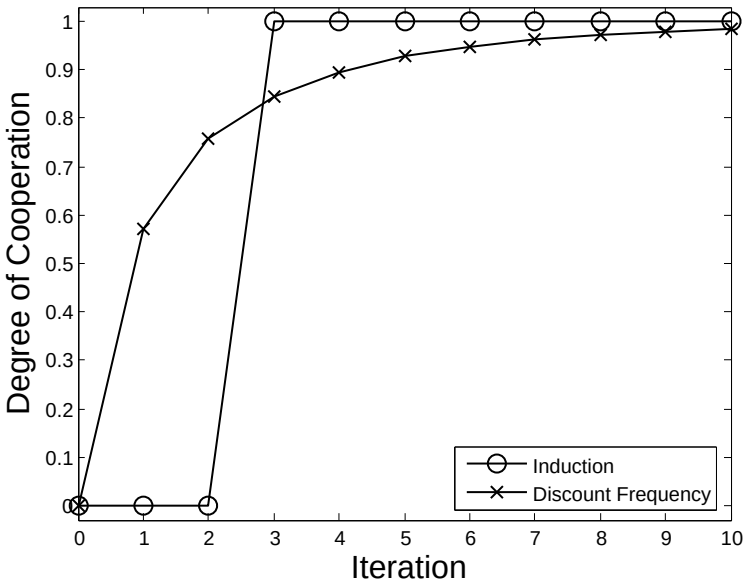

Figure 4.3: Learning speeds of the induction method and the discounted frequency method when the other player always cooperates. The initial degree of cooperation is zero, the discounted rate is 0.75, and the promotion threshold is 3.

4.5.4 Symbolic Noise Detection and Temporary Tolerance

Once DBS has identified the set of deterministic rules, it can readily use them to detect noise. As we said earlier, if the other player's move violate a deterministic rule, it can be caused either by noise or by a change in the other player's behavior, and DBS uses an evidence collection process to figure out which is the case. More precisely, once a deterministic rule $\mathsf{Cond}_i \rightarrow o_i$ is violated (i.e., the history up to the previous iteration satisfies Cond_i but the other player's move in the current iteration is different from o_i), DBS keeps the violated rule but marks it as violated. Then DBS starts an *evidence collection process* that in the implementation of our competition entries is a violation counting: for each violated probabilistic rule DBS maintains a counter called the *violation count* to record how

many violations of the rule have occurred (Line 12).[7] In the subsequent iterations, DBS increases the violation count by one every time a violation of the rule occurs. However, if DBS encounters a positive example of the rule, DBS resets the violation count to zero and unmark the rule (Line 11). If any violation count excesses a threshold called the *violation threshold*, DBS concludes that the violation is not due to noise; it is due to a change of the other player's behavior. In this case, DBS invokes a special procedure (described in Section 4.5.5) to handle this situation (Line 13).

This evidence collection process takes advantages of the fact that the pattern of moves affected by noise is often quite different from the pattern of moves generated by the new behavior after a change of behavior occurs. Therefore, it can often distinguish noise from a change of behavior by observing moves in the next few iterations and gather enough evidence.

As discussed in Section 4.5.2, we want to set a larger violation threshold in order to avoid the drawback of the discount frequency method in dealing with several misinterpretations caused by noise within a small time interval. However, if the threshold is too large, it will slow down the speed of adaptation to changes in the other player's behavior. In the competition, we entered DBS several times with several different violation thresholds; and in the one that performed the best, the violation threshold was 4.

4.5.5 Coping with Ignorance of the Other Player's New Behavior

When the evidence collection process detects a change in the other player's behavior, DBS knows little about the other player's new behavior. How DBS copes with this

[7] We believe that a better evidence collection process should be based on statistical hypothesis testing.

ignorance is critical to its success.

When DBS knows little about the other player's new behavior when it detects a change of the other player's behavior, DBS temporarily uses the previous hypothesized policy as the current hypothesized policy, until it deems that this substitution no longer works. More precisely, DBS maintains two sets of deterministic rules: the *current rule set* R_c and the *default rule set* R_d. R_c is the set of deterministic rules that is learned after the change of behavior occurs, while R_d is the set of deterministic rules before the change of behavior occurs. At the beginning of a game, R_d is π_{TFT} and R_c is an empty set (Line 1 and Line 2). When DBS constructs a hypothesized policy π for move generation, it uses every rule in R_c and R_d. In addition, for any missing rule (i.e., the rule those condition are different from any rule's condition in R_c or R_d), we regard it as a probabilistic rule and approximate its degree of cooperation by Equation 4.2 (Line 17). These probabilistic rules form the probabilistic rule set $R_p \subseteq R_{k+1}^{prob}$.

While DBS can insert any newly found deterministic rule in R_c, it insert rules into R_d *only when* the evidence collection process detects a change of the other player's behavior. When it happens, DBS copies all the rules in R_c to R_d, and then set R_c to an empty set (Line 13).

The default rule set is designed to be *rejected*: we maintain a violation count to record the number of violations to any rule in R_d. Every time any rule in R_d is violated, the violation count increased by 1 (Line 14). Once the violation count exceeds a *rejection threshold*, we drop the default rule set entirely (set it to an empty set) and reset the violation count (Line 15 and Line 16). We also reject R_d whenever any rule in R_c contradicts any rule in R_d (Line 15).

We preserve the rules in R_c mainly for sake of providing a smooth transition: we don't want to convert all deterministic rules to probabilistic rules at once, as it might suddenly alter the course of our moves, since the move generator in DBS generates moves according to the current hypothesized policy only. This sudden change in DBS's behavior can potentially disrupt the cooperative relationship with the other player. Furthermore, some of the rules in R_c may still hold, and we don't want to learn them from scratch.

Notice that symbolic noise detection and temporary tolerance makes use of the rules in R_c but not the rules in R_d, although DBS makes use of the rules in both R_c and R_d when DBS decides the next move (Line 18). We do not use R_d for symbolic noise detection and temporary tolerance because when DBS inserts rules into R_d, a change of the other player's behavior has already occurred—there is little reason to believe that anomalies detected using the rules in R_d are due to noise. Furthermore, we want to turn off symbolic noise detection and temporary tolerance temporarily when a change of behavior occurs, in order to identify a whole new set of deterministic rules from scratch.

4.6 The Move Generator in DBS

We devised a simple and reasonably effective move generator for DBS. As shown in Figure 4.1, the move generator takes the current hypothesized policy π and the current history $\tau_{current}$ whose length is $l = |\tau_{current}|$, and then decides whether DBS should cooperate in the current iteration. It is difficult to devise a good move generator, because our move could lead to a change of the hypothesized policy and complicate our projection of the long-term payoff. Perhaps, the move generator should take the other player's model

of DBS into account [18]. However, we found that by making the assumption that hypothesized policy will not change for the rest of the game, we can devise a simple move generator that generates fairly good moves. The idea is that we compute the *maximum expected score* we can possibly earn for the rest of the game, using a technique that combines some ideas from both game-tree search and Markov Decision Processes (MDPs). Then we choose the first move in the set of moves that leads to this maximum expected score as our move for the current iteration.

To accomplish the above, we consider all possible histories whose prefix is $\tau_{current}$ as a tree. In this tree, each path starting from the root represents a possible history, which is a sequence of past interactions in $\tau_{current}$ plus a sequence of possible interactions in future iterations. Each node on a path represents the interaction of an iteration of a history. Figure 4.4 shows an example of such a tree. The root node of the tree represents the interaction of the first iteration.

Let $\mathsf{interaction}(S)$ be the interaction represented by a node S. Let $\langle S_0, S_1, \ldots, S_k \rangle$ be a sequence of nodes on the path from the root S_0 to S_k. We define the *depth* of S_k to be $k - l$, and the *history* of S_k be $\tau(S_k) = \langle \mathsf{interaction}(S_1), \mathsf{interaction}(S_2), \ldots, \mathsf{interaction}(S_k) \rangle$. S_i is called the *current node* if the depth of S_i is zero; the current node represents the interaction of the last iteration and $\tau(S_i) = \tau_{current}$. As we do not know when the game will end, we assume *it will go for N^* more iterations*; thus each path in the tree has length of at most $l + N^*$.

Our objective is to compute a non-negative real number called the *maximum expected score* $E(S)$ for each node S with a non-negative depth. Like a conventional game tree search in computer chess or checkers, the maximum expected scores are de-

fined recursively: the maximum expected score of a node at depth i is determined by the maximum expected scores of its children nodes at depth $i + 1$. The maximum expected score of a node S of depth N^* is assumed to be the value computed by an *evaluation function* f. This is a mapping from histories to non-negative real numbers, such that $E(S) = f(\tau(S))$. The maximum expected score of a node S of depth k, where $0 \leq k < N^*$, is computed by the *maximizing rule*: suppose the four possible nodes after S are S_{CC}, S_{CD}, S_{DC}, and S_{DD}, and let p be the degree of cooperation predicted by the current hypothesized policy π (i.e., p is the right-hand side of a rule $(\mathsf{Cond} \rightarrow p)$ in π such that $\tau(S)$ satisfies the condition Cond). Then $E(S) = \max\{E_C(S), E_D(S)\}$, where $E_C(S) = p(u_{CC} + E(S_{CC})) + (1-p)(u_{CD} + E(S_{CD}))$ and $E_D(S) = p(u_{DC} + E(S_{DC})) + (1-p)(u_{DD} + E(S_{DD}))$. Furthermore, we let $\mathsf{move}(S)$ be the decision made by the maximizing rule at each node S, i.e., $\mathsf{move}(S) = C$ if $E_C(S) \geq E_D(S)$ and $\mathsf{move}(S) = D$ otherwise. By applying this maximizing rule recursively, we obtain the maximum expected score of every node with a non-negative depth. The move that we choose for the current iteration is $\mathsf{move}(S_i)$, where S_i is the current node.

The number of nodes in the tree increases exponentially with N^*. Thus, the tree can be huge—there are over a billion nodes when $N^* \geq 15$. It is infeasible to compute the maximum expected score for every node one by one. Fortunately, we can use dynamic programming to speed up the computation. As an example, suppose the hypothesized policy is $\pi = \{(C, C) \rightarrow p_{CC}, (C, D) \rightarrow p_{CD}, (D, C) \rightarrow p_{DC}, (D, D) \rightarrow p_{DD}\}$, and suppose the evaluation function f returns a constant $f_{o_1 o_2}$ for any history that satisfies the

condition (o_1, o_2), where $o_1, o_2 \in \{C, D\}$. Then, given our assumption that the hypothesized policy does not change, it is not hard to show by induction that all nodes whose histories have the same length and satisfy the same condition have the same maximum expected score. By using this property, we construct a table of size $4 \times (N^* + 2)$ in which each entry, denoted by $E^k_{o_1 o_2}$, stores the maximum expected score of the nodes whose histories have length $l + k$ and satisfy the condition (o_1, o_2), where $o_1, o_2 \in \{C, D\}$. We also have another table of the same size to record the decisions the procedure makes; the entry $m^k_{o_1 o_2}$ of this table is the decision being made at $E^k_{o_1 o_2}$. Initially, we set $E^{N+1}_{CC} = f_{CC}$, $E^{N+1}_{CD} = f_{CD}$, $E^{N+1}_{DC} = f_{DC}$, and $E^{N+1}_{DD} = f_{DD}$. Then the maximum expected scores in the remaining entries can be computed by the following recursive equation:

$$\begin{aligned} E^k_{o_1 o_2} = \max \quad & \big\{ p_{o_1 o_2}(u_{CC} + E^{k+1}_{CC}) + (1 - p_{o_1 o_2})(u_{CD} + E^{k+1}_{CD}), \\ & p_{o_1 o_2}(u_{DC} + E^{k+1}_{DC}) + (1 - p_{o_1 o_2})(u_{DD} + E^{k+1}_{DD}) \big\}, \end{aligned} \tag{4.3}$$

where $o_1, o_2 \in \{C, D\}$. Similarly, $m^k_{o_1 o_2} = C$ if $(p_{o_1 o_2}(u_{CC} + E^{k+1}_{CC}) + (1 - p_{o_1 o_2})(u_{CD} + E^{k+1}_{CD})) \geq (p_{o_1 o_2}(u_{DC} + E^{k+1}_{DC}) + (1 - p_{o_1 o_2})(u_{DD} + E^{k+1}_{DD})$ and $m^k_{o_1 o_2} = D$ otherwise.

The policy $\{(C, C) \rightarrow m^0_{CC}, (C, D) \rightarrow m^0_{CD}, (D, C) \rightarrow m^0_{DC}, (D, D) \rightarrow m^0_{DD}\}$ is called the *recommended policy*. If the interaction of the previous iteration is (o_1, o_2), we pick $m^0_{o_1 o_2}$ as the *recommended move* for the current iteration. The pseudocode of this dynamic programming algorithm is shown in Figure 4.5.

4.6.1 Generalizing the Move Generator

In general, the policy schema of the hypothesized policy can be different from the simple policy schema that have been used throughout this chapter (i.e.,

$\{(C,C),(C,D),(D,C),(D,D)\}$). According to our experience, this simple policy schema is good enough for move generation and noise detection in the IPD. In other problems, however, the policy schema may not be expressive enough to describe the behavior of the opponents for move generation and noise detection. If we use a different policy schema, the equation 4.3 is no longer applicable and we have to use a different set of recursive equations.

In general, the problem can be viewed as a Markov Decision Process (MDP) in which the other player constitutes a nondeterministic environment whose states are the conditions in the policy schema and whose transition matrix has probabilities that are equal to the degrees of cooperation of the rules in the hypothesized policy. Then the recursive equations are similar to the ones in the value iteration for computing an optimal policy for MDPs.

There are subtle differences between solving an MDP and generating a move for the IPD. First, a policy in MDPs is different from a policy we defined here. A policy in MDPs is a mapping from states to actions, while a policy we defined in this chapter is a mapping from conditions to probabilities of cooperation. The difference is due to the fact that DBS uses a policy to model an opponent's behavior; DBS does not use a policy to represent a strategy for itself to interact with other players. As discussed in Section 4.3, a policy for agent modeling is just a hypothesized policy of the behavior of the other players. Since we do not know exactly the set of states of the opponent, we replace the concepts of states in MDPs by policy schemata, which is a set of conditions over histories. Furthermore, since we do not know exactly what the opponent would do under each condition, we use a probability distribution over the set of actions at the right-hand side of a rule in a policy.

These difference reflects the purpose of opponent modeling.

Second, the MoveGen procedure is different from value iteration for MDPs in a number of ways. Value iteration computes the optimal solution for an infinite-horizon MDP. But the horizon of a game of the IPD is finite, despite the fact that the players do not know the length of a game, and this makes the games look like an infinite game from the players' viewpoint. Instead of using a discount factor, the MoveGen procedure uses a fixed search depth and an evaluation function to estimate the expected utilities at the terminal nodes. The use of fixed search depth and the evaluation function is the remnants of game-tree search. We haven't checked to see whether MoveGen is better than value iteration in our context, thus we cannot conclude that MoveGen outperforms value iteration with a discount factor. But the obvious benefit of the MoveGen procedure is that the expected utilities are intuitive than the discounted expected utilities in value iteration. Moreover, we believe that the results of the MoveGen procedure are similar to value iteration with a large discount factor.

4.6.2 An Analysis of the Move Generator

The dynamic programming algorithm for move generation can virtually search as deep as we want in the game tree. In the competition, the dynamic programming algorithm stops exploring the game tree at the cut-off level $N^* = 60$, and then uses the evaluation function f as described in the caption of Figure 4.5 to estimate the expected scores of the nodes at the cut-off level. Would there be any difference with the recommended policies and the recommended moves if we increase the cut-off level and use

other evaluation functions? We conducted an experiment to investigate this issue, and the result is presented in this section.

In the experiment, we generated a set of 625 hypothesized policies of the form $\{(C,C) \rightarrow u_{CC}, (C,D) \rightarrow u_{CD}, (D,C) \rightarrow u_{DC}, (D,D) \rightarrow u_{DD}\}$ where the degrees of cooperation u_{CC}, u_{CD}, u_{DC}, and u_{DD} is one of these values: 0.00, 0.25, 0.5, 0.75, and 1.0. For each hypothesized policy, we did the following: first, we varied the search depth N^* from 1 to 100, and computed a recommended policy for each search depth using the MoveGen procedure in Figure 4.5. We labeled the recommended policies as π_1, π_2, ..., π_{100}, where π_i is generated with search depth i. Second, we compared π_i with π_{i+1} to see whether they are different. If there is a difference, we said there is a change of recommended policy at search depth i. Finally, we counted how many changes of recommended policy at search depth between 1 and k to see how often recommended policies changed as we increased the search depth. We then plotted the results in a graph in Figure 4.6. The graph contains a total of 625 lines, one for each hypothesized policy. As can be seen, the recommended policies for most hypothesized policies did not change as we increase the search depth. However, a small number of them changed at every search depth. This phenomenon worries us, because it indicates that the MoveGen procedure is unstable—its result is too sensitive to the choice of the search depth.

To see why this phenomenon occurs, we printed out a few sequences of the recommended policies to see if there is any pattern. We found that in most cases the sequence of recommended policies is just an alternation of two recommended policies as the search depth increases. In a small number of cases, there is an alternation of three, four, or five

recommended policies. Then, we plotted another graph to look at the periodicity of the cycles of the recommended policies in the sequences. First, for each hypothesized policy, we varied the search depth to generate a sequence of 100 recommended policies. Second, for each search depth i (called the starting search depth), we looked at a subset of recommended policies generated with search depths between i and 100 in a sequence, to see if the recommended policies repeat themselves periodically in this subset. Finally, we calculated the percentage of sequences that exhibits a cycle of recommended policies with a particular periodicity for each starting search depth, and plot the results in a graph as shown in Figure 4.7.

As can be seen, over 90% of the sequences has a periodicity of 1 starting from the starting search depth of 5; in other words, the recommended policies do not change as the search depth increases. Around 8% of belief policies would cause the MoveGen procedure to generate a sequence of recommended policies that alternates between two different recommended policies. The group of horizontal lines at the bottom of Figure 4.7 corresponds to the sequences exhibits periodicities of 3, 4, or 5, or "no periodicity" (which means that there is no observable pattern of changes of recommended policies in the sequences). But their periodicity dropped to either 1 or 2 as the starting search depth increases. Thus, increasing the search depth can help to stabilize the solutions returned by the MoveGen procedure. But in about 10% of the situations, the MoveGen procedure still returns alternating sequences of recommended policies as the search depth increases.

It is interesting to see what recommended policies the MoveGen procedure actually returns. Table 4.8 shows the percentage of recommended policies returned by MoveGen

given the set of hypothesized policies in the experiments. Each recommended policy is represented by four characters $m_1m_2m_3m_4$, which means that the recommended policy is $\{(C,C) \rightarrow m_1, (C,D) \rightarrow m_2, (D,C) \rightarrow m_3, (D,D) \rightarrow m_4\}$. For example, $CDDD$ in the table refers to the recommended policy $\{(C,C) \rightarrow C, (C,D) \rightarrow D, (D,C) \rightarrow D, (D,D) \rightarrow D\}$. We did not take the sequence of recommended policies whose periodicity is greater than 1 into account.

We can see that over 60% of recommended policies are ALLD ($DDDD$). Around 8% cooperates only when both our agent and the other player cooperate in the previous round ($CDDD$). A slightly more generous strategy that cooperates when the interaction of the previous round is (C,C) or (D,D) is equally common (8% of $CDDC$). In only about 6% of hypothesized policies MoveGen recommends to cooperate no matter what happened in the previous iterations ($CCCC$ = ALLC). Tit-for-Tat ($CDCD$) is only recommended in less than 1% for all hypothesized policies.

One caveat for interpreting the data in Table 4.8 is that the table does not show how often a recommended policy is returned by MoveGen in the *actual* tournament; it just tells us that if we choose a hypothesized policy *randomly* how often a recommended policy will be used. Thus, the MoveGen procedure might return TFT ($CDCD$) or ALLC ($CCCC$) more often than ALLD in a tournament, since the hypothesized policy for returning ALLD does not occur frequently in the tournament.

4.7 Competition Results

The 2005 IPD Competition was actually a set of four competitions, each for a different version of the IPD. The one for the Noisy IPD was Category 2, which used a noise level of 0.1.

Of the 165 programs entered into the competition, eight of them were provided by the organizer of the competition. These programs included ALLC (always cooperates), ALLD (always defects), GRIM (cooperates until the first defection of the other player, and thereafter it always defects), NEG (cooperate (or defect) if the other player defects (or cooperates) in the previous iteration), RAND (defects or cooperates with the $1/2$ probability), STFT (suspicious TFT, which is like TFT except it defects in the first iteration) TFT, and TFTT. All of these strategies are well known in the literature on IPD.

The remaining 157 programs were submitted by 36 different participants. Each participant was allowed to submit up to 20 programs. We submitted the following 20:

- **DBS.** We entered nine different versions of DBS into the competition, each with a different set of parameters or different implementation. The one that performed best was DBSz, which makes use of the exact set of features we mentioned in this chapter. Versions that have fewer features or additional features did not do as well.

- **Learning of Opponent's Strategy with Forgiveness (LSF).** Like DBS, LSF is a strategy that learns the other player's strategy during the game. The difference between LSF and DBS is that LSF does not make use of symbolic noise detection. It uses the discount frequency (Equation 4.2) to learn the other player's strategy, plus a forgiveness strategy that decides when to cooperate if mutual defection occurs.

