Utilising Policy Types to Achieve Effective Ad Hoc Coordination in the Game Othello

Edward Stevinson

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

Abstract

Ad hoc coordination problems in multiagent systems involve heterogeneous agents that do not know, a priori, how other agents in the system behave. Traditional attempts to create autonomous agents in such settings have been limited in their flexibility, being designed in the context of homogeneous agents, or requiring lengthy learning periods based on past observations, limiting their efficiency. Albrecht (2015) derives a game model called *Harsanyi-Bellman Ad Hoc Coordination* (HBA) which assumes other agents draw their latent type from some unknown distribution over a hypothesised set of types, each of which specifies a complete behaviour. Based on the current interaction history, HBA forms beliefs about the likelihood of player types which are subsequently used to compute optimal action responses. This allows the HBA agent to form dynamic models of its opponents and hypothesise a wide range of behaviours, and by committing to fixed predefined types, only requires a fraction of the observations used