We entered one instance of LSF. It placed around the 30'th in three of the runs and around 70'th in the other two runs. We believe the poor ranking of LSF is due to the deficiency of using discount frequency alone as we discussed at the beginning of Section 4.5.

- **Tit-for-Tat Improved (TFTI).** TFTI is a strategy based on a totally different philosophy from DBS's. It is not an opponent-modeling strategy, in the sense that it does not model the other player's behavior using a set of rules. Instead, it is a variant of TFT with a sophisticated forgiveness policy that aims at overcoming some of the deficiencies of TFT in noisy environments. We entered ten instantiations of TFTI in the competition, each with a different set of parameters or some differences in the implementation. The best of these, TFTIm, did well in the competition (see Table 4.1), but not as well as DBS.

Three of the other participants each entered the full complement of twenty programs: Wolfgang Kienreich, Jia-wei Li, and Perukrishnen Vytelingum. All three of them appear to have adopted the *master-and-slaves strategy* that was first proposed by Vytelingum's team from the University of Southampton. A master-and-slaves strategy is not a strategy for a single program, but instead for a team of collaborating programs. One of the programs in such a team is the *master*, and the remaining programs are *slaves*. The basic idea is that at the start of a run, the master and slaves would each make a series of moves using a predefined protocol, in order to identify themselves to each other. From then on, the master program would always play "defect" when playing with the slaves, and the slave programs would always play "cooperate" when playing with the master, so

that the master would gain the highest possible payoff at each iteration. Furthermore, a slave would always plays "defect" when playing with a program other than the master, in order to try to minimize that player's score.

Wolfgang Kienreich's master program was CNGF (CosaNostra Godfather), and its slaves were 19 copies of CNHM (CosaNostra Hitman). Jia-wei Li's master program was IMM01 (Intelligent Machine Master 01), and its slaves were IMS02, IMS03, ..., IMS20 (Intelligent Machine Slave n, for $n = 02, 03, \ldots 20$). Perukrishnen Vytelingum's master program was BWIN (S2Agent1_ZEUS), and its slaves were BLOS2, BLOS3, ..., BLOS20 (like BWIN, these programs also had longer names based on the names of ancient Greek gods).

We do not know what strategies the other participants used in their programs.

4.7.1 Overall Average Scores

Category 2 (IPD with noise) consisted of five runs. Each run was a round-robin tournament in which each program played with every program, including itself. Each program participated in 166 games in each run (recall that there is one game in which a player plays against itself, which counts as two games for that player). Each game consisted of 200 iterations. A program's score for a game is the sum of its payoffs over all 200 iterations (note that this sum will be at least 0 and at most 1000). The program's total score for an entire run is the sum of its scores over all 166 games. On the competition's website, there is a ranking for each of the five runs, each program is ranked according to its total score for the run.

When a program P1 plays with a program P2 n times, the *average score* of P1 is the sum of the total scores divided by n. For example, in the 2005 IPD tournament, every pair of programs played five time, one for each run. Thus a program's expected score against another program P2 is the average of the scores the program received in the five games it played with P2.

The *overall average score* of a program P1 in a tournament is the average of the average scores of P1 against every programs in the tournament, including a copy of P1 itself. The overall average score of P1 is also the average of all scores P1 received in a tournament. In the 2005 IPD tournament, a program's overall average score is its average over all games in all five runs, i.e., its total over all five runs divided by $830 = 5 \times 166$.

The table in Table 4.1 shows the average scores in each of the five runs of the top twenty-five programs when the programs are ranked by their overall average scores. Of our nine different versions of DBS, all nine of them are among the top twenty-five programs, and they dominate the top ten places. This phenomenon implies that DBS's performance is insensitive to the parameters in the programs and the implementation details of an individual program. The same phenomenon happens to TFTI—nine out of ten programs using TFTI are ranked between the 11th place and the 25th place, and the last one is at the 29th place.

4.7.2 DBS versus the Master-and-Slaves Strategies

We analyze the performance of DBS programs against the master-and-slaves strategies in four different ways: (1) compare the performance of the DBS programs with the

performance of the entire master-and-slaves teams as a whole; (2) study the change of the overall average scores as the number of slaves decreases; (3) analyze the percentages of different kinds of interactions among programs; and (4) differentiate the distributions of the average scores of DBSz and the master-and-slave programs using some exploratory data analysis techniques.

4.7.2.1 Group Performance

Recall from Table 4.1: that DBSz placed third in the competition: it lost only to BWIN and IMM01, the masters of two master-and-slaves strategies. DBS does not use a master-and-slaves strategy, nor does it conspire with other programs in any other way—but in contrast, BWIN's and IMM01's performance depended greatly on the points fed to them by their slaves. In fact, if we average the score of each master with the scores of its slaves, we get 379.9 for BWIN and 351.7 for IMM01, both of which are considerably less than DBSz's score of 408.

There are two kinds of programs in the competition that can be viewed as members of a group, rather than as a single program. These include master-slave programs, and variants on the same algorithm. Another way to view the performance of such groups of programs is to average the performance of the members of the group, and that is what we have done in Table 4.2. In this table, the best master-slave strategy is the one submitted by Perukrishnen Vytelingum, which ranks only 14th. DBS, on the other hand, ranks first.

The poor group performance of the master-and-slaves teams implies that a master can hardly recruit a large number of slaves that sacrifices for him in many realistic situa-

Table 4.2: A modified version of Table 4.1 in which we have averaged the scores of each collection of programs that was either (i) a group of conspirators or (ii) a collection of of variants of the same algorithm. The average score for DBS is 402.1, which is higher than the average score of any other program. The master of the best master-slave strategy, BWIN came in 14th with a score of 379.9. Only the top fifteen groups are listed.

Rank	Program(s)	Participant	Overall Avg. Score
1	DBS*	Tsz-Chiu Au	402.1
2	Mod	Philip Hingston	396.9
3	TTFT	Louis Clement	393.4
4	TFTI*	Tsz-Chiu Au	392.9
5	*ESTFT_*	Michael Filzmoser	390.5
6	T4T	David Fogel	390.0
7	TFTT	Tit-for-Two-Tats	388.4
8	(no name)	Bingzhong Wang	388.3
9	TFT	Tit-for-Tat	388.2
10	SOMETHING	Jan Humble	383.8
11	TTFT 1	Quan Zhou	383.6
12	LSF	Tsz-Chiu Au	382.5
13	STFT	Suspicious TFT	382.1
14	BWIN/BLOS*	Perukrishnen Vytelingum	379.9
15	RANB	Muhammad Ahmad	379.3

tions, because there is a strong incentive for slaves to betray their master as they can do much better by themselves.

One might argue that the master can recruit more slaves if the master promises to distribute some of his payoffs to the slaves. However, the possibility of this "private" distributions of payoffs destroys the incentive for agents to form a master-and-slaves team. The average score of a team of 20 agents in which each team members use DBS or any one of top 20 non-master-and-slaves programs (i.e., without collusion) is much higher than the average score a team of 20 agents using the master-slave strategies in this competition. Thus, the team may be better off not to collude but to use any top 20 non-master-and-slaves programs (e.g., DBS), and then distribute their payoffs later (no matter the payoffs is evenly distributed, or the master would get more than the slaves).

4.7.2.2 Overall Average Scores versus Number of Slaves

To see the effect of the presence of slaves in the tournament, we want to study the change of the overall average scores of the programs when the number of slaves varies. Unfortunately we cannot rerun the tournament since the organizer of the tournaments does not release the programs in the tournament to the public. But there is a trick to get around this problem. The organizer provided data files containing the records of the histories of every game in the tournament. From the data files, we can compute the scores of every games in the tournament. Our approach is to recompute the overall average scores of the programs while ignoring the games that involved the slave programs, as if the slave programs did not participate in the tournament. By virtually removing slaves from the

tournament one by one, we can determine the effects of the slaves on the overall average scores of other programs, including their own masters.

Each of the three master-and-slaves teams in the tournament has 19 slaves. We randomly selected k slaves from each teams, and then recompute the overall average scores of the remaining programs while ignoring the scores of the games that involves these slaves, for $0 \leq k \leq 19$. To minimize the biases due to the selection of the slaves, we repeated the above process 20 times with randomly chosen sets of slaves for each k. The averages of the 20 overall average scores of selected programs are shown in Figure 4.9. Due to the space limitation, we cannot show the overall average scores of all $165 - k$ programs in one graph. Thus, we only selectively plotted the overall average scores some programs that performed well in the tournament in Figure 4.9. These programs are BWIN, IMM01, CNGF, DBSz, lowEsTFT_classic, TFTIm, Mod, TTFT, and mediumESTFT_classic. The error bars of the data points in the figure indicate the maximums and the minimums of the overall average scores in the 20 repetitions.

We can see from Figure 4.9 that the overall average scores of the non-master-slave programs including DBSz, lowEsTFT_classic, TFTIm, Mod, TTFT, and mediumESTFT_classic increases super-linearly as the number of slaves decreases. This clearly indicates the destructive effects of the presence of slaves in the tournament. The non-master-slave programs could have done much better without the defections intentionally made by slaves. For instance, the overall average scores of DBSz could increase by approximately 19% if there were no slave in the tournament. Let x be the average scores of DBSz when playing with any slave. Then we get $x = 258.3$ by the equation

$408 = \frac{165-3\times19}{165} \times 487 + \frac{3\times19}{165} \times x$. Since the average scores of a program is at most 200 when playing with ALLD in a game of 200 interactions, we can see that slaves almost always defected with they played with non-master-slave programs such as DBSz.

As opposed to the intuition that the master programs would perform worse in the absence of slaves, the overall average scores of the master programs also increases as the number of slaves decreases. But the rate of increases is not as high as that of the non-master-slave programs. This paradox is due to the fact that the slaves defected when they played with the masters of another teams. Since the slaves of other teams outnumbered the slaves of its own team, a master would perform better if no slave is allowed in the tournament.

The rate of increases in the overall average scores of the masters is slower than the rate of increases of non-master-slave programs, and eventually the overall average scores of non-master-slave programs surpassed the masters' scores. DBSz's scores can potentially surpass BWIN's when the number of slaves is reduced by 9, but almost always surpass BWIN's when the number of slaves is reduced by 11. Thus, it is safe to say that if the size of the master-and-slaves team was restricted to at most 10 in the tournament (i.e., no participant can submit more than 10 programs), then DBSz would have placed first in the tournament.

4.7.2.3 Percentages of Interactions

The reason for the poor performance of master programs with few or no slaves is that the master-and-slaves strategies did not cooperate the other players as much as they

did amongst themselves. In particular, Table 4.3 gives the percentages of each of the four possible interactions when any program from one group plays with any program from another group. Note that:

- When BWIN and IMM01 play with their slaves, about 64% and 47% of the interactions are (D, C), but when non-master-and-slaves strategies play with each other, only 19% of the interactions are (D, C).

- When the slave programs play with non-master-and-slaves programs, over 60% of interactions are (D, D), but when non-master-and-slaves programs play with other non-master-and-slaves programs, only 31% of the interactions are (D, D).

- The master-and-slaves strategies decrease the overall percentage of (C, C) from 31% to 13%, and increase the overall percentage of (D, D) from 31% to 55%.

4.7.2.4 Distributions of Average Scores

All analysis we conducted so far focus on the overall average scores of a program, which, as defined in Section 4.7.1, is a summary of all average scores against every program in a tournament. In order to get a more clearer picture of the performance of a program in the tournament, we have to look at individual average scores that constitute the overall average scores. The problem is that there are large amount of average scores, and we need a way to succinctly present the data in a meaningful way. Our solution is to display the average scores in both *density plots* and *dot plots*, both of them are types of

Table 4.3: Percentages of different interactions. *"All but M&S"* means all 105 programs that did not use master-and-slaves strategies, and *"all"* means all 165 programs in the competition.

Player 1	Player 2	(C, C)	(C, D)	(D, C)	(D, D)
BWIN	BWIN's slaves	12%	5%	64%	20%
IMM01	IMM01's slaves	10%	6%	47%	38%
CNGF	CNGF's slaves	2%	10%	10%	77%
BWIN's slaves	*all but M&S*	5%	9%	24%	62%
IMM01's slaves	*all but M&S*	7%	9%	23%	61%
CNGF's slaves	*all but M&S*	4%	8%	24%	64%
TFT	*all but M&S*	33%	20%	20%	27%
DBSz	*all but M&S*	54%	15%	13%	19%
TFTT	*all but M&S*	55%	20%	11%	14%
TFT	*all*	23%	19%	16%	42%
DBSz	*all*	36%	14%	11%	39%
TFTT	*all*	38%	21%	10%	31%
all but M&S	*all but M&S*	31%	19%	19%	31%
all	*all*	13%	16%	16%	55%

graphically display widely used in exploratory data analysis for visualizing the distributions of numbers.

Figure 4.10 contains seven density plots (the curved lines) overlapped with seven dot plots (the data points) for DBSz, the three master programs, and three slave programs. In the tournament, each program plays against 166 different programs (including a copy of itself), thus has 166 average scores. Since the number of iterations in a game is about 200, the average scores are between 0 and 1000. But none of the average scores is below 200 or above 900. The density plots, as shown as the lines in the graph, outlined the distribution of the average scores of a program against all other programs in the tournament. To draw a density plot for a program, say DBSz, for each average score x we counted how many average scores, out of the 166 average scores of DBSz, fall in the range of $[x-2, x+2]$, and then normalized the counts and plotted the normalized counts. The normalization is needed because the scale in the y-axis is not important—the density plots are intended to show the shape of the distribution rather than the frequency of the numbers. For example, there are two peaks in the density plot of DBSz, and this shows that most of the average scores are around either 260 or 530.

To see why there are two peaks in the density plot of DBSz, we used dot plots to identify which program DBSz plays with when DBSz obtained the average scores. We partitioned the set of all programs in the tournament into six groups: DBS* (all versions of DBS strategies), TFTI* (all versions of TFTI strategies), BWIN and its slaves, IMM01 and its slaves, CNGF and its slaves, and the remaining programs. For each group, we plotted the average scores on top of the density plot. As can be seen in Figure 4.10, the

DBSz's average score when played with the slave programs are around $x = 260$, while the average scores played with many non-master-slave programs are around $x = 530$. This shows that the peak at around 260 is due to the slave programs which defected most of the time, and the peak at around 530 is due to the cooperation between DBSz and non-master-slave programs. It is interesting to see that there is a small peak at the around 580, which is mainly due to the cooperation among different versions of DBS programs but also some non-master-slave programs. Clearly, if both agents use symbolic noise detection, the degree of cooperation can be even higher in noisy environments. From the density plot and the dot plot of DBSz, we can see that the overall average scores of DBSz (the vertical line at $x = 408$) is roughly the midpoint between the two peaks.

The density plot of BWIN is very different from the density plot of DBSz in that the peak at around $x = 530$ is missing; alternatively, we can say that the peak is more spread out than the peak for DBSz. From the density plot, we can see how the slaves of BWIN work: when BWIN played with its slaves, the average scores were often more than 600. This is possible only if BWIN defected most of the time while the slaves cooperated. BWIN can also exploit some of the non-master-slave problem (i.e., some dots beyond $x = 600$). In general, most of the DBS programs tried to cooperate with BWIN. Despite that BWIN also suffers from the slaves of other master-and-slaves team as well, the overall average scores (the vertical line at $x = 433$) is still much higher than DBSz's, mainly due to its slaves.

IMM01's density plot is similar to the one for BWIN, but IMM01's second peak seems to be non-existence. This is why BWIN outperforms IMM01. DBS programs cannot cooperate with IMM01 as well as with BWIN. CNGF cannot even cooperate with

its slaves. BLOS10 is a slave of BWIN. We can see that most of the DBS programs cannot cooperate with BLOS10. Compared with IMS02 (a slave of IMM01), BLOS10 is more successful in cooperating with its teammates.

The graphical analysis of Figure 4.10 shows that slaves did greatly affect the tournament. But what if the slaves did not exist? We drew another graph in which the average scores of the slave programs are removed. In addition, the average scores of DBS programs and TFTI programs, except DBSz and TFTIm, are removed too, such that collusion among programs submitted by the author, if exists at all, would have no effect. The result is shown in Figure 4.11.

When there is no slave, all master programs did not do well. Thus, the top programs among all 91 programs did not include the master programs. We selected the following six top programs and drew the density plots and dot plots in Figure 4.11: DBSz, lowESTFT_classic, TFTIm, Mod, TTFT, and mediumESTFT_classic.

The peak at $x = 260$ in the density plot for DBSz disappears when there is no slave. The remaining peak is the one at $x = 530$. This peak is mainly to due to the cooperation of DBSz with non-master-and-slaves programs. The peaks for other programs are more spread out. More importantly, the peaks are spread to the left side of the graph. This is why the overall average score of DBSz's is higher: the cooperation between DBSz and other programs is more stable in noisy environments. Moreover, this stability can be achieved by DBSz alone—the other players did not use symbolic noise detection.

4.7.3 A comparison between DBSz, TFT, and TFTT

Next, we consider how DBSz performs against TFT and TFTT. Table 4.3 shows that when playing with another cooperative player, TFT cooperates ((C, C) in the table) 33% of the time, DBSz does so 54% of the time, and TFTT does so 55% of the time. Furthermore, when playing with a player who defects, TFT defects ((D, D) in the table) 27% of the time, DBSz does so 19% of the time, and TFTT does so 14% of the time. From this, one might think that DBSz's behavior is somewhere between TFT's and TFTT's.

But on the other hand, when playing with a player who defects, DBSz cooperates ((C, D) in the table) only 15% of the time, which is a lower percentage than for TFT and TFTT (both 20%). Since cooperating with a defector generates no payoff, this makes TFT and TFTT perform worse than DBSz overall. DBSz's average score was 408 and it ranked 3rd, but TFTT's and TFT's average scores were 388.4 and 388.2 and they ranked 30th and 33rd.

Figure 4.12 shows the average scores DBSz got when played against TFT, TFTT, and other programs provided by the organizer of the competition. We can see that TFT did not perform as well as DBSz and TFTT. In fact, TFT performed better when it played against DBSz rather than TFT itself in noisy environments.

4.8 Related Work

Early studies of the effect of noise in the Iterated Prisoner's Dilemma focused on how TFT, a highly successful strategy in noise-free environments, would do in the presence of noise. TFT is known to be vulnerable to noise; for instance, if two players use

TFT at the same time, noise would trigger long sequences of mutual defections [45]. A number of people confirmed the negative effects of noise to TFT [45, 11, 46, 5, 47, 12]. Axelrod found that TFT was still the best decision rule in the rerun of his first tournament with a one percent chance of misperception [3, page 183], but TFT finished sixth out of 21 in the rerun of Axelrod's second tournament with a 10 percent chance of misperception [24]. In Competition 2 of the 2005 IPD competition, the noise level was 0.1, and TFT's overall average score placed it 33rd out of 165.

The oldest approach to remedy TFT's deficiency in dealing with noise is to be more forgiving in the face of defections. A number of studies found that more forgiveness promotes cooperation in noisy environments [12, 46]. For instance, Tit-For-Two-Tats (TFTT), a strategy submitted by John Maynard Smith to Axelrod's second tournament, retaliates only when it receives two defections in two previous iterations. TFTT can tolerate isolated instances of defections caused by noise and is more readily to avoid long sequences of mutual defections caused by noise. However, TFTT is susceptible to exploitation of its generosity and was beaten in Axelrod's second tournament by TESTER, a strategy that may defect every other move. In Competition 2 of the 2005 IPD Competition, TFTT ranked 30—a slightly better ranking than TFT's. In contrast to TFTT, DBS can tolerate not only an isolated defection but also a sequence of defections caused by noise, and at the same time DBS monitors the other player's behavior and retaliates when exploitation behavior is detected (i.e., when the exploitation causes a change of the hypothesized policy, which initially is TFT). Furthermore, the retaliation caused by exploitation continues until the other player shows a high degree of remorse (i.e., cooperations when DBS defects) that changes the hypothesized policy to one with which DBS

favors cooperations instead of defections.

[45] proposed to mix TFT with ALLC to form a new strategy which is now called Generous Tit-For-Tat (GTFT) [49]. Like TFTT, GTFT avoids an infinite echo of defections by cooperating when it receives a defection in certain iterations. The difference is that GTFT forgives randomly: for each defection GTFT receives it randomly choose to cooperate with a small probability (say 10%) and defect otherwise. DBS, however, does not make use of forgiveness explicitly as in GTFT; its decisions are based entirely on the hypothesized policy that it learned. But temporary tolerance can be deemed as a form of forgiveness, since DBS does not retaliate immediately when a defection occurs in a mutual cooperation situation. This form of forgiveness is carefully planned and there is no randomness in it.

Another way to improve TFT in noisy environments is to use contrition: unilaterally cooperate after making mistakes. One strategy that makes use of contrition is Contrite TFT (CTFT) [61, 16, 64], which does not defect when it knows that noise has occurred and affected its previous action. However, this is less useful in the Noisy IPD since a program does not know whether its action is affected by noise or not. DBS does not make use of contrition, though the effect of temporary tolerance resembles contrition.

A family of strategies called "Pavlovian" strategies, or simply called Pavlov, was found to be more successful than TFT in noisy environments [35, 36, 37, 48]. The simplest form of Pavlov is called Win-Stay, Lose-Shift [48], because it cooperates only after mutual cooperation or mutual defection, an idea similar to Simpleton [55]. When an accidental defection occurs, Pavlov can resume mutual cooperation in a smaller number of iterations than TFT [35, 36]. Pavlov learns by conditioned response through rewards and

punishments; it adjusts its probability of cooperation according to the previous interaction. Like Pavlov, DBS learns from its past experience and makes decisions accordingly. DBS, however, has an intermediate step between learning from experience and decision making: it maintains a model of the other player's behavior, and uses this model to reason about noise. Although there are probabilistic rules in the hypothesized policy, there is no randomness in its decision making process.

For readers who are interested, there are several surveys on the Iterated Prisoner's Dilemma with noise [5, 32, 50, 38].

The use of opponent modeling is common in games of imperfect information such as Poker [15, 7, 8, 9, 21, 14] and RoShamBo [27]. One entry in Axelrod's original IPD tournament used opponent modeling, but it was not successful. There have been many works on learning the opponent's strategy in the non-noisy IPD [26, 31, 53]. By assuming the opponent's next move depends only on the interactions of the last few iterations, these works model the opponent's strategy as probabilistic finite automata, and then use various learning methods to learn the probabilities in the automata. For example, [31] proposed an adaptive agent called an opponent modeling agent (OMA) of order n, which maintains a summary of the moves made up to n previous iterations. Like DBS, OMA learns the probabilities of cooperations of the other player in different situations using an updating rule similar to the Equation 4.2, and generates a move based on the opponent model by searching a tree similar to that shown in Figure 4.4. The opponent model in [26] also has a similar construct. The main way they differ from DBS is how they learn the other player's strategy, but there are several other differences: for example, the tree they used has a maximum depth of 4, whereas ours has a depth of 60.

The agents of both [31] and [26] learned the other player's strategy by exploration—deliberately making moves in order to probe the other player's strategy. The use of exploration for learning opponent's behaviors was studied by [19], who developed a lookahead-based exploration strategy to balance between exploration and exploitation and avoid making risky moves during exploration. [31] and [26] used a different exploration strategy than [19]; [31] introduced noise to 1% of their agent's moves (they call this method the trembling hand), whereas the agent in [26] makes decisions at random when it uses the opponent's model and finds a missing value in the model. Both of their agents used a random opponent model at the beginning of a game.

DBS does not make deliberate moves to attempt to explore the other player's strategy, because we believe that this is a high-risk, low-payoff business in IPD. We believe it incurs a high risk because many programs in the competition are adaptive; our defections made in exploration may affect our long-term relationship with them. We believe it has a low payoff because the length of a game is usually too short for us to learn any non-trivial strategy completely. Moreover, the other player may alter its behavior at the middle of a game, and therefore it is difficult for any learning method to converge. It is essentially true in noisy IPD, since noise can provoke the other player (e.g., GRIM). Furthermore, our objective is to cooperate with the other players, not to exploit their weakness in order to beat them. So as long as the opponent cooperates with us there is no need to bother with their other behaviors. For these reasons, DBS does not aim at learning the other player's strategy completely; instead, it learns the other player's recent behavior, which is subject to change. In contrast to the OMA strategy described earlier in this section, most of our DBS programs cooperated with each other in the competition.

Our decision-making algorithm combines elements of both minimax game tree search and the value iteration algorithm for Markov Decision Processes. In contrast to [18], we do not model the other player's model of our strategy; we assume that the hypothesized policy does not change for the rest of the game. Obviously this assumption is not valid, because our decisions can affect the decisions of the other players in the future. Nonetheless, we found that the moves returned by our algorithm are fairly good responses. For example, if the other player behaves like TFT, the move returned by our algorithm is to cooperate regardless of the previous interactions; if the other player does not behave like TFT, our algorithm is likely to return defection, a good move in many situations.

To the best of our knowledge, ours is the first work on using opponent models in the IPD to detect errors in the execution of another agent's actions.

4.9 Summary

For conflict prevention in noisy environments, a critical problem is to distinguish between situations where another player has misbehaved intentionally and situations where the misbehavior was accidental. That is the problem that DBS was formulated to deal with. DBS's impressive performance in the 2005 Iterated Prisoner's Dilemma competition occurred because DBS was better able to maintain cooperation in spite of noise than any other program in the competition.

To distinguish between intentional and unintentional misbehaviors, DBS uses a combination of symbolic noise detection plus temporary tolerance: if an action of the

other player is inconsistent with the player's past behavior, we continue as if the player's behavior has not changed, until we gather sufficient evidence to see whether the inconsistency was caused by noise or by a genuine change in the other player's behavior.

Since clarity of behavior is an important ingredient of long-term cooperation in the IPD, most IPD programs have behavior that follows clear deterministic patterns. The clarity of these patterns made it possible for DBS to construct policies that were good approximations of the other players' strategies, and to use these policies to fend off noise.

We believe that clarity of behavior is also likely to be important in other multi-agent environments in which agents have to cooperate with each other. Thus it seems plausible that techniques similar to those used in DBS may be useful in those domains.

In the future, we are interested in studying the following issues:

- The evidence collection process takes time, and the delay may invite exploitation. For example, the policy of temporary tolerance in DBS may be exploited by a "hypocrite" strategy that behaves like TFT most of the time but occasionally defects even though DBS did not defect in the previous iteration. DBS cannot distinguish this kind of intentional defection from noise, even though DBS has built-in mechanism to monitor exploitation. We are interested to seeing how to avoid this kind of exploitation.

- In multi-agent environments where agents can communicate with each other, the agents might be able to detect noise by using a predefined communication protocol. However, we believe there is no protocol that is guaranteed to tell which action has been affected by noise, as long as the agents cannot completely trust each other. It

would be interesting to compare these alternative approaches with symbolic noise detection to see how symbolic noise detection could enhance these methods or vice versa.

- The type of noise in the competition assumes that no agent know whether an execution of an action has been affected by noise or not. Perhaps there are situations in which some agents may be able to obtain partial information about the occurrence of noise. For example, some agents may obtain a plan of the malicious third party by counter-espionage. We are interested to see how to utilize these information into symbolic noise detection.

- Symbolic noise detection is not designed for noise-free environments. That is why DBS did not work as well in Competition 1 (where there was no noise) as it did in Competition 2. [8] We believe if we have turned off the noise detection mechanism by setting the violation thresholds to zero, the DBS programs would have performed better in Competition 1. Thus, if we want a DBS program to be able to work well in both noisy and noise-free environments, we need a way to detect whether there is noise in the environment, so that DBS can turn off the noise detection mechanism when there is no noise. In general, the questions are (1) how to estimate the level of noise in the environment, and (2) how symbolic noise detection should be adjusted for different noise level.

- It would be interesting to put DBS in an evolutionary environment to see whether

[8] we submitted the same set of programs in both Competition 1 and Competition 2. Although DBS did not win in Competition 1, the best DBS program consistently ranked top 10 among 192 programs.

it can survive after a number of generations. Is it evolutionarily stable?

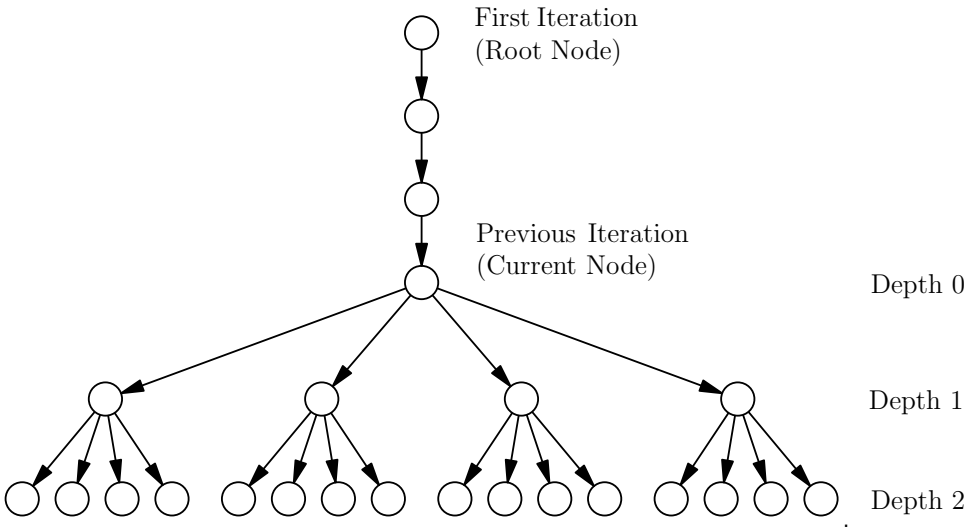

Figure 4.4: An example of the tree that we use to compute the maximum expected scores. Each node denotes the interaction of an iteration. The top four nodes constitute a path representing the current history $\tau_{current}$. The length of $\tau_{current}$ is $l = 2$, and the maximum depth N^* is 2. There are four edges emanating from each node S after the current node; each of these edges corresponds to a possible interaction of the iteration after S. The maximum expected scores (not shown) of the nodes with depth 2 are set by an evaluation function f; these values are then used to calculate the maximum expected scores of the nodes with depth 1 by using the maximizing rule. Similarly, the maximum expected scores of the current node is calculated using four maximum expected scores of the nodes with depth 1.

Procedure MoveGen(π, τ)

$\langle p_{CC}, p_{CD}, p_{DC}, p_{DD} \rangle := \pi$

$\{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\} := \tau$

$(a_0, b_0) := (C, C)$; $(a, b) := (a_k, b_k)$

$\langle E_{CC}^{N^*+1}, E_{CD}^{N^*+1}, E_{DC}^{N^*+1}, E_{DD}^{N^*+1} \rangle := \langle f_{CC}, f_{CD}, f_{DC}, f_{DD} \rangle$

For $k = N^*$ down to 0

For each (o_1, o_2) in $\{(C, C), (C, D), (D, C), (D, D)\}$

$F_{o_1 o_2}^k := p_{o_1 o_2}(u_{CC} + E_{CC}^{k+1}) + (1 - p_{o_1 o_2})(u_{CD} + E_{CD}^{k+1})$

$G_{o_1 o_2}^k := p_{o_1 o_2}(u_{DC} + E_{DC}^{k+1}) + (1 - p_{o_1 o_2})(u_{DD} + E_{DD}^{k+1})$

$E_{o_1 o_2}^k := \max(F_{o_1 o_2}^k, G_{o_1 o_2}^k)$

If $F_{o_1 o_2}^k \geq G_{o_1 o_2}^k$, then $m_{o_1 o_2}^k := C$

If $F_{o_1 o_2}^k < G_{o_1 o_2}^k$, then $m_{o_1 o_2}^k := D$

End For

End For

Return m_{ab}^0

Figure 4.5: The procedure for computing a recommended move for the current iteration.

In the competition, we set $N^* = 60$, $f_{CC} = 3$, $f_{CD} = 0$, $f_{DC} = 5$, and $f_{DD} = 1$.

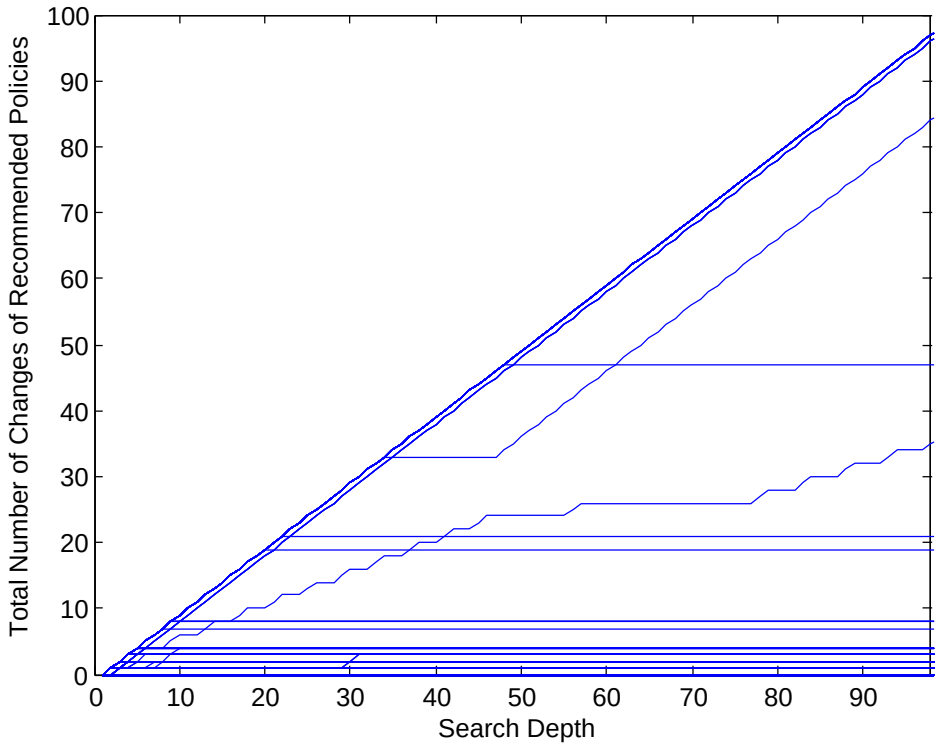

Figure 4.6: The total number of changes of recommended policies generated by MoveGen for each hypothesized policy as the search depth increases.

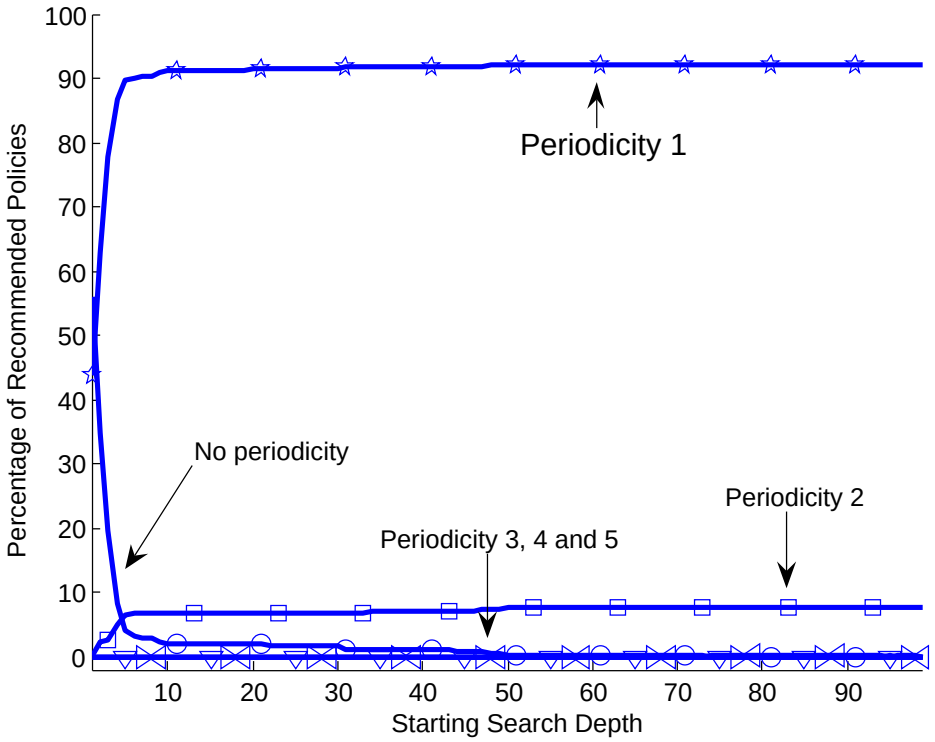

Figure 4.7: The distribution of the periodicity of the cycle of recommended policies versus the starting search depths. No periodicity means that there is no obvious cycle of recommended policies in the sequences of recommended policies generated starting from a given starting search depth.

Recommended Policy	Percentage
$DDDD$	61.16%
$CDDD$	7.52%
$DCDD$	0.00%
$CCDD$	0.64%
$DDCD$	0.16%
$CDCD$	0.96%
$DCCD$	0.00%
$CCCD$	0.64%
$DDDC$	3.68%
$CDDC$	7.52%
$DCDC$	0.00%
$CCDC$	1.28%
$DDCC$	0.64%
$CDCC$	2.24%
$DCCC$	0.00%
$CCCC$	5.76%

Figure 4.8: The percentage of recommended policies returned by the MoveGen procedure. The search depth is 100. Each recommended policy is represented by four characters $m_1m_2m_3m_4$, which means that the recommended policy is $\{(C,C) \rightarrow m_1, (C,D) \rightarrow m_2, (D,C) \rightarrow m_3, (D,D) \rightarrow m_4\}$. This table excluded the hypothesized policies with which the MoveGen procedure returns a sequence of recommended policies that change as the search depth increases.

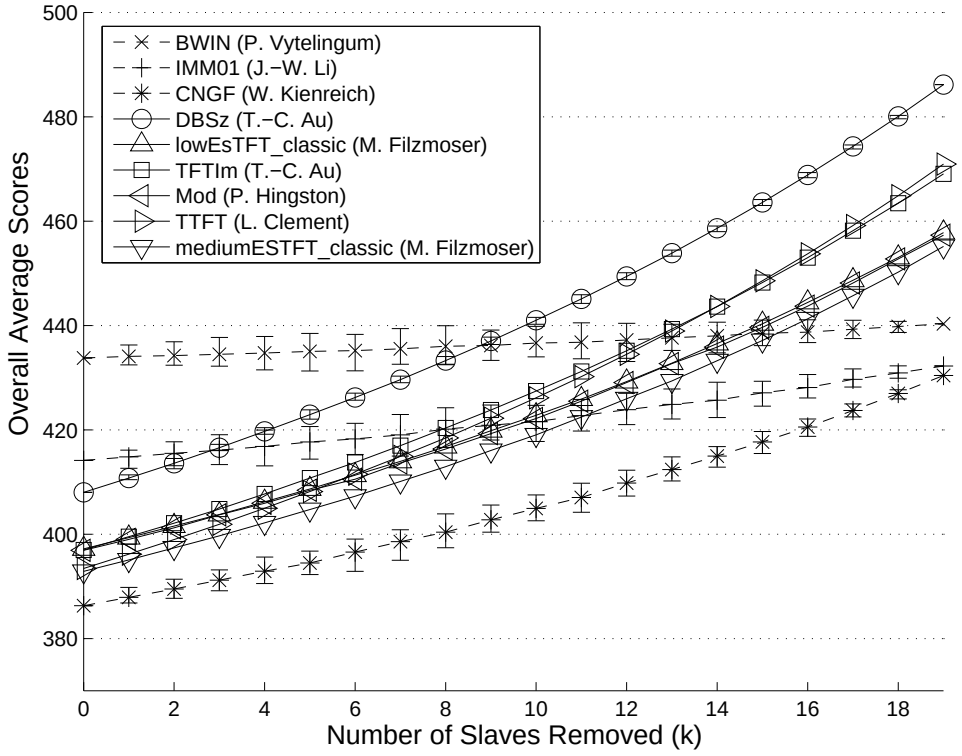

Figure 4.9: The overall average scores of selected programs versus the number of slaves removed from the tournament.

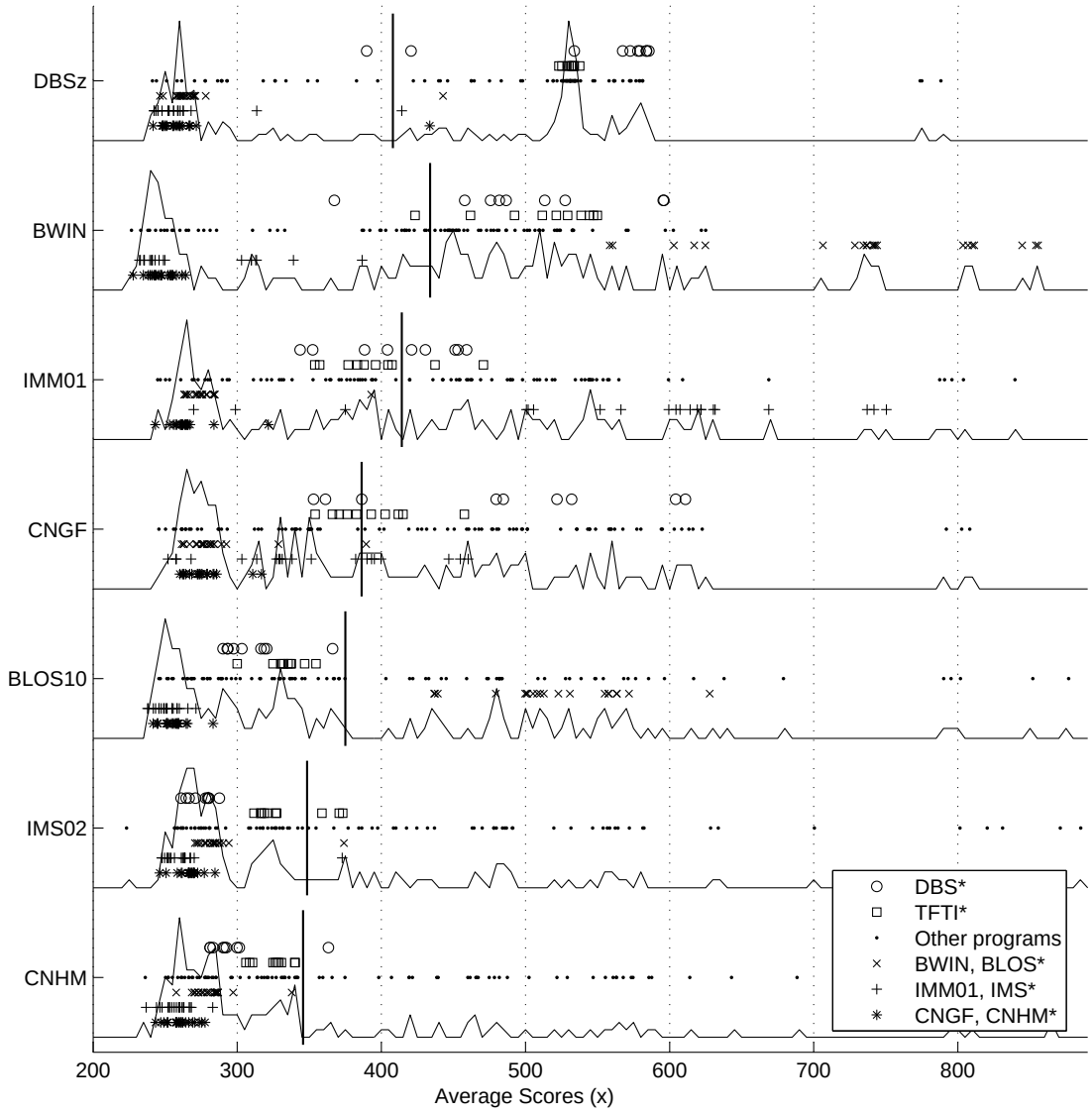

Figure 4.10: Density plots of the average scores of selected programs, overlapped with dot plots of the average scores. The vertical lines mark the overall average scores of the programs.

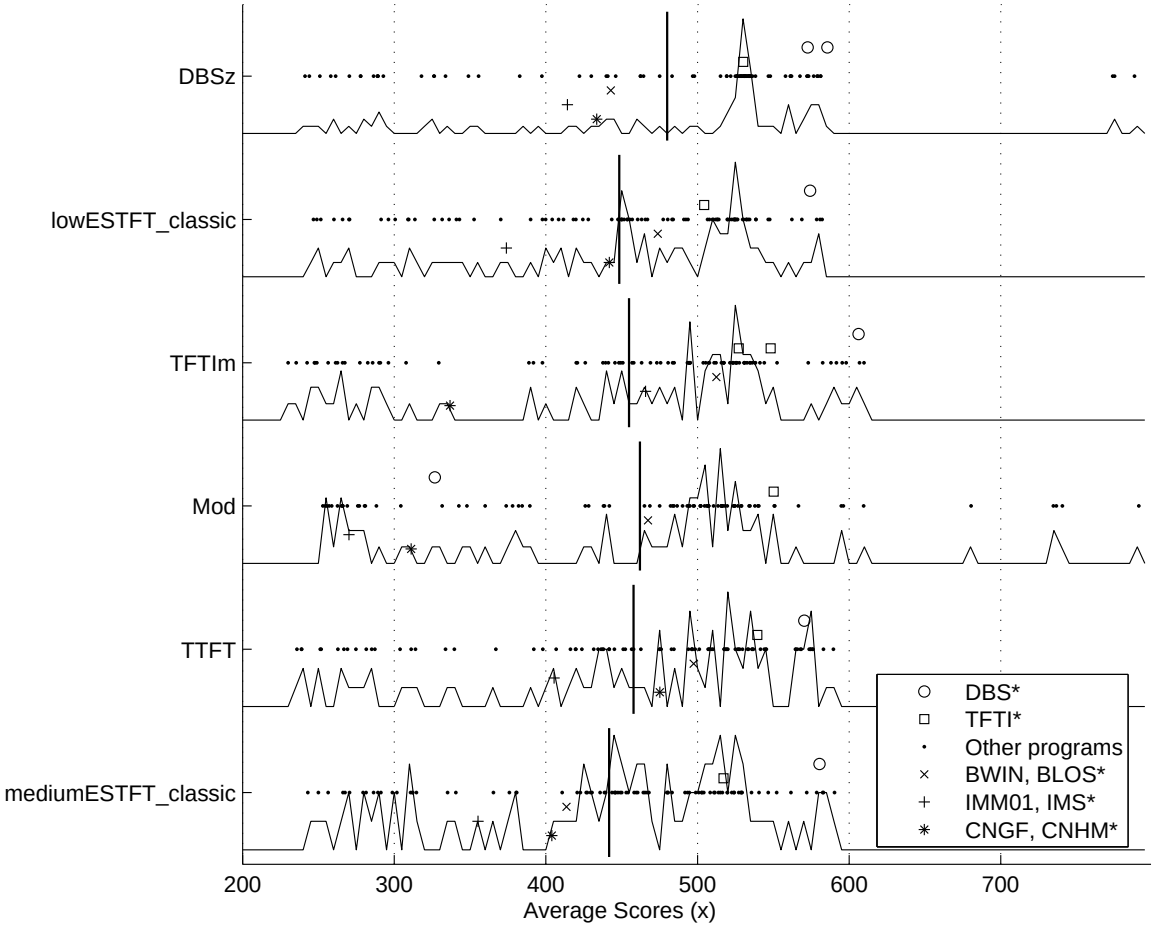

Figure 4.11: Density plots of the average scores of selected programs, overlapped with dot plots of the average scores. The slave programs and the programs submitted by the author, except DBSz and TFTIm, are excluded.

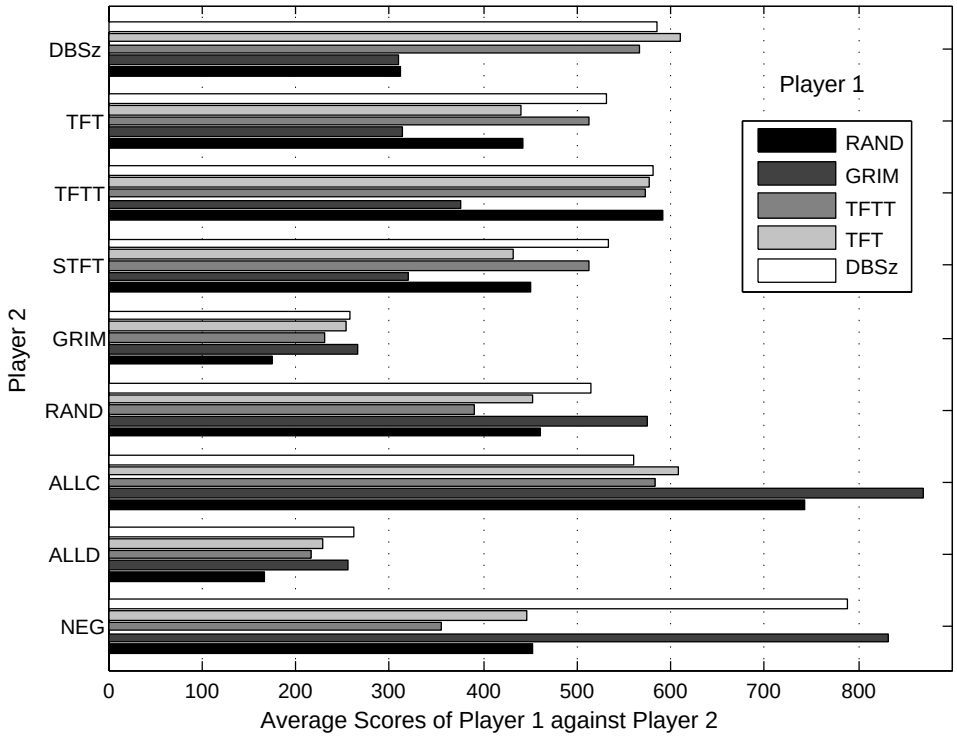

Figure 4.12: Average scores of the selected programs when played against DBSz and the programs provided by the organizer of the competition.

Chapter 5

Symbolic Noise Filter

Symbolic noise detection (SND) has been shown to be highly effective in the Noisy Iterated Prisoner's Dilemma, in which an action can accidentally be changed into a different action. This chapter evaluates this technique in two other 2×2 repeated games: the Noisy Iterated Chicken Game (ICG) and the Noisy Iterated Battle of the Sexes (IBS).

We present a generalization of SND that can be wrapped around *any* existing strategy. To test its performance, we organized ICG and IBS tournaments in which we solicited several dozen strategies from different authors, and we tested these strategies with and without our SND wrapper. In our tests, SND identified and corrected noise with 71% accuracy in the ICG, and 59% accuracy in the IBS. We believe the reason why SND was less effective in the ICG was because of a tendency for IBS strategies to change more frequently from one pattern of interactions to another, causing SND to make a higher number of wrong corrections. This leads us to believe that SND will be more effective in any game in which strategies often show a stable behavior.

5.1 Introduction

The performance of multiagent systems often depends on the robustness of interaction among agents. But errors can occur during the interaction, and that could break the premises the agents make about their interaction. This problem is compounded by

the fact the agents are self-interested and do not completely trust each other; agents can no longer trust each other because of the mistakes that the other agents make, albeit the mistakes are not intentional but accidental. How to cope with such mistakes is a critical factor in the maintenance of cooperation among agents.

Our previous work on the study of this issue focus on a famous normal-form game called the *Iterated Prisoner's Dilemma* (IPD), which is well known as an abstract model for studying cooperative behavior between two self-interested parties. We studied an important variant of the IPD is the *Noisy* IPD, in which there is a small probability, called the *noise level*, that accidents will occur [45]. Strategies that do quite well in the ordinary (non-noisy) IPD may do quite badly in the Noisy IPD [5, 11, 12, 45, 46, 47]. For example, if two players both use the well-known Tit-For-Tat (TFT) strategy, then an accidental defection may cause a long series of defections by both players as each of them punishes the other for his non-intentional defecting.

In Chapter 4, we have shown that a technique called *symbolic noise detection* (SND) has been shown to be quite effective in coping with noise in the Noisy IPD [2, 1]. The basic idea is to use a deterministic model of the other player to identify actions affected by noise during a game. In Category 2 of the 20th Anniversary IPD competition,[1] seven out of nine programs using symbolic noise detection are among the top ten. They lost only to a group of programs that work in a conspiracy to push one program to the top by giving as many points as possible to this program while sacrificing the performance of the

[1]The results of the competition can be found on the competition's homepage at `http://www.prisoners-dilemma.com`.

rest.[2]

It is natural to ask how helpful SND can be in other kind of games, and whether there are particular kinds of games in which it is especially helpful. Studying SND is important since to the best of our knowledge there is no other general procedure that can handle noise efficiently in such games. This chapter addresses these questions by studying SND in the noisy, repeated version of two other 2×2 games: the Game of Chicken [23, 60] and the Battle of the Sexes [42]. The Chicken Game models the situations in which two self-interested parties compete for a resource, but if neither of them concedes, both of them could get none of the resource. A typical example of this type of game is one that two people drive head-to-head towards each other at a very high speed, so that the first driver who swerves is the loser. The Battle of the Sexes models the situations in which two parties need to coordinate with each other to accomplish a task, but they favor different actions. For example, a husband and a wife prefer to go to a football game and an opera, respectively. However, if they go to different places, both of them would not be happy.

We choose these games because they model many interesting real-life social situations in which dilemma occurs. Solutions to these games have practical implications. Moreover, both the Chicken Game and the Battle of the Sexes relates to IPD by an appropriate scaling of the payoff matrix [17]. Thus, these games facilitate our studies of how well SND works as the parameters of the payoff matrix changes.

[2]The other two programs in the top nine used a very different strategy called the *master-and-slaves* strategy. The rules of the competition allowed each participant to submit up to 20 programs, and some participants submitted 20 programs that operated conspirators in which 19 programs (the "slaves") sacrificed their own performance in order to feed as many points as possible to the 20th program (the "master").

Our approach for evaluating SND was to organize tournaments similar to the past IPD tournaments. We organized two tournaments called the *the Noisy Iterated Chicken Game (ICG)* and *the Noisy Iterated Battle of the Sexes (IBS)*, and asked students of an AI class to participate. We also devised a wrapper that can be placed on each of the students' programs. The function of the wrapper is to correct any observed action that is regarded to be affected by noise according to the principle of SND. Then we put the wrapper around students' programs and repeated the tournaments. Our objective was to compare the performance of the strategies (i.e., student's programs) before and after using the wrapper. In particular, we conducted experiments to study the relationships of three variables: (1) the average score of a strategy, (2) the difference of the average scores of a strategy before and after using the wrapper, and (3) the accuracy of correction made by the wrapper.

The main points of this chapter are:

- We provide a general procedure called the *Naïve Symbolic Noise Filter* (NSNF) that can be placed around any existing strategy.

- In our competitions, NSNF was highly accurate in predicting the other player's next moves—96% and 93% of predictions are correct in ICG and IBS, respectively. Not all of these predictions would prompt NSNF to make corrections; NSNF corrects an observed action only when the corresponding prediction is different from the observed action. NSNF also did a decent job in correcting actions affected by noise—out of all corrections made by NSNF, 71% and 59% of them successfully rectified actions that have actually affected by noise in ICG and IBS, respectively.

- In both ICG and IBS, NSNF increased the scores of most programs, and the average increase was higher in ICG than in IBS. One reason for this is that NSNF often has a higher accuracy of noise correction in ICG.

- If we look at each strategy individually, the accuracy in correction does not strongly correlate with the increases in average scores of the strategies due to the use of NSNF. Some strategies, especially those in IBS, actually performed worse with NSNF than without it.

- In both games, strong players and weak players behave quite differently—strong players are more exploitive and they choose defect frequently during a game. However, in ICG this exploitive behavior causes the other player to exhibit a more clear behavior, but in IBS this does not. We explain this phenomenon via certain characteristics of decision making process of the strategies and the structure of the payoff matrix.

5.2 Our Hypothesis

Symbolic noise detection is a principle for identifying which the other player's actions has been affected by noise, using a deterministic model of the other player's behavior learnt from the current game history. This idea can be summarized as follows.

- Build a deterministic model of how the other player behaves.

- Watch for any *deviation* from the deterministic behavior predicted by the model.

- If a deviation occurs, check to see if the inconsistency persists in the next few iterations:

    1. If the inconsistency does not persist, assume the deviation is due to noise.

    2. If the inconsistency persists, assume there is a change in the behavior.

The clarity of behavior in IPD has already been discussed by Axelrod in his analysis of TFT [3]. In Chapter 4, it was argued that clarity of behavior is an important ingredient of long-term cooperation, and therefore most IPD agents exhibit deterministic behavior in tournaments. Thus, SND is effective in IPD because deterministic behavior is abundant in the IPD. However, if we use SND in other kind of games, the intention for cooperation may no longer be abundant; perhaps, cooperation may even be a undesirable behavior in other games. Therefore, an interesting question is to see in what kind of games SND would be effective.

In general, players in any zero-sum game tend to have little clarity in their behavior. For instance, in a game called RoShamBo [27], the objective is to predict the opponent's decision, and therefore players tried hard to conceal their thought patterns. Likewise, it is often hard for chess players to predict the opponent's next move with a high degree of certainty that is large enough for SND to be effective. In some non-zero-sum games, however, deterministic behaviors is more plentiful. It would be beneficial if we have a way to predict the amount of clarity in the player's behavior by just looking at the structure of a game. But this feat is currently out of our reach.

To summarize, our hypothesis is

Hypothesis 1 *Symbolic noise detection will be effective in non-zero-sum games in which strategies often exhibit deterministic behavior.*

We will evaluate this hypothesis by using the Noisy ICG and the Noisy IBS tournaments. [3]

5.3 The Noisy ICG Tournament and The Noisy IBS Tournament

Our tournaments are similar to the Axelrod's IPD tournaments and the 2005 IPD tournament, except that the payoff matrices are the following ones:

Chicken Game's Payoff Matrix		Player 2	
		Cooperate	Defect
Player 1	Cooperate	$(4, 4)$	$(3, 5)$
	Defect	$(5, 3)$	$(0, 0)$

Battle of the Sexes' Payoff Matrix		Husband	
		Football	Opera
Wife	Football	$(1, 2)$	$(0, 0)$
	Opera	$(0, 0)$	$(2, 1)$

In order to allow a program to play the roles of both husband and wife in the IBS, we will use the following modified payoff matrix in the IBS:

[3] A similar but weaker hypothesis is "Symbolic Noise detection will be effective in non-zero-sum games where there are rewards for deterministic behavior." This hypothesis is weaker because there can be games in which deterministic behavior is not rewarded but still abundant.

Battle of the Sexes' Modified Payoff Matrix		Player 2	
		Defect	Cooperate
Player 1	Cooperate	$(1, 2)$	$(0, 0)$
	Defect	$(0, 0)$	$(2, 1)$

At first glance, the modified Battle of the Sexes' payoff matrix seemed to be different from the usual one. In fact, the difference is just the labels in the matrices; we can obtain the modified payoff matrix by labeling Wife as Player 1, Husband as Player 2, Wife's Football as Cooperate, Wife's Opera as Defect, Husband's Football as Defect, and Husband's Opera as Cooperate. According to the modified payoff matrix, Defect is always more favorable than Cooperate for both Player 1 and Player 2. In order for the players to coordinate with each other, the players has to choose different actions in the modified payoff matrix.

A *game* consists of a finite sequence of *iterations*. In each iteration, two players, namely Player 1 and Player 2, play an ordinary Chicken Game or the Battle of the Sexes. At the beginning of a game, each player knows nothing about the other player, and does not know how many iterations he will play. In each iteration, each player chooses a *move*, which is either cooperate (C) or defect (D). A move is also called an *action*. After both players choose a move, noise may occur and alter the moves—changing 'Cooperate' to 'Defect', or 'Defect' to 'Cooperate'. If noise occurs and changes a move, the other player would see the altered move, not the move originally chosen by the player.

To distinguish the moves chosen by the players from the moves eventually seen by the other players, we call the former the *decisions* and the latter the *physical actions*.

Suppose the decisions of Player 1 and Player 2 in an iteration are a and b, respectively. Then the *decision pair* of this iteration is a pair of decisions (a, b), and the *interaction pair* is a pair of physical actions (a', b'), where, (1) $a' = \{C, D\} \setminus a$ if a has been affected by noise, or $a' = a$ if otherwise, and (2) $b' = \{C, D\} \setminus b$ if b has been affected by noise, or $b' = b$ if otherwise.

Noise has the following characteristics.

- The *noise level*, the probability that noise occurs and affects a move, is 10%. Each action has an equal probability of being affected by noise, and the occurrences of noise are independent of each other.

- If Player 1 chooses cooperate but noise changes his action to defect, then (1) Player 2 sees that Player 1's action is defect, and does not know that Player 1 originally chooses cooperate; and (2) Player 1 also does not know that his action has been affected by noise—after Player 1 chooses cooperate there is no way for Player 1 to tell whether his action has been affected by noise. In short, Player 1 knows a and b', and Player 2 knows a' and b; Player 1 does not know a' and b, and Player 2 does not know a and b'.

- The payoff of an iteration is determined by the interaction pair (a', b'), not the decision pair (a, b). But this payoff is not announced to the players, and the players cannot compute the payoff since Player 1 does not know a' and Player 2 does not know b'.

The score of a player in a game is the sum of the payoff he accumulated in all the iterations of the game.

5.4 Naïve Symbolic Noise Filter

Our next step is to supplement the collected strategies with symbolic noise detection. Our approach is to develop a procedure called *Symbolic Noise Filter* (SNF) that can be placed around *any* existing strategy to filter the input (the observed action of the other player) to a strategy using SND. The benefit of this approach is that SNF, once implemented, is readily applicable to all strategies. On the other hand, this is helpful to our study because the noise filter of all strategies are the same, and it is easier to compare their performance than the custom-made noise filters.

Our study will use a simplified version of SND called *Naïve Symbolic Noise Detection* (NSND), which is like SND but does not defer judgment about whether a derivation is due to noise or not—it immediately regards a derivation is due to noise when a derivation is detected. This is different from the full-strength SND proposed in Chapter 4, which utilizes the information *before and after* a derivation to improve the accuracy of noise detection. The benefit of NSND over the full-strength version of SND is that its implementation is much simpler than SND's—there is no need to remember when derivation occurs and adjust the underlying move generator when a change of behavior is detected. Of course, the drawback is that the accuracy of NSND can be lower than SND's. But this deficiency is outweighed by its simplicity, which is important for our wrapper-approach to SND.

SNF based on NSND is called *Naïve Symbolic Noise Filter* (NSNF). Figure 5.1 illustrates the architecture of NSNF, and Figure 5.2 outlines the pseudo-code of our implementation of NSNF. Before we discuss the NSNF procedure, let us give the definitions

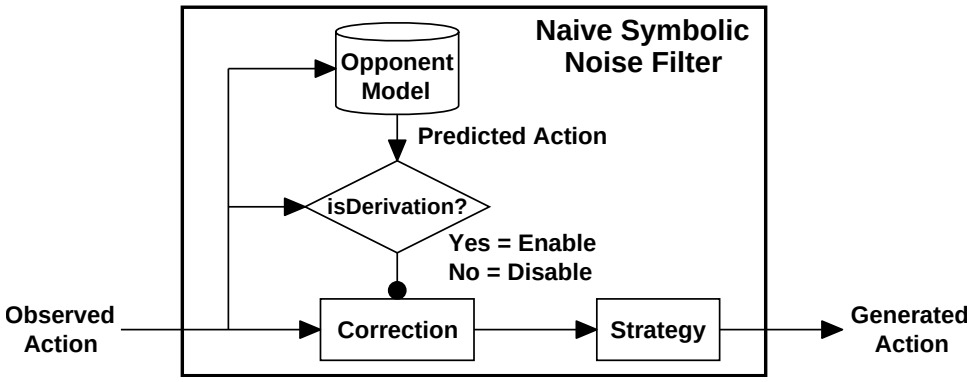

Figure 5.1: Naïve Symbolic Noise Filter (NSNF).

of various terms. A *history* τ of length k is a sequence of action pairs of all iterations up to and including iteration k. We write $\tau = \langle (a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k) \rangle$. A *strategy* is a mapping $\psi : \mathcal{H} \to \{C, D\}$, where $\mathcal{H} = \langle (C, C), (C, D), (D, C), (D, D) \rangle^*$ is the set of all possible histories. A *condition* $\mathsf{Cond} : \mathcal{H} \to \{\mathsf{True}, \mathsf{False}\}$ is a mapping from histories to Boolean values. For any action a and b, we define $\mathsf{Cond}_{a,b}$ to be a condition such that $\mathsf{Cond}_{a,b}(\tau) = \mathsf{True}$ if and only if either (1) $k \geq 1$, $a_k = a$ and $b_k = b$, (where $k = |\tau|$), or (2) $k = 0$ and $a = b = C$. In other words, $\mathsf{Cond}_{a,b}(\tau)$ is true when the last action pair of τ is (a, b). A *deterministic rule* is $\mathsf{Cond} \to b$, where Cond is a condition and b is an action.

The NSNF procedure in Figure 5.2 has two input parameters: a strategy ψ and a positive integer promotion_threshold. The strategy ψ takes a history as its input and generate an action a. The promotion threshold is to control the likelihood of picking up a deterministic behavior. Increasing the promotion threshold reduces the chance that the function **isDerivation** mistakes a random behavior as a deterministic behavior, but ignores certain genuine deterministic behavior. In our tournaments, the promotion threshold is 3.

The procedure has two variables about the current game histories: the recorded his-

tory $\tau_{recorded}$ and the filtered history $\tau_{filtered}$. The recorded history $\tau_{recorded}$ is a sequence of action pairs, each of them is the decision a of the strategy ψ and the other player's physical action b' in an iteration. The filtered history $\tau_{filtered}$ is like $\tau_{recorded}$, except that the other player's physical actions in $\tau_{filtered}$ have been "corrected" if the procedure decides that the physical actions have been affected by noise. The history seem by the strategy ψ is $\tau_{filtered}$ rather than $\tau_{recorded}$ (Line 4 in Figure 5.2).

This implementation of NSNF is simpler than the implementation of the Derived Belief Strategy (DBS) in Chapter 4, because it does not explicitly maintain the opponent model (i.e., the hypothesized policy in DBS) but deduces the deterministic rules on demand. The procedure simply searches backward on the current recorded history to locate a deterministic rule $\mathsf{Cond}_{a_j,b'_j} \rightarrow b_{next}$, where (a_j, b'_j) is the second to the last action pair of the current recorded history (Line 13 to Line 23). If NSNF finds such a rule, NSNF will make a prediction based on b_{next}. If b'_{j+1}, the last observed physical action of the other player, is the same as b_{next}, no derivation is observed; otherwise, NSNF observes a derivation and regards b'_{j+1} has been affected by noise. Then it corrects b'_{j+1} using the invert function, which returns C if $b'_{j+1} = D$ and returns D if $b'_{j+1} = C$.

As an example, suppose both Player 1 (P_1) and Player 2 (P_2) uses a strategy called Tit-For-Tat (TFT), which starts with Cooperate and then repeats the other player's action in the previous iteration. If both P_1 and P_2 do *not* use the Naïve Symbolic Noise Filter, a possible history can be:

Iteration:	1	2	3	4	5	6	7	8	9	10
Physical Actions of P_1:	C	C	C	<u>D</u>	C	D	C	D	D	D

NaïveSymbolicNoiseFilter(ψ, promotion_threshold)

1. $\tau_{recorded} := \emptyset$ // *the recorded history*
2. $\tau_{filtered} := \emptyset$ // *the filtered history*
3. Loop until the end of the game
4. $a := \psi(\tau_{filtered})$ // ψ *is the base strategy*
5. Output a and get the other player's physical action b'
6. $\tau_{recorded} := \tau_{recorded} \circ \langle (a, b') \rangle$
7. If **isDerivation**($\tau_{recorded}$) = True, then
8. $\tau_{filtered} := \tau_{filtered} \circ \langle (a, \text{invert}(b')) \rangle$
9. Else
10. $\tau_{filtered} := \tau_{recorded} \circ \langle (a, b') \rangle$

Function **isDerivation**($\tau_{recorded}$)

11. Let $\tau_{recorded}$ be $\langle (a_1, b'_1), (a_2, b'_2), \ldots, (a_{k+1}, b'_{k+1}) \rangle$
12. Cond := Cond_{a_k, b'_k} ; $count := 0$
13. For $j = k - 1$ DownTo 1
14. If $\text{Cond}(a_j, b'_j)$ = True, then
15. If $count = 0$, then
16. $b_{next} := b'_{j+1}$; $count := count + 1$
17. Else
18. If $b'_{j+1} = b_{next}$, then
19. $count := count + 1$
20. If $count \geq$ promotion_threshold, then
21. If $b_{next} = b'_{k+1}$, return False, else return True
22. Else
23. Return False

Figure 5.2: The pseudo-code of the Naive Symbolic Noise Filter. The function invert(b')

Physical Actions of P_2:	C C C C D C D <u>D</u> D D

Here, the underlined characters refer to the physical actions that have been affected by noise. We can see that in the fourth iteration, the decision of P_1 was originally C, but was changed to D due to noise. Then this triggered the retaliation of P_2 and started a long sequence of mutual defection between the two players. The situation worsened when noise occurred again in the eighth iteration.

But if Player 2 uses Naïve Symbolic Noise Filter, the above history would become:

Iteration:	1 2 3 4 5 6 7 8 9 10
Physical Actions of P_1:	C C C <u>D</u> C C C C D C
Physical Actions of P_2:	C C C C C C C <u>D</u> C C

At the end of the third iteration, NSNF can readily identify the deterministic rule $\mathsf{Cond}_{C,C} \rightarrow C$, because the rule is true in the first three iterations and we set $\mathsf{promotion\_threshold} = 2$. In the fourth iteration, NSNF saw a derivation from the rule, causing P_2 to correctly consider D in the fourth iteration as being affected by noise. Therefore, P_2 did not retaliate in the fifth iteration. In the ninth iteration, P_1 retaliates for the defection it observes in the eighth iteration since P_1 does not use NSNF. But P_2 did not defect in the tenth iteration, because the deterministic rule $\mathsf{Cond}_{C,C} \rightarrow C$ had held repeatedly since the fifth iteration.

In this example, NSNF prevented mutual defection in two different occasions, helping both players to maintain cooperation. Furthermore, NSNF can be effective even if only one player is using it.

5.5 Tournament Setup

We have asked students of an advanced-level AI class to participate in two tournaments: the Noisy ICG tournament and the Noisy IBS tournament. There are 37 students in the class, and all of them have submitted programs to both tournaments. We told students that the noise level is 10% and the number of iterations of each game is at least 50. Thus, students do not know the exact number of iterations, which is 300. The ICG tournament is held first. Before the IBS tournament, students were informed about the ranking of their programs in the ICG tournament. This information should not affect the IBS tournaments since the tournaments are different. In each tournament, students were given approximately 2 weeks to complete their programs.

First, we conducted experiments by repeating the ICG tournament (without NSNF) 1000 times as follows. Let Λ_{ICG} be the set of all programs for ICG. For any pair $\lambda_i, \lambda_j \in \Lambda_{ICG}$ of programs, λ_i has a chance to play with λ_j in a tournament. Notice that λ_j can be λ_i itself. The average score of λ_i is the average of the scores of λ_i in the 37000 ICG games in which λ_i participated. We also collected statistics about λ_i such as the number of defection, etc.

Second, for each $\lambda_i \in \Lambda_{ICG}$, we augmented λ_i with NSNF and denote the augmented program by $\hat{\lambda_i}$. Then for each $\lambda_j \in \Lambda_{ICG}$, we set $\hat{\lambda_i}$ to be Player 1 and play against λ_j for 1000 times. Notice that λ_j can be λ_i itself (but without NSNF). The average score of $\hat{\lambda_i}$ is the average of the scores of λ_i in the 37000 ICG games in which Player 1 is $\hat{\lambda_i}$. We also collected statistics about $\hat{\lambda_i}$ and its NSNF such as the average number of iterations in which NSNF correctly predicted the occurrence of noise in a game, etc.

The IBS tournaments were also conducted in the same way.

5.6 Experimental Analysis of NSNF

Our analysis is divided into three sections. The first section presents some basic statistics about the tournaments. Our focus is on three variables: (1) the average scores, (2) the increases in average scores due to NSNF, and (3) the accuracy of correction of NSNF. The second section tries to explain the relationships of these variables by measuring the frequency of change of decisions and the frequency of defects made by strategies. The third section compares the distribution of decision pairs in ICG and IBS.

5.6.1 Basic Statistics

In this chapter, average scores are normalized—a normalized average score is equal to the average score divided by the maximum possible scores of a game (1500 in ICG and 600 in IBS). This allows us to put data from different tournaments in the same graph.

5.6.1.1 Average scores

Figure 5.3 shows the normalized average scores of the strategies, with and without using NSNF, in ICG and IBS. The normalized average scores are ordered according to the ranks of the strategies in the original (without NSNF) tournaments. This shows that the differences of the normalized average scores of the best strategy and the worst strategy are small: 0.144 for ICG and 0.262 for IBS. Therefore, a small change in the average score would have a decisive effect to the rank of a strategy. On average, an increase of

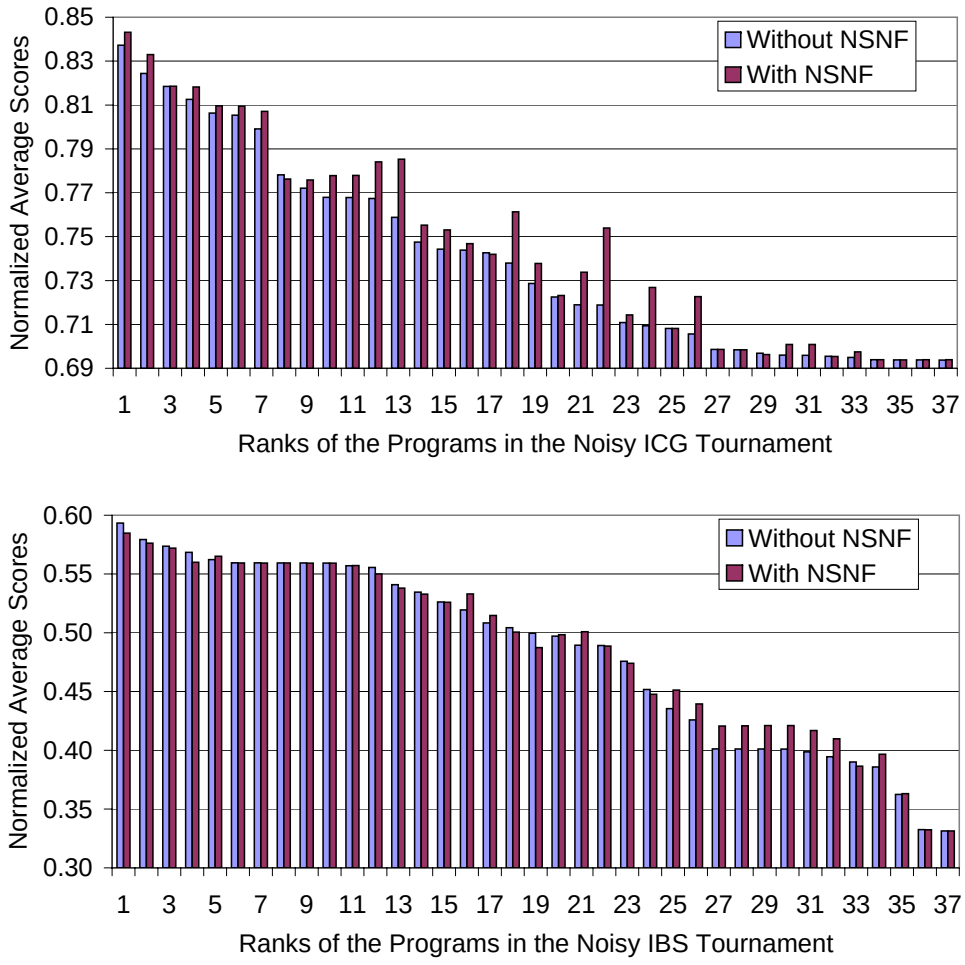

Figure 5.3: Normalized average scores.

0.0039 and 0.0071 in the normalized average score can change the rank of a program in ICG and IBS, respectively.

Table 5.1 presents *the overall normalized average scores*, the averages of the normalized scores of all strategies in a tournament. This result shows that NSNF does increase the overall normalized average scores of both ICG and IBS, and the increase for ICG is larger than the increase for IBS.

We computed the *the increase in normalized average scores* of a strategy due to

Table 5.1: The overall normalized average scores (the average of the normalized average scores of all strategies in Figure 5.3.)

	Without NSNF	With NSNF	Difference
ICG	0.7407	0.7475	0.0068
IBS	0.4834	0.4869	0.0035

NSNF by subtracting the normalized average score of a strategy without NSNF from the normalized average score of a strategy with NSNF, using the data in Figure 5.3. Then we present how the increases in normalized average scores relates to the normalized average scores in Figure 5.4. The relationship for ICG is quite different from the relationship for IBS. In ICG, there is no obvious relationship between the increases in normalized average scores and the normalized average. However, in IBS, the increases in normalized average scores decrease as the normalized average scores increase. In addition, some strategies do not have an increase but a decrease in the normalized average scores. This is essentially true for strategies in IBS, especially those that originally have a high average score.

5.6.1.2 Accuracy of correction

We measured various statistics about the accuracy of NSNF, and the results is summarized in Table 5.2. In this table, a true positive is referred to a situation in which NSNF correctly predicts that an action is affected by noise. A true negative is a situation in which NSNF correctly predicts that an action is not affected by noise. False positives and false negatives are situations in which NSNF make wrong predictions (noise occurs

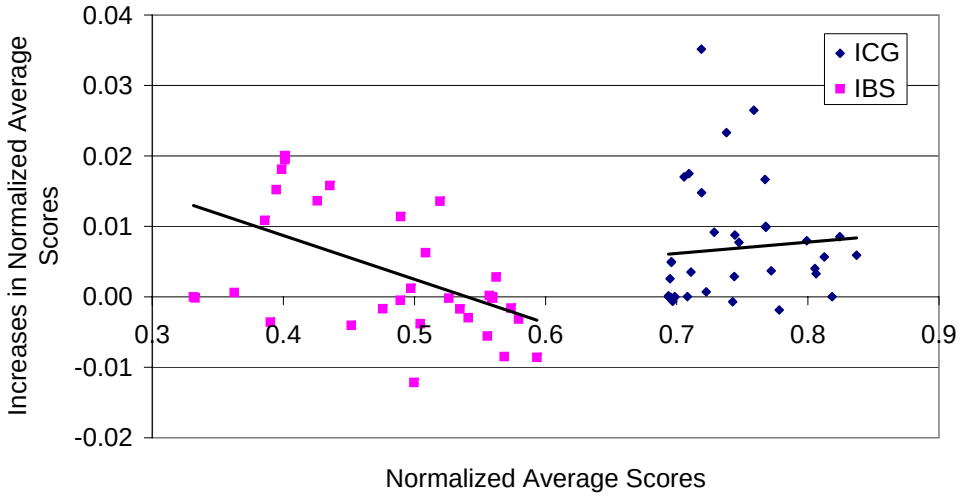

Figure 5.4: Increases in normalized average scores due to NSNF versus normalized average scores. Notice that 11 points are clustered at (0.7, 0.00).

and noise does not occur, respectively). The accuracy of prediction, which is equal to $(n_t^+ + n_t^-)/n_p$, is a measure of the likelihood that NSNF makes correct prediction. This is different from the accuracy of correction, which is equal to $n_t^+/(n_t^+ + n_f^+)$ and is a measure of the likelihood that NSNF makes correct prediction about the occurrence of noise. The effective number of correction is equal to $n_t^+ - n_f^+$.

Let us discuss the accuracy of NSNF in ICG first. In each iteration, NSNF predicts the next move of the other player by searching backward to see whether a deterministic behavior pertaining to the current condition of the current iteration can be picked up. Thus, a prediction is made only if an appropriate deterministic behavior is found. In ICG, NSNF were able to pick up deterministic behavior and made predictions in 188.86 out of 300 iterations; this is an interesting result, because this shows that there is a significant amount of deterministic behavior in ICG (and in IBS too).

Our statistics showed that most of these 188.86 predictions were correct—the ac-

Table 5.2: Accuracy of Predictions and Corrections of NSNF.

	ICG	IBS
Avg. no. of actions affected by noise	30.00	30.01
Avg. no. of predictions made (n_p)	188.86	172.86
Avg. no. of derivations detected	25.43	27.14
Avg. no. of corrections made	25.43	27.14
Avg. no. of false negatives (n_f^-)	0.82	1.23
Avg. no. of false positives (n_f^+)	7.36	11.08
Avg. no. of true negatives (n_t^-)	162.61	144.49
Avg. no. of true positives (n_t^+)	18.06	16.06
Accuracy of predictions	95.67%	92.88%
Accuracy of corrections	71.04%	59.19%
Effective no. of correction	10.70	4.98

curacy of prediction is as high as 95.67%. Only 4.33% of these predictions were false positives (predicting there is noise when there is no noise) and false negatives (predicting there is no noise when there is noise). This indicates that the deterministic model of the other player's behavior has a high predictive power indeed.

Nonetheless, the usefulness of these deterministic behaviors depends on whether they can be used to detect noise. Thus, our concern is how often NSNF corrects actions that have actually affected by noise. On average, out of 25.43 corrections made by NSNF, 18.06 of them were correct (i.e., they are true positives). Therefore, the accuracy of corrections is 71.04%. This is a decent accuracy; at least, it shows that NSNF were not making random correction: if we compare our result to the accuracy of a random noise filter, which randomly select 25 actions out of 300 observed actions and correct them and hence only about 2.5 of these corrections are correct (i.e., the accuracy is only 10%), the accuracy of NSNF is much higher.

On the other hand, NSNF made about 28.96% wrong corrections, changing about 7.36 actions when they should have not been changed. The result of these wrong corrections is similar to the introduction of noise into the observed actions (though the introduction is not random). According to this view, the effective number of correction is 10.70.

In IBS, NSNF's prediction is also highly accurate: it is as high as 92.88%. However, the number of true positives among these predictions is slight smaller than that in ICG, resulting in an 59.19% accuracy of correction. The effective number of correction is 4.98 only, which is less than half of the effective number of correction in ICG.

NSNF can be considered as a noise filtering device, causing a $10.70/300 = 3.57\%$

reduction of the noise level. We define the *effective noise level* for the other player's actions (not the player's own actions) to be the reduced noise level due to the use of NSNF. For example, the effective noise level in the ICG is $10\% - 3.57\% = 6.43\%$, while the effective noise the level in IBS is $10\% - 1.66\% = 8.34\%$. In IBS, NSNF's predictions are also moderately accurate: the accuracy of correction is 59.19%. But due to wrong predictions, the effective noise level is dropped by 1.66% only. This means that NSNF in IBS is less than half as effective as NSNF in ICG in the reduction of noise level.

One fact that has not been shown in Figure 5.2 is that NSNF makes almost the same number of corrections for every strategies. In ICG, the average number of corrections for a strategy is between 24 and 27; in IBS, the average number of corrections is between 25 and 30. This suggests we can rely on the accuracy of correction as a performance indicator of NSNF. [4]

5.6.1.3 Accuracy of correction vs Average Scores

Figure 5.5 presents how the average scores relate to the accuracy of correction. Surprisingly, a positive linear correlation is observed in ICG, but a negative linear correlation is observed in IBS (after excluding a few outliners due to some weak strategies in IBS).

[4] In the terminology of Information Retrieval, the accuracy of correction is equivalent to precision, and the number of correct correction is equivalent to recall. A good information retrieval system should have both high precision and high recall. Thus, we cannot measure the performance of an information retrieval system by precision (or recall) alone; it is necessary to strike a balance between precision and recall. But for NSNF, high precision implies high recall since the number of correction is roughly a constant. Therefore, we can use the accuracy of correction as a performance indicator of NSNF.

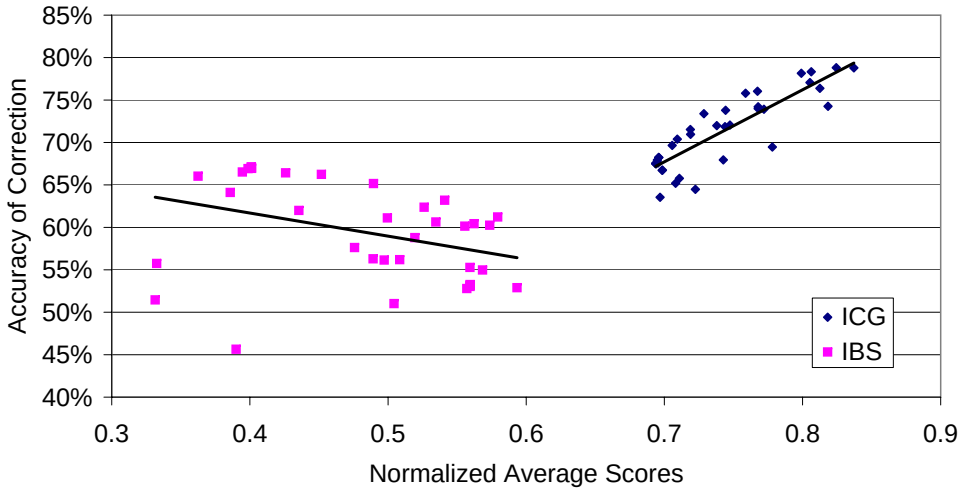

Figure 5.5: Accuracy of correction versus normalized average scores

Moreover, most strategies in ICG have a higher accuracy than all of the strategies in IBS.

But how the accuracy of correction relates to the increases in average scores? In Figure 5.6, we found no obvious relationship between these two variables. Even worse, around half of the strategies in IBS do not have an increase in their average scores no matter what their accuracy of correction are. But one thing we observe in both ICG and IBS is that none of the strategies with a low accuracy of correction (when compared to other strategies in the same tournament) has a higher-than-average increase in their normalized average scores. This is why the slopes of the lines in Figures 5.6 are positive.

5.6.2 Explanations via Characteristics of Decisions Making Process

This section offers explanations on (1) the relationship between the average scores and the accuracy of correction, and (2) the relationship between the average scores and the increases in average scores. From these, we can deduce the relationship between the

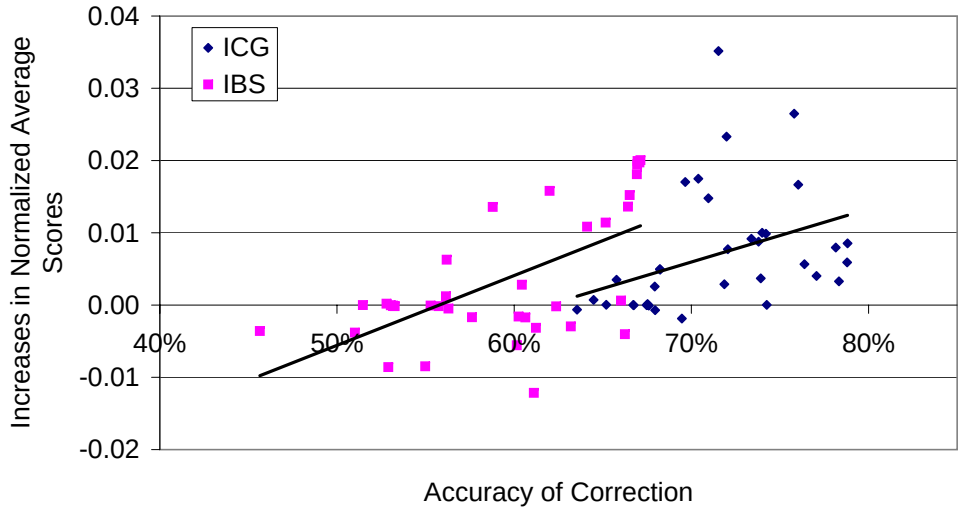

Figure 5.6: Increases in normalized average scores versus accuracy of correction

accuracy of correction and the increases in average scores.

Our explanations are based on two variables concerning decisions made by players: (1) the frequency of choosing defect (Figure 5.7), and (2) the frequency of change of the player's own decisions—from cooperate to defect, or vice versa—between two consecutive iterations (Figure 5.8).

5.6.2.1 Average scores vs accuracy of correction

We observe that the accuracy of correction increases with average scores in ICG in Figure 5.5. At the same time, we observe that the frequency of choosing defect increases with average scores in Figure 5.7. We believe this is not a coincidence; in fact, we can see that they are naturally linked together by looking at the structure of the Chicken Game's payoff matrix—if a player chooses defect, the other player is much better off choosing cooperate (and earning 3 points) rather than defect in order to avoid mutual defection that

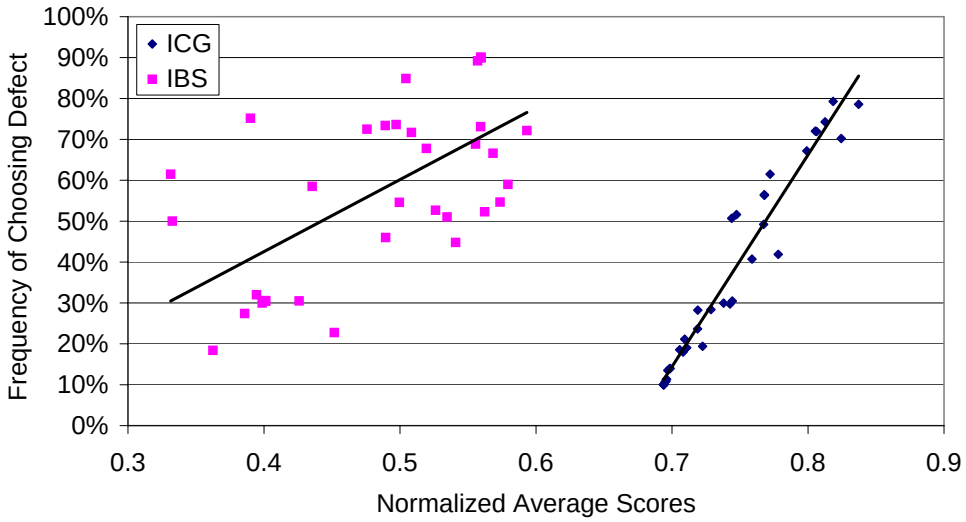

Figure 5.7: Frequency of choosing defect versus normalized average scores.

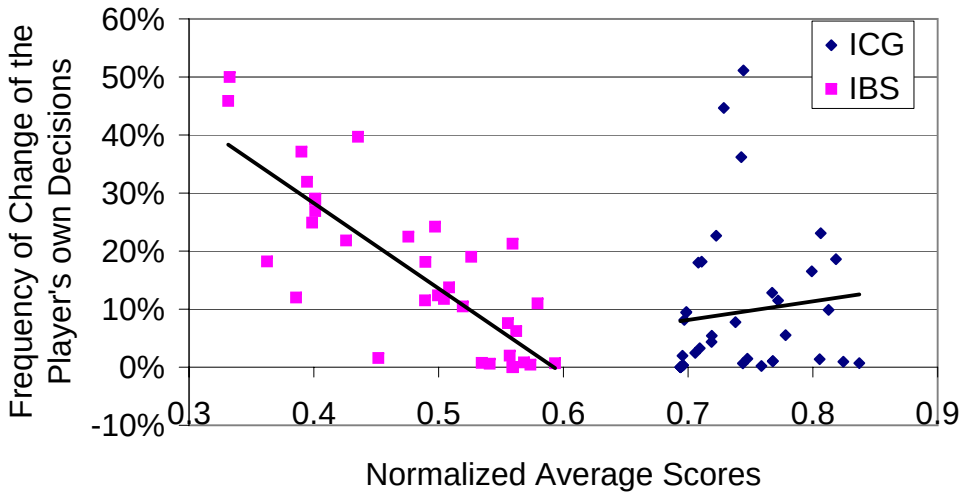

Figure 5.8: Frequency of changes of the player's own decisions versus normalized average scores.

gives 0 point; but if a player chooses cooperate, the choice of the other player would make little difference to the other player's payoff (4 points for choosing cooperate, and 5 points for choosing defect). Thus, if a player manages to choose defect frequently, the other player's behavior would become more deterministic, because he often chooses cooperate. The linear correlation in both Figure 5.5 and Figure 5.7 seems to strongly support our explanation.

We also observe a gentle decrease in the accuracy of correction as the average scores increase in IBS in Figure 5.5. At the same time, we observe that the frequency of choosing defect increases with the average scores in IBS in Figure 5.7. These figures suggest that strategies that defects frequently have a slight lower accuracy of correction than those that cooperate frequently. Our explanation for this is based on the following observations. First, the cost of mutual defection is small according to the payoff matrix—the player loses 1 point only if he chooses defect instead of cooperate in response to defect. This cost is much lower than the cost in ICG, which is 3 points. Hence, the other players in IBS are more willing to choose defect in response to defect. See the caption of Table 5.4 for evidence of this point. Second, weak players (who have low average scores) tend to be those who want to play fairly—having the same cumulative payoff as the other player's. By contrast, strong players (who have high average scores) tend to be exploitive—they often defect. Data in Figure 5.7 and Figure 5.8 seem to support this description of the characters of strong and weak players. When playing with strong players, weak players would often change their decisions frequently (as they are willing to defect in response to defect) as they struggle for fairness, causing a lower accuracy of the strong players' prediction of their behavior. But from a weak players' viewpoint, the strong player's

behavior is very clear.

5.6.2.2 Average scores vs increases in average scores

In Figure 5.4, we observe that there is no obvious relationship between the average scores and the increases of average scores due to NSNF in ICG, but a negative correlation is observed in IBS. A common feature of the data for ICG and the data for IBS in Figure 5.4 is that strong players often have a lower increase in average scores than other players. One reason for this is that strong players defect frequently according to Figure 5.7. More importantly, they defect no matter which move the other player chooses. Thus, strong players are less sensitive to noise because they often choose detect no matter what. Therefore, noise would cause less damage to these players than those that change their decisions frequently. NSNF would be less beneficial to strong players, as strong players do not behave much differently even after the correction of noise. On the other hand, some weak players rely on the establishment of certain interaction patterns with the other player. This process is prone to noise, and thus NSNF can help these players a lot.

5.6.3 Distribution of Decision Pairs

We would like to further investigate the general effect of NSNF in different kind of games by looking at the distribution of different decision pairs as shown in Table 5.3.

Traditionally, researchers use Nash equilibrium and its extensions to study repeated games; for example, they use Folk Theorems and their variants to answer questions pertaining repeated games [51]. These theorems, however, do not offer any prediction about

the distribution of strategies in a tournament. Many interesting phenomena have been observed in simulations, but cannot yet be explained completely by contemporary theories. A famous example is the emergence of cooperation in IPD [3]: about 31% of the interaction pairs in the Category 2 of 2005 IPD competition are (C, C) when there is no master-and-slaves strategy (Table 5.3). The similar emergence of cooperation was observed in our ICG tournaments—the most-frequent decision pairs turns out to be (C, C), rather than the two pure Nash equilibria of the Chicken Game ((C, D) and (D, C)). In IBS, however, the interaction pairs of strategies are often in two pure Nash equilibria of the Battle of the Sexes ((C, D) and (D, C)). But (D, D) is also a high-frequent decision pairs in IBS. We call (C, C) in ICG and (D, D) in IBS the *emergent decision pairs*, since they have a high frequency of occurrence, but they are not the pure Nash equilibria of the corresponding one-shot games.

NSNF affects the distribution of decisions pairs. In ICG, NSNF leads to a 5.75% increase of (D, C), most of them are converted from (C, C). In IBS, NSNF causes an 3.22% increase of (C, D), most of them are converted from (D, D). A striking similarity of these effects is that there is a decrease in the frequency of the emergent decision pairs—(C, C) in ICG and (D, D) in IBS. In future, we are interested to study whether this also occurs in the Noisy IPD and other kind of games as well.

We also look at the pattern of how often one decision pair changes to another in consecutive iterations. The numbers in Table 5.4 are computed by dividing the average number of the consecutive iterations that one decision pair change to another by 299, the total number of consecutive iterations in a game. Thus, the sum of all 16 numbers in a

Table 5.3: Distributions of different decision pairs. The IPD's data are collected from Category 2 of the 2005 IPD competition. *"All but M&S"* means all 105 programs that did not use master-and-slaves strategies, and *"all"* means all 165 programs in the competition. Note that the numbers for IPD are not the number of decision pairs but interaction pairs.

ICG	(C,C)	(C,D)	(D,C)	(D,D)
Without NSNF	38.26%	29.84%	29.83%	2.06%
With NSNF	33.35%	28.87%	35.58%	2.20%
Difference	-4.90%	-0.97%	5.75%	0.14%

IBS	(C,C)	(C,D)	(D,C)	(D,D)
Without NSNF	5.06%	35.91%	35.94%	23.09%
With NSNF	4.94%	39.13%	35.06%	20.87%
Difference	-0.11%	3.22%	-0.88%	-2.22%

IPD	(C,C)	(C,D)	(D,C)	(D,D)
all	13%	16%	16%	55%
all but M&S	31%	19%	19%	31%

Table 5.4: Frequency of change of decision pairs.

ICG without NSNF

From \ To	(C,C)	(C,D)	(D,C)	(D,D)
(C,C)	**30.89%**	3.56%	3.56%	0.26%
(C,D)	3.47%	**25.48%**	0.18%	0.70%
(D,C)	3.47%	0.18%	**25.48%**	0.70%
(D,D)	0.46%	0.63%	0.63%	**0.33%**

ICG with NSNF

From \ To	(C,C)	(C,D)	(D,C)	(D,D)
(C,C)	**26.45%**	3.22%	3.44%	0.24%
(C,D)	3.13%	**24.87%**	0.19%	0.68%
(D,C)	3.37%	0.18%	**31.12%**	0.91%
(D,D)	0.42%	0.61%	0.87%	**0.31%**

IBS without NSNF

From \ To	(C,C)	(C,D)	(D,C)	(D,D)
(C,C)	**0.56%**	1.17%	1.17%	2.15%
(C,D)	1.52%	**29.75%**	0.39%	4.26%
(D,C)	1.52%	0.39%	**29.77%**	4.26%
(D,D)	1.46%	4.67%	4.67%	**12.28%**

IBS with NSNF

From \ To	(C,C)	(C,D)	(D,C)	(D,D)
(C,C)	**0.60%**	1.46%	1.02%	1.87%
(C,D)	1.64%	**33.63%**	0.39%	3.47%
(D,C)	1.43%	0.41%	**29.14%**	4.07%
(D,D)	1.27%	3.71%	4.57%	**11.33%**

matrix is equal to 100%. Notice that the other players in IBS are much more willing to choose defect in response to defects; there is a $12.9\% = (0.39 + 4.26)/(1.52 + 0.39 + 29.77 + 4.26)$ of chance of choosing defect when the previous decision pair is (D, C), as opposed to $2.97\% = (0.18 + 0.70)/(3.47 + 0.18 + 25.48 + 0.70)$ of chance in ICG.

From Table 5.4, we see that there is a large number of changes to and from the emergent decision pairs in both tournaments. More precisely, in ICG, most changes of decision pairs starts from (C, C) or ends at (C, C), whereas in IBS, most changes of decision pairs starts from (D, D) or ends at (D, D). Perhaps this has something to do with the underlying mechanism that causes the emergence of these decision pairs. However, the flow in and out of these decision pairs drop after we augmented strategies with NSNF.

Table 5.5 shows how often strategies remain in the same decision pairs in consecutive iterations. First, strategies in ICG more often remain in the same decision pair than strategies in IBS. Perhaps this explains why the accuracy of correction is higher in ICG than in IBS (Figure 5.5). Second, strategies in both ICG and IBS are more likely to remain in the same decision pairs after using NSNF. This stabilizing effect may have caused the

Table 5.5: The sum of the diagonal entries in Table 5.4.

	Without NSNF	With NSNF	Difference
ICG	82.18%	82.75%	0.57%
IBS	72.36%	74.71%	2.35%

increases in the overall average scores as shown in Table 5.1.

5.7 Discussions

Strategies using SND were very successful in the 2005 IPD competition. We believe SND is more effective in the IPD than in the ICG or the IBS. The reasons are

- In the IPD, the clarity of behavior are further enforced by the fact that agents anticipate strategies similar to TFT are common in the population. Since defection can often get retaliation, agents often remain cooperative and do not defect.

- However, good strategies in the ICG and the IBS tend to be less sensitive to noise, because they try to defect as often as possible. Therefore, noise correction may not be able to change the behavior of these strategies.

5.8 Summary

The robustness of interactions is an important issue in multi-agent systems, but mistakes due to noise can interfere with the interactions among agents and decrease the performance of the agents. Symbolic noise detection (SND) is a general method for

handling noise efficiently in games like the Noisy IPD. The purpose of this chapter has been to investigate what kinds of features of a game may cause SND to work well or work poorly.

For our investigation, we have evaluated SND in two other games: the Noisy Iterated Chicken Game (ICG) and the Noisy Iterated Battle of the Sexes (IBS), using a simplified version of SND called Naïve Symbolic Noise Filter (NSNF). We found that NSNF was highly accurate in predicting the other player's next moves (96% and 93% of predictions were correct in ICG and IBS, respectively) and did a decent job in correcting actions affected by noise (71% and 59% of corrections were correct in ICG and IBS, respectively).

Moreover, the overall average scores (the average of the scores of all strategies) increase after using NSNF in both ICG and IBS. Also, the increase in ICG is larger than in IBS. Since most strategies in ICG have a higher accuracy of noise correction than strategies in IBS, this seems to support our hypothesis that symbolic noise detection will be more effective in games in which strategies often exhibit deterministic behavior.

However, if we look at each strategy individually, the results are mixed: the accuracy of noise correction does not strongly correlate with the increases in average scores. Some strategies, especially those in IBS, perform worse than before after using NSNF.

To explain these results, we have explored the relationships between (1) the average scores, (2) the increases of average scores due to NSNF, and (3) the accuracy of noise correction of NSNF. We found that strong players and weak players behave quite differently—strong players are more exploitive, having a clear behavior of choosing defect. While this is true for both games, this exploitive behavior has different effect to

the clarity of the other players' behavior in different games. In ICG this causes the other player to exhibit a more clear behavior, but in IBS this does not. We have offered explanations to this phenomenon via the observed characteristics of decision making process of the strategies and the structure of the payoff matrix.

Chapter 6

A Framework for Building Strategies for Repeated Games

In previous sections, we discussed many techniques to synthesize strategies for repeated games in noisy and noise-free environments. To sum up, we present a general framework for constructing strategies from data in repeated games. This framework revolves around three key data structures for opponent modeling and decision making in repeated games: opponent models, action-value functions, and strategies. In this framework, the task of constructing strategies from data can be considered as a process of the construction of data structures from data and then the transformation of one data structure to another. The result of the process is a fully functional strategy that be used to interact with other agents in the environments. Furthermore, if the construction algorithm and the transformation algorithm do not have errors (i.e., it generates data structures that is exactly the corresponding mathematical objects), then the strategies generated by the process are optimal.

6.1 An Overview of the Framework

The diagram in Figure 6.1 is an outline of the framework. According to this framework, data are recorded in previous games and stored in a *data portfolio*, which is basically a collection of objects. Each object in a data portfolio is designated by a *type* which describes what the object is about. For example, the type of an interaction trace is

"interaction trace".

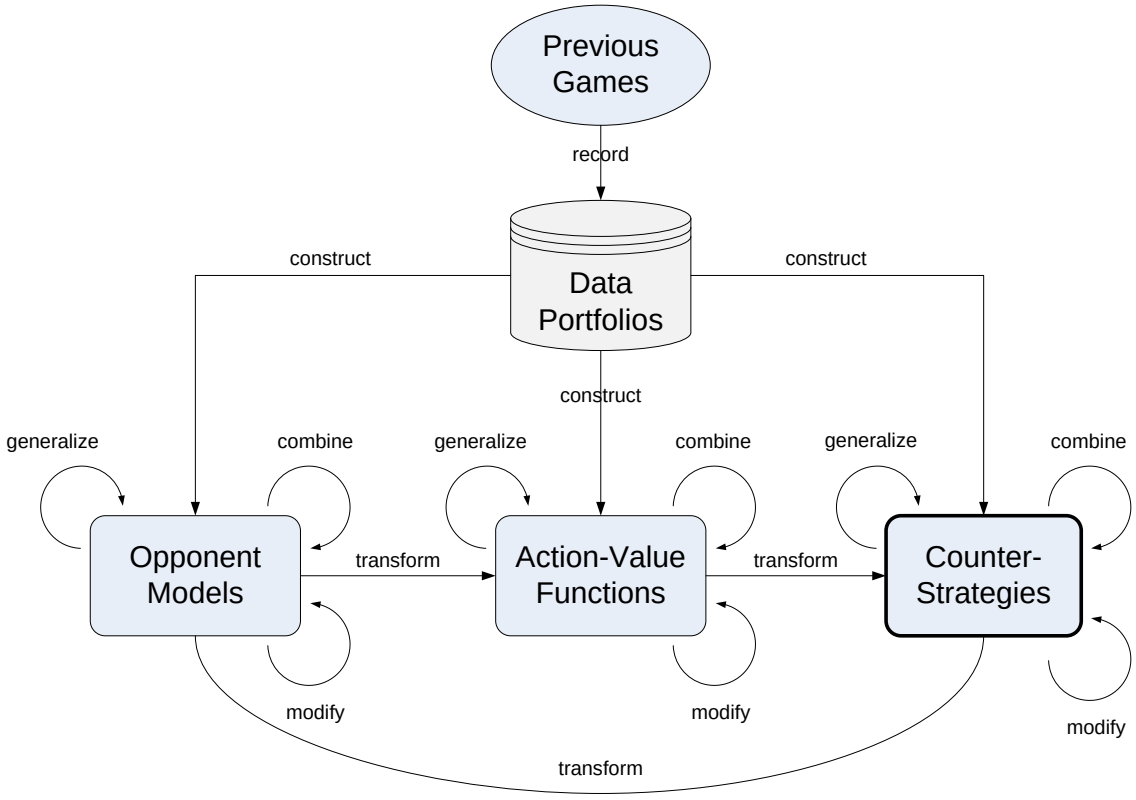

Figure 6.1: A framework for constructing strategies from data.

Construction algorithms, denoted by the "construct" edges in the diagram, use data in a data portfolio to construct a data structure such as an opponent model or a counter-strategy. Each construction algorithm has several *preconditions* that specify what types of data that must present in the data portfolio in order to use the construction algorithm. For example, the CIT algorithm we presented in Section 2 is a construction algorithm that takes a set of interaction traces and some other information and construct a composite strategy. Then the presence of interaction traces in the data portfolio, among other things, must be a precondition of the CIT algorithm.

Transformation algorithms are any algorithms that take some data structures in our

framework and create another data structure. The edges among data structures in the diagram denote transformation algorithms. The preconditions of an transformation algorithm specify what data structures it takes as an input. For example, a transformation algorithm can take an opponent model and compute the corresponding action-value function by dynamic programming such as value iteration. This dynamic programming algorithm is denoted by a "transform" edge from opponent models to action-value functions in the diagram, and its precondition is that the opponent model must be a policy or a finite state automaton.

Some transformation algorithms take two or more data structures and produce a new data structure. If all input data structures are the same kind, we say these transformation algorithms are *combining algorithms*, and are denoted by the "combine" edges in the diagram. One such combining algorithm is about combining several opponent models together by averaging the opponent's moves predicted by the opponent models.

Some transformation algorithms can take several different data structures or a data portfolio as an input. These algorithms will be represented by hyperedges in the diagram.

6.2 Strategy Construction Graphs

Our objective is to construct a counter-strategy such that our agent can use the strategy to interact with the opponent. According to our framework, there are more than one way to achieve this objective. Generally speaking, every path from the root node (i.e., the "Previous Games" node) to the terminal node (i.e., the "Counter-Strategies" node) in the diagram in Figure 6.1 represents a possible way to construct a counter-strategy

from data that are gathered from previous games. Thus, we call such a path a *strategy construction path*. If the diagram has hyperedges, a strategy can be constructed according to a *strategy construction graphs* from the root node to the terminal node. For example, the CIT technique can be represented by the strategy construction graph in Figure 6.2. The CIT technique basically takes a set of interaction traces recorded previous games, and then construct a composite strategy which is a partial strategy. The composite strategy is then "generalized" to a total strategy by augmenting it with a base strategy. This is exactly what the strategy construction path represents in Figure 6.2.

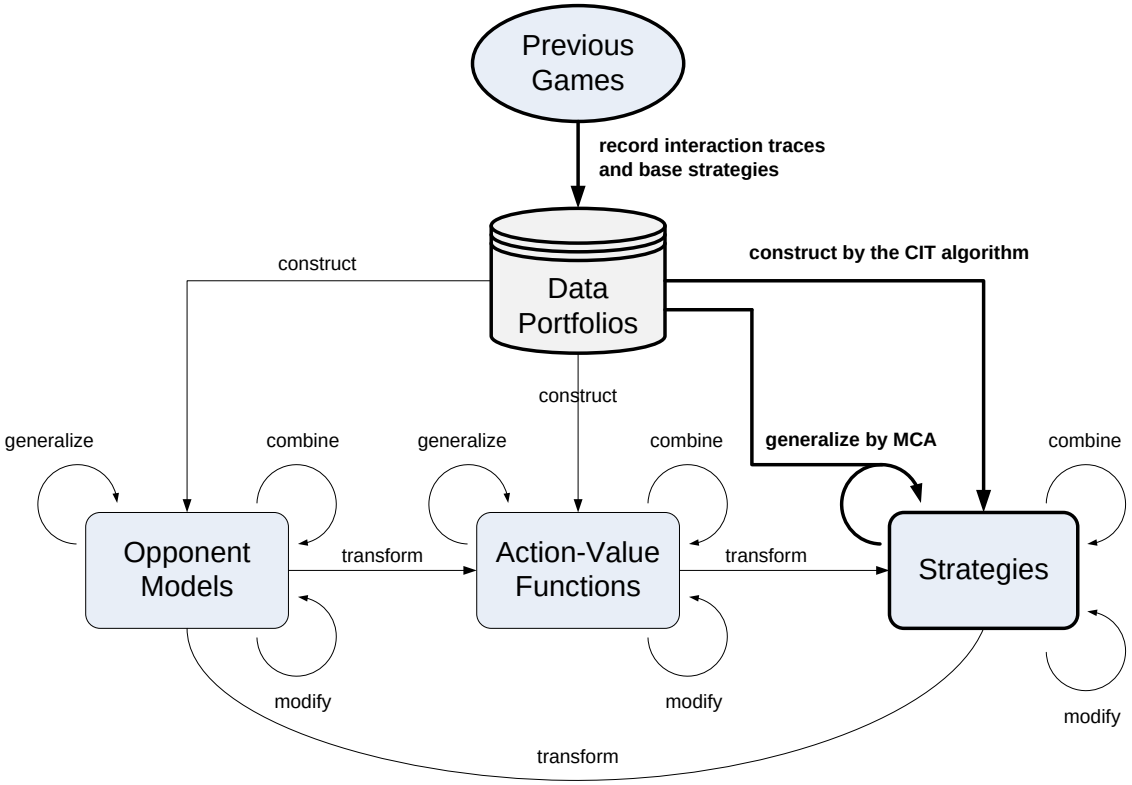

Figure 6.2: The bond nodes and the bond edges represents the strategy construction graph of the CIT technique. Notice that the edge "generalize by MCA" is a hyperedge.

The symbolic noise filter can be considered as a modification to an existing strat-

egy. Figure 6.3 shows how to modify a strategy constructed by the CIT technique with symbolic noise filter.

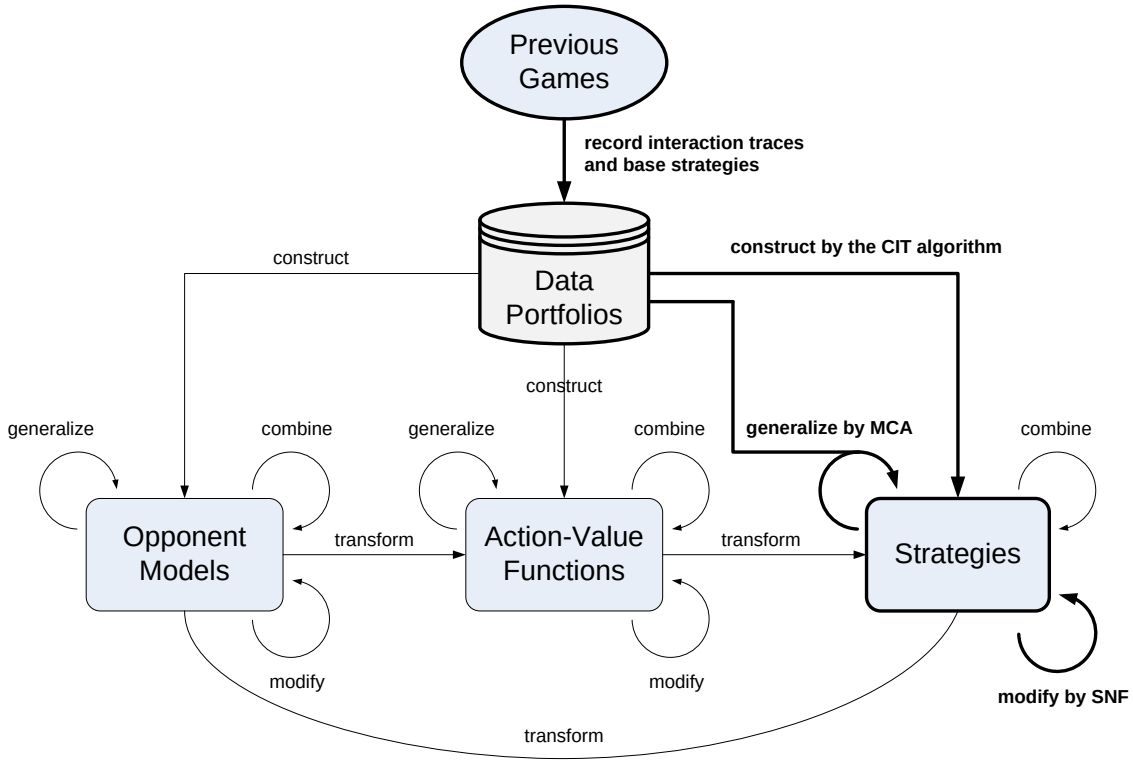

Figure 6.3: The strategy construction graph of the CIT technique with symbolic noise filter.

The strategy construction process can involve a number of data structures such as opponent models and the action-value functions. Each data structure corresponds a mathematical object that has certain desirable properties. More precisely, an opponent model $\hat{\psi}$ corresponds to the true behavior of the opponent (i.e., its mixed strategy ψ); an action-value function $\hat{\nu}$ corresponds to the function ν that returns the *maximum* long-term expected utility of choosing an action at every decision node in a game; a counter-strategy $\hat{\phi}$ corresponds to an optimal strategy ϕ against the opponent.

The idea is that if the construction and transformation algorithms produce data structures that are exactly the same as the corresponding mathematical objects, then the counter-strategy produced by the strategy construction process (according to a strategy construction graph) is always the optimal strategy against the opponent. However, it is difficult to build data structures that are the same as the mathematical objects, due to the fact that data in the data portfolio is incomplete and algorithms are imperfect. Anyway, the goal of the construction and transformation algorithms is to produce a close approximation to the mathematical objects.

Notice that this framework does not tell us which strategy construction graph is the best. We have to conduct cross-validation experiments in order to determine which strategy construction graph is the best.

Chapter 7

Conclusions and Future Work

The central theme of this dissertation is to study how to identify and exploit structures in agents' behavior in non-zero-sum repeated games for effective decision making. In our course of study, we discussed some useful structures in agents' behavior in these games, and proposed several techniques to identify and exploit those structures. This chapter summarizes the accomplishments and proposes new directions for future work.

7.1 Contributions

The five main contributions of this thesis are:

1. **Synthesis of Strategies from Interaction Traces**

   Chapter 2 presented a novel technique for combining interaction traces, which are produced by many different agents in a two-player repeated game, into a composite strategy. The technique is called the CIT technique.

   The idea is stemmed from the observation that given a set $\mathcal{T}$ of interaction traces generated the interaction between an agent λ and a set Λ of agents, it is possible to "reconstruct" λ from $\mathcal{T}$ by synthesizing a new agent λ' whose behavior is exactly the same as λ when interacting with the agents in Λ. It turns out that the same reconstruction procedure can be used to *mix* the interaction traces generated by several different agents together, to form a *composite* strategy whose behavior is

a mixture of the behavior of the agents. Chapter 2 gives (1) the necessary and sufficient conditions under which a set of interaction traces can be combined together to form a composite strategy, and (2) an algorithm that, in polynomial time, generates the best such composite strategy from a database of interaction traces.

One interesting feature of the CIT technique is that it does not rely on the knowledge of the set of opponents' internal states. In open environments such as the IPD tournaments, it is impossible to know the set of all internal states of the opponent. The CIT technique can avoid using the notion of states or belief states by directly using the interaction traces to synthesize a strategy. It shows that it is possible to maximize the expected utility of the composite strategy, with respect to an estimated probability distribution of opponents, without using the notion of policies or automata.

The results in our experiments show that given a collection of agents our technique can produce a strategy that does better than all of the given agents, except the best agents, by combining the "best" behaviors (in terms of interaction traces) exhibited by the given agents. The use of composite strategies significantly improved the agent's rank (compared to the other agents) by 12% in the IPD, 38% in the ICG, and 33% in the IBS.

2. **Solvability of Task-Completion Problems with Goal Uncertainty**

Chapter 3 focused on task-completion problems in which a goal-based agent must interact with a nondeterministic environment in order to achieve a goal, but the agent is uncertain about the goals. These problems, however, are not always solv-

able; in other words, some task-completion problems (with or without goal uncertainty) have no strategy that can guarantee an agent to achieve the goals. To distinguish unsolvable problems from those that are solvable, we introduced the notion of strong solvability and weak solvability, and stated the necessary and sufficient conditions for a problem to be strongly solvable (i.e., there exists agents that can always succeed).

The CIT algorithm in Chapter 2 can only be used to find a solution for strongly solvable problems. But the CIT algorithm is not applicable to weakly solvable problems, since it cannot find a compatible set of interaction traces for every possible configuration of the environment. Therefore we devised the CIT-search algorithm, which takes a database of successful interaction traces for each configuration of the nondeterministic environments, and then produces a composite agent function that has the highest probability of success among all combinations of the interaction traces.

The CIT-search algorithm can produce an optimal solution for weakly solvable problems if the interaction trace database is large enough. But even if there are only few interaction traces in the database, our experiments showed that composite agent functions produced by CIT-search can still perform well—composite agent functions, when incorporated in an existing problem solving strategy, can improve the success rate of the strategy.

3. **Symbolic Noise Detection**

Chapter 4 introduced an approach for coping with noise in 2×2 repeated games,

and evaluated the approach in the Noisy Iterated Prisoner's Dilemma. The approach, called symbolic noise defection (SND), is radically different from previous approaches which are mostly based on the principle of forgiveness. SND is more effective than previous approaches because SND is more accurate in identifying situations in which forgiveness (i.e., temporary tolerance) are needed.

To test the performance of our noise detection approach, we wrote several programs, called DBS, to participate in the 2005 Iterated Prisoner's Dilemma competition. The idea behind DBS is that in order to distinguish between intentional and unintentional misbehaviors, DBS uses a combination of symbolic noise detection plus temporary tolerance: if an action of the other player is inconsistent with the player's past behavior, we continue as if the player's behavior has not changed, until we gather sufficient evidence to see whether the inconsistency was caused by noise or by a genuine change in the other player's behavior.

The performance of DBS in the competition was quite impressive: seven of nine DBS strategies were placed top-ten, and were only lost to strategies that have slave programs to feed points to them. According to our analysis of the data provided by the organizer of the tournament, our best DBS strategy would have placed first in the tournament if the size of each master-and-slaves team had been limited to less than 10.

The effectiveness of SND is due to a nice property of agent's behavior in the Iterated Prisoner's Dilemma: there are clear and simple patterns in the moves made by IPD agents, and therefore DBS can easily construct deterministic models of the other

player's behavior, and then use the models to fend off noise. This phenomenon is due to the fact that clarity of behavior is an important ingredient of long-term cooperation in the IPD, and therefore agents are encouraged to play deterministically most of the time.

It is well-known that accidents can cause great difficulty in cooperation with others. To our knowledge, there are few general techniques that can effectively handle errors in interaction among agents (triggered by noise or malicious third parties) in multi-agent environments. The performance of SND shows the potentials of SND as a general technique for coping with noise.

4. **Analysis of Symbolic Noise Filter**

Chapter 5 experimentally evaluated the performance of SND in games other than the IPD. The approach is to take out the SND mechanism in DBS and formulate it as a wrapper that can be placed around any existing strategy in 2×2 repeated games, in order to endow the strategy with the capability of noise detection and correction. The wrapper is called symbolic noise filter (SNF) and it performs two functions: identify actions that has been affected by noise in the inputs of the underlying agent (noise detection), and then correct those actions before feeding the actions to the underlying agent (noise correction).

The evaluation of SNF was conducted in two repeated games: the Iterated Chicken game and the Iterated Battle of the Sexes. The results is that SNF was highly effective in these games. More precisely, NSNF was highly accurate in predicting the other player's next moves (96% and 93% of predictions were correct in ICG

and IBS, respectively) and did a decent job in correcting actions affected by noise (71% and 59% of corrections were correct in ICG and IBS, respectively).

Further experimental analysis focused on the relationships of (1) the average scores of the programs, (2) the increases in average scores due to the use of SNF, and (3) the accuracy of correction of SNF. The relationships can be explained by measuring the frequency of change of decisions and the frequency of defects made by the programs. The analysis showed that strong players and weak players behave differently—strong players are more exploitive, having a clear behavior of choosing defect. In conclusion, the effectiveness of SNF depends on (1) whether the behavior of the other programs is predictable, and (2) how the underlying agent responds to the defections made by the other players. We discuss how the payoff matrix affects the decision making process of the agents, which in turn affects the effectiveness of noise detection and correction.

5. **A Framework for Experience-Based Strategy Construction for Repeated Games**

Chapter 6 proposes a general framework for construction of strategies from data for repeated games. The framework models the learning process that involves the construction of several key data structures from data, the transformation of one data structure to another, and eventually the construction of strategies from the data structures. The data structures in this framework are opponent models, action-value functions, and strategies. We discussed how the CIT technique and the noise detection technique fit in this framework.

Apart from the above technical contributions, this dissertation also includes two scientific discoveries pertaining to the behavior of agents in the IPD, ICG, and IBS. The good performance of our techniques is due to the effective exploitation of these empirical properties pertaining to the behavior of the agents. We predict that these empirical properties of agents' behavior are also valid in other non-zero-sum repeated games as well.

1. **Deterministic behavior is abundant.**

   The abundance of deterministic behavior in the IPD is not a news: Axelrod has already said that clarity of behavior is an important ingredient of long-term cooperation [3]. But what is new here is that deterministic behavior can also be abundant in other non-zero-sum repeated games as well (e.g., the ICG and the IBS), and the exhibition of agents' deterministic behavior are not necessarily motivated by the incentives for cooperation; the deterministic behavior can be due to other reasons such as the incentives for making threats, as demonstrated by top ICG and IBS agents who often dominate other players by unambiguously choosing defections. This is why noise are often detectable by symbolic noise detection in these games.

2. **Distributions of interaction traces are highly skewed.**

   It is unsurprising to observe certain recurrent structures in agents' behavior in repeated games. But what's surprising is that in the repeated games we considered the interaction traces generated by agents playing against each other often belongs to a very small group of interaction traces. In short, the distributions of interaction traces in those games are highly skewed. That is why composite strategies con-

taining only about 200 interaction traces can interact successfully with the agents in those games without resorting to the base agents most of the time, despite that there are $4^{200} = 2.58 \times 10^{120}$ possible ways of interaction in a game that lasts 200 iteration. In general, we believe distributions of interaction traces in non-zero-sum repeated games are more skewed than zero-sum repeated games.

A counter-argument against the above statements is that it is possible for one to create agents that do not have these properties. For example, one can submit a strategy which returns moves randomly in the IPD tournament. Clearly this strategy does not exhibit clarity of behavior and the interaction traces produced by it are very diverse. The problem with this argument is that many agents would not use such strategies since they do not perform well in the IPD, ICG, or IBS, and hence, such strategies are unlikely to occur in these games.

Although in most circumstances the above empirical properties can be easily observed in the IPD, ICG, and IBS, these properties can disappear under certain conditions. For example, in one particular experiment we told students who participated in our tournaments that the best agent for the Noisy IPD is one that exploits the clarity of behavior of other agents. Also, we *encouraged* students to study the best agents and find ways to beat them. The result was that many agents submitted by the students have little or no clarity of behavior. But in the 2005 IPD tournament and several other IPD tournaments we held very few agents behave ambiguously. The results of the experiments indicates that whether agents would exhibit clarity of behavior or diversified behaviors depends on the set-up of the tournament and the conditions under which the tournaments is run. But

despite of the exceptional cases, the empirical properties we mentioned here are generally true.

My conviction is that there are laws governing the behavior of agents in non-zero-sum repeated games. But these laws, like physical laws in the nature, are not absolute true but close approximations to the reality whose truth is always out of reach. Hence, no matter how I formulate the laws about the agents' behavior there are exceptional cases in which these laws fail. But progress can be made by refining existing laws so as to take those exceptional cases into account. For example, we currently don't know any non-zero-sum repeated game whose distribution of interaction traces is not skewed but uniform. But if one of these games is discovered in the future, our statement about the distribution of interaction traces should be refined.

But from an engineering standpoint simple laws that are close approximations to the reality is often good enough. That is why mechanical engineers still prefer Newtonian mechanics to Einstein's mechanics, even though Einstein's mechanics is a more accurate description of force in the nature. Hence, SND might still be useful in many non-zero-sum repeated games, despite that there may be better theories about agents' behavior in those games.

7.2 Directions for Future Work

At the end of each chapter, there is a discussion about extensions and future work of the techniques in that chapter. Here we summarize the main points of these discussions and provide a broader view about the synthesis of strategies in repeated games. The future

work can be categorized into the following subcategories:

1. Adding new functionality to the existing techniques

2. Adapting the existing techniques to other problems or applications; and

3. Using the general framework for the synthesis of strategies from data in repeated games.

7.2.1 New Functionality

The CIT algorithm requires an estimate of the probability with which we'll encounter each of opponents we have observed. In fact, this requirement may be unnecessary because the database of interaction traces contains lots of information about the distribution of opponents' strategy. In the future, we would like to study how to estimate the probability from a given set of interaction traces and modify the CIT techniques to alleviate this requirement.

Unlike reinforcement learning and POMDPs, our formalization for the CIT technique cannot handle problems whose horizon is infinite. The reason for this shortcoming is that we deliberately use interaction traces in place of states, and all interaction traces in the composite strategies are finite. But without the notion of states it is hard to extend the solutions of the CIT technique to problems with infinite horizon. In the future, we would like to see how to extend the CIT algorithm to deal with interaction trace of potentially infinite length (e.g., interaction traces with loops).

Currently, the CIT technique relies on a base strategy to deal with situations in which composite strategies fail due to the lack of interaction traces. But in more com-

plicated problems we usually do not have enough interaction traces to deal with all contingencies in the problems. It is essentially true for the CIT search algorithm, which is designed for task-completion problems in which agents has to complete a task despite goal uncertainty, because these problems are more complicated than repeated games. Thus in the future, we want to develop a better technique to deal with the issues of the lack of interaction traces. One direction is to use some sorts of generalizations schemes similar to neural networks as an generic nonlinear function approximator for state-action mappings in TD-Gammon [62].

The CIT-search algorithm in Chapter 3 is an algorithm for finding composite agent functions for weakly solvable task completion problems. We cannot use the CIT algorithm in Chapter 2 for these problems because the CIT algorithm can only be used for solving strongly solvable problems. But unlike the CIT algorithm, CIT-search is a brute-force search algorithm and thus is quite inefficient when the size of interaction trace database is large. In the future, we would like to develop a more efficient algorithm for weakly solvable problems.

The major criticism of the symbolic noise detection technique is that if opponents defect occasionally, SND cannot distinguish these intentional defections from defections triggered by noise. Thus, the policy of temporary tolerance in DBS may be exploited by a "hypocrite" strategy that behaves like TFT most of the time but occasionally defects even though DBS did not defect in the previous iteration. But fortunately according to our experience very few agents use hypocrite strategies. Perhaps the scarcity of hypocrite strategies is due to the fact that the benefit of occasional defections may not outweigh the risk of triggering mutual defections: occasional defections can earn very little points,

but the hypocrite strategy could lose many points if there are strategies in the tournament that cannot tolerate occasional defections. However, we do believe that as long as the frequency of occasional defections does not exceed certain levels hypocrite strategies can still benefit from occasional defection without causing mutual defections. We are interested to determine the level of occasional defections the agents in the IPD tournaments can tolerate, and modify DBS to deal with intentional defections.

Another criticism of the SND technique is that in reality there is no way to determine the correct noise level. In most situations, an agent doesn't even know whether noise exists or not. The use of SND when there is noise can actually decrease the performance of an agent. One way to deal with this problem is to include a mechanism in SND for the estimation of the noise level in the environment, and then use this estimation to adjust the parameters in SND or turn off SND when there is no noise. Then such "adaptive" SND can be used in both noisy and noise-free environments. The questions, of course, are (1) how to estimate the level of noise in the environment, and (2) how to tune the parameters in symbolic noise detection for different noise level. These are the issues we need to address in the development of adaptive SND.

In our studies of noise in repeated games, the assumption is that agents do not know whether an action has been affected by noise. While this assumption holds in many domains, there are situations in which some agents can acquire partial information about the occurrence of noise. For example, agents may obtain a plan of a malicious third party by counter-espionage. In the future, we are interested in study how SND can utilize these information to improve the accuracy of noise detection.

7.2.2 New Problems and Applications

The CIT technique, in its current form, is not suitable for large-scale zero-sum games such as chess, since games among experts in zero-sum games usually lead to situations that no player has ever seen before. Thus, the CIT technique is not effective in zero-sum games since the number of interaction traces is usually too small to provide large increases in an agent's performance. We would like to develop a way to combine the CIT technique with game-tree search, to see how to use the knowledge in the interaction traces for game-tree search in zero-sum games.

The fact that composite strategies in our experiments do not fail in about half of the games even if some agents in the test sets are probabilistic is a surprising result. In spite of the contingencies due to the use of random number generators, the opponents' behavior tends to vary very little. This is why the number of interaction traces in the composite strategies is large enough to cover most of the contingencies in the probabilistic agents. But in other domains agents may exhibit so many different behavior that a small composite strategy is not sufficient to deal with all possible agent's behavior. But fortunately in some nondeterministic environments contingencies are more or less predictable and can be properly modeled (unlike zero-sum games in which the opponents' moves are less predictable). In the future, we would like to extend the CIT technique to nondeterministic environments in which there are lots of contingencies.

The CIT technique can be considered as a case-based reasoning technique in which a case is an interaction trace. But problems can occur when an interaction trace is "adapted" to other problems, since interaction traces recorded for one game may not be

directly used in other games. Therefore, some sort of case adaption of interaction traces is needed when using the CIT technique for knowledge transfer or case-based reasoning. In the future, we would like to see how to adapt interaction traces recorded in one situation to other situations.

The success of the noise detection technique for the IPD depends on a number of factors that are unique to the 2×2 repeated games. For example, it relies on the fact that there are only two possible actions such that it can correct an noise-affected action by flipping it to the other action. But the action space in many real-world applications is usually very large, and the task of noise correction is not simple. Hence, it is necessary to develop techniques for error correction in conjunction with noise detection.

An interesting question is whether the DBS strategy is evolutionarily stable. Apparently it is not, because once every agent in the population uses DBS, the hypocrite strategies, which acts like TFT most of the time but intentionally choose defections in a small number of iterations, can outperform DBS. However, it is questionable that hypocrite strategies can dominate the population, since the performance of hypocrite strategies when playing with each other is poorer than the performance of DBS. Thus, it would be interesting to see what kind of strategies is evolutionary stable in the noisy IPD. We believe a small modification to DBS can make it more stable in evolutionary environments.

In multi-agent environments where agents can communicate with each other, agents might be able to detect noise and avoid conflicts by using a predefined communication protocol. Hence, communications can be an alternative solutions to cope with noise in noisy environments. However, we believe that as long as the agents cannot completely trust each other there is no protocol that is guaranteed to tell which action has been af-

fected by noise. It would be interesting to compare SND with these alternative approaches for coping noise, and study how to combine these approaches.

7.2.3 Synthesis of Strategies for Repeated Games

Chapter 6 described a framework for the synthesis of strategies for repeated games. In this framework, we introduced the concept of strategy construction paths, which shows the way how data gathered from previous games or tournaments can be turned into a strategy via the approximations and transformations of data structures such as opponent models, action-value functions, and strategies. In the future, we would like to conduct experiments to see which construction path is the best for different repeated games.

7.3 Summary

This dissertation contributes techniques for coping with noise in non-zero-sum games and combining interaction traces for creating new and better agents. These techniques demonstrate some possible ways to identify and exploit useful structures in agents' behavior for effective decision making in non-zero-sum repeated games.

The issue of noise and the ignorance about the behavior of other agents in environments are major problems in many other problems as well. We hope that the success of our techniques in simple games like repeated games can shed light on the same issues in other domains such as the creation of robots for interacting people or other robots in the real world.

Appendix A

List of Acronyms

The acronyms that appear in this thesis:

CA: Composite Agent

CIT: Combining Interaction Traces

IBS: Iterated Battle of the Sexes

ICG: Iterated Chicken Game

IPD: Iterated Prisoner's Dilemma

MCA: Modified Composite Agent

MDP: Markov Decision Process

The acronyms of the strategies for the Iterated Prisoner's Dilemma in the literature.

ALLC: Always Cooperate

ALLD: Always Defect

BWIN: The master program submitted by the University of Southampton in the 2005 IPD tournament.

DBS: Derived Belief Strategy

DBSz: Derived Belief Strategy (version z)

GRIM: Grim trigger / the grim strategy

IMM01: The master program submitted by Jia-wei Li

LSF: Learning of Opponent's Strategy with Forgiveness

NEG: Negation strategy

PAVLOV: Pavlov strategy

STFT: Suscpicious TFT

TFT: Tit-for-Tat

TFTI: Tit-for-Tat Improved

TFTT: Tit-for-Two-Tat

WSLS: Win-Stay, Lose-Shift

Appendix B

Glossary of Notation

Basic Mathematics

$\mathbb{R}$	The set of real numbers
$\mathbb{R}^{\geq 0}$	The set of non-negative real numbers
$\mathbb{R}^{+}$	The set of positive real numbers
$\mathsf{dom}(f)$	The domain of a function f
$\mathsf{range}(f)$	The range of a function f
$a := b$	Assign b to a
P or Δ or δ	A probability distribution
$P(a)$ or $\Delta(a)$ or $\delta(a)$	The probability of the event a under P or Δ
P_S or Δ_S	A probability distribution over the set S
$\mathbb{P}_S$	The set of all probability distributions over the set S

Sequences

$\alpha = \langle a_1, a_2, \ldots, a_n \rangle$	A sequence
$\langle\rangle$	An empty sequence
$[\alpha]_k$	The k'th element in the sequence α
$\mathsf{prefix}_k(\alpha)$	The k'th prefix of the sequence α (including the k'th element)
$\mathsf{suffix}_k(\alpha)$	The k'th suffix of the sequence α (excluding the k'th element)
$\tau_1 \circ \tau_2$	The concatenation of τ_1 and τ_2
$\mathsf{lcp}(\alpha_1, \alpha_2)$	The longest common prefix of two sequences α_1 and α_2
Nil	A "nil" element

Agents

A	The set of actions (for our agent)
B	The set of opponent's actions or the set of percepts
λ	An agent
λ_A	Our agent (for which we have to construct a strategy)
λ_B	The opponent
Λ	A set of agents.

Strategies

ϕ	A pure strategy.
ϕ_A	The pure strategy of our agent.
ϕ_B	The pure strategy of the opponent.
Φ	A set of pure strategies.
Φ_A	The set of all possible pure strategies for our agent.
Φ_B	The set of all possible pure strategies for the opponent.
ψ	A mixed strategy.
$\mathsf{strategy}(\lambda)$	The (pure or mixed) strategy of the agent λ

Interaction Traces / Histories

N	The length of a repeated game (i.e., the number of normal-form games in a repeated game)
τ	An interaction trace or a history
τ^A or $\mathsf{actions}(\tau)$	The sequence of our agent's actions in τ
τ^B or $\mathsf{percepts}(\tau)$	The sequence of the opponent's actions in τ
$\lvert\tau\rvert$	The length of τ
$\mathsf{prefix}_k(\tau)$	The k'th prefix of τ
$\mathsf{suffix}_k(\tau)$	The k'th suffix of τ
$\mathcal{T}$	A set of interaction traces
$\mathcal{H}$	The set of all possible histories in a repeated games
$\mathsf{s}(\tau)$	The situation led by a history from the initial state

Interaction

$\mathsf{trace}(\phi_1, \phi_2)$	The interaction trace generated by two deterministic agents using pure strategies ϕ_1 and ϕ_2
$\mathcal{H}(\lambda, \cdot)$	The set of all possible histories generated by the agent λ as our agent
$\mathcal{H}(\cdot, \lambda)$	The set of all possible histories generated by the agent λ as the opponent
$\mathcal{H}(\phi, \cdot)$	The set of all possible histories generated by the pure strategy ϕ as our agent
$\mathcal{H}(\cdot, \phi)$	The set of all possible histories generated by the pure strategy ϕ as the opponent

Utilities

u	A utility function
$u_A(\tau)$	The utility for our agent when the history is τ
$u_B(\tau)$	The utility for the opponent when the history is τ
EU	Expected Utility
$EU_A(\lambda_A, \lambda_B)$	The expected utility of our agent λ_A when interacts with the opponent λ_B
$EU_B(\lambda_A, \lambda_B)$	The expected utility of the opponent λ_B when interacts with our agent λ_A

Policies

Cond	A condition (a mapping from histories to Boolean values)
$\mathsf{Cond} \rightarrow p$	A rule in a policy
r	A rule
R	A set of rules
$\mathcal{C}$	A policy schema
π	A policy
Π	A set of policies

Environments, Percepts, and Task-Completion Problems

e	A deterministic environment
$\bar{\mathsf{e}}$	A nondeterministic environment
$\hat{\mathsf{e}}$	A probabilistic environment
ϕ	A deterministic agent function (i.e., a pure strategy)
$\bar{\phi}$	A nondeterministic agent function
$\hat{\phi}$	A probabilistic agent function (i.e., a mixed strategy)
$\mathbb{E}$	A set of deterministic environments
$\mathbb{E}^{\bar{\mathsf{e}}}$	The set of configurations (i.e., deterministic environments) that is equivalent to $\bar{\mathsf{e}}$
$(\mathbb{E}^{\hat{\mathsf{e}}}, \Delta^{\hat{\mathsf{e}}})$	The set of configurations, together with the probability distribution over $\mathbb{E}^{\hat{\mathsf{e}}}$, that is equivalent to $\hat{\mathsf{e}}$

Environments, Percepts, and Task-Completion Problems

$\mathcal{P}$	A deterministic task-completion problem (without goal uncertainty)
$\bar{\mathcal{P}}$	A nondeterministic task-completion problem (with or without goal uncertainty)
$\hat{\mathcal{P}}$	A probabilistic task-completion problem (with or without goal uncertainty)
$\alpha = \langle a_1, a_2, \ldots, a_n \rangle$	A sequence of actions
$\beta = \langle b_1, b_2, \ldots, b_n \rangle$	A sequence of percepts
$\mathsf{actions}(\mathcal{T})$	The set of all action sequences of the interaction traces in the set $\mathcal{T}$
$\mathsf{percepts}(\mathcal{T})$	The set of all percept sequences of the interaction traces in the set $\mathcal{T}$
$\mathsf{trace}(\phi, \mathsf{e})$	The interaction trace generated by the deterministic agent function ϕ interacting with the deterministic environment e
$\mathsf{trace}(\alpha, \mathsf{e})$	The interaction trace generated by feeding the action sequence α into the deterministic environment e
$\mathsf{trace}(\phi, \beta)$	The interaction trace generated by feeding the percept sequence β into the deterministic agent function ϕ

Environments, Percepts, and Task-Completion Problems

$\mathsf{traces}(\phi, \bar{\mathsf{e}})$	the set of all possible interaction traces when ϕ interacts with $\bar{\mathsf{e}}$
$\mathsf{traces}(\phi, \hat{\mathsf{e}})$	the set of all possible interaction traces when ϕ interacts with $\hat{\mathsf{e}}$
ξ	An environment-interaction mapping (EI-mapping)

Opponent Models and Action-Value Functions

$\hat{\phi}$	An opponent model of a pure strategy.
$\hat{\psi}$	An opponent model of a mixed strategy.
ν	An action-value function
$\hat{\nu}$	An action-value model

Bibliography

[1] T.-C. Au and D. Nau. Accident or intention: That is the question (in the noisy iterated prisoner's dilemma). In *Proceedings of the International Joint Conferenceon Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 561–568, 2006.

[2] T.-C. Au and D. Nau. Is it accidental or intentional? a symbolic approach to the noisy iterated prisoner's dilemma. In *The Iterated Prisoners' Dilemma: 20 Years on*, pages 231–262. World Scientific, 2007.

[3] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[4] R. Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, 1997.

[5] R. Axelrod and D. Dion. The further evolution of cooperation. *Science*, 242(4884):1385–1390, 1988.

[6] B. Banerjee and P. Stone. General game learning using knowledge transfer. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007.

[7] L. Barone and L. While. Evolving adaptive play for simplified poker. In *Proceedings of IEE International Conference on Computational Intelligence (ICEC-98)*, pages 108–113, 1998.

[8] L. Barone and L. While. An adaptive learning model for simplified poker using evolutionary algorithms. In *Proceedings of the Congreess of Evolutionary Computation (GECCO-1999)*, pages 153–160, 1999.

[9] L. Barone and L. While. Adaptive learning for poker. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 566–573, 2000.

[10] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1–2):81–138, 1995.

[11] J. Bendor. In good times and bad: Reciprocity in an uncertain world. *American Journal of Politicial Science*, 31(3):531–558, 1987.

[12] J. Bendor, R. M. Kramer, and S. Stout. When in doubt... cooperation in a noisy prisoner's dilemma. *The Journal of Conflict Resolution*, 35(4):691–719, 1991.

[13] S. Bhansali and M. T. Harandi. Synthesis of UNIX programs using derivational analogy. *Machine Learning*, 10:7–55, 1993.

[14] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, 2003.

[15] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 493–499, 1998.

[16] R. Boyd. Mistakes allow evolutionary stability in the repeated prisoner's dilemma game. *Journal of Theoretical Biology*, 136:47–56, 1989.

[17] B. Carlsson and K. I. Jönsson. Differences between the iterated prisoner's dilemma and the chicken game under noisy conditions. In *Proc. of the 2002 ACM symposium on Applied computing*, pages 42–48, 2002.

[18] D. Carmel and S. Markovitch. The M* algorithms: Incorporating opponent models into adversary search. Technical Report CIS9402, Computer Science Department Technion, 1994.

[19] D. Carmel and S. Markovitch. How to explore your opponent's strategy (almost) optimally. In *Proceedings of the Third International Conference on Multi-Agent Systems*, pages 64–71, 1998.

[20] D. Carmel and S. Markovitch. Exploration strategies for model-based learning in multiagent systems. *Autonomous Agents and Multi-agent Systems*, 2:147–172, 1999.

[21] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'2000)*, pages 1467–1473, 2000.

[22] J. Denzinger and J. Hamdan. Improving modeling of other agents using tentative stereotypes and compactification of observations. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 106–112, 2004.

[23] M. Deutsch. *The Resolution of Conflict: Constructive and Destructive Processes*. Yale University Press, 1973.

[24] C. Donninger. *Paradoxical Effects of Social Behavior*, chapter Is it always efficient to be nice?, pages 123–134. Heidelberg: Physica Verlag, 1986.

[25] C. Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research (JAIR)*, 16:59–104, 2002.

[26] D. W. Dyer. Opponent modelling and strategy evolution in the iterated prisoner's dilemma. Master's thesis, School of Computer Science and Software Engineering, The University of Western Australia, 2004.

[27] D. Egnor. Iocaine powder explained. *ICGA Journal*, 23(1):33–35, 2000.

[28] E. Even-Dar, S. M. Kakade, and Y. Mansour. Reinforcement learning in POMDPs without resets. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[29] M. J. Fischer and N. A. Lynch. Impossibility of distributed consensus with one faulty. *Journal of the ACM*, 32(2):374–382, 1985.

[30] J. C. Gittins. *Multi-Armed Bandit Allocation Indices*. John Wiley & Sons Inc, 1989.

[31] P. Hingston and G. Kendall. Learning versus evolution in iterated prisoner's dilemma. In *Proceedings of the Congress on Evolutionary Computation (CEC'04)*, 2004.

[32] R. Hoffmann. Twenty years on: The evolution of cooperation revisited. *Journal of Artificial Societies and Social Simulation*, 3(2), 2000.

[33] G. Kendall, X. Yao, and S. Y. Chong. *The Iterated Prisoner's Dilemma: 20 Years On*. World Scientific, 2007.

[34] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[35] D. Kraines and V. Kraines. Pavlov and the prisoner's dilemma. *Theory and Decision*, 26:47–79, 1989.

[36] D. Kraines and V. Kraines. Learning to cooperate with pavlov an adaptive strategy for the iterated prisoner's dilemma with noise. *Theory and Decision*, 35:107–150, 1993.

[37] D. Kraines and V. Kraines. Evolution of learning among pavlov strategies in a competitive environment with noise. *The Journal of Conflict Resolution*, 39(3):439–466, 1995.

[38] S. T. Kuhn. Prisoner's dilemma. `http://karmak.org/archive/2002/11/Prisoner'sDilemma.html` Stanford Encyclopedia of Philosophy, 2001.

[39] D. B. Leake. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, 1996.

[40] J. Li. How to design a strategy to win an IPD tournament. In G. Kendall, X. Yao, and S. Y. Chong, editors, *The Iterated Prisoner's Dilemma: 20 Years On*, pages 89–104. World Scientific, 2007.

[41] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 362–370, 1995.

[42] R. D. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Wiley, 1957.

[43] A. McGovern and R. S. Sutton. Macro-actions in reinforcement learning: An empirical analysis. Technical Report 98-70, University of Massachusetts, Amherst, 1998.

[44] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[45] P. Molander. The optimal level of generosity in a selfish, uncertain environment. *The Journal of Conflict Resolution*, 29(4):611–618, 1985.

[46] U. Mueller. Optimal retaliation for optimal cooperation. *The Journal of Conflict Resolution*, 31(4):692–724, 1987.

[47] M. Nowak and K. Sigmund. The evolution of stochastic strategies in the prisoner's dilemma. *Acta Applicandae Mathematicae*, 20:247–265, 1990.

[48] M. Nowak and K. Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364:56–58, 1993.

[49] M. A. Nowak and K. Sigmund. Tit for tat in heterogeneous populations. *Nature*, 355:250–253, 1992.

[50] C. O'Riordan. Iterated prisoner's dilemma: A review. Technical Report NUIG-IT-260601, Department of Information Technology, National University of Ireland, Galway, 2001.

[51] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[52] A. Parker, D. Nau, and V. Subrahmanian. Overconfidence or paranoia? search in imperfect-information games. submitted to AAAI, 2006.

[53] R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[54] D. Precup, R. S. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *ECML*, pages 382–393, 1998.

[55] A. Rapoport and A. M. Chammah. *Prisoner's dilemma*. University of Michigan Press, 1965.

[56] A. Rubinstein. *Modeling Bounded Rationality*. The MIT Press, 1998.

[57] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[58] H. A. Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(99–118):1, 1955.

[59] H. A. Simon. *Decision Making and Problem Solving*. National Academy Press, 1986.

[60] J. M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.

[61] R. Sugden. *The economics of rights, co-operation and welfare*. Blackwell, 1986.

[62] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3), 1995.

[63] M. M. Veloso and J. G. Carbonell. Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278, 1993.

[64] J. Wu and R. Axelrod. How to cope with noise in the iterated prisoner's dilemma. *Journal of Conflict Resolution*, 39:183–189, 1995.

[65] D. Zeng and K. Sycara. Using case-based reasoning as a reinforcement learning framework for optimization with changing criteria. In *ICTAI*, 1995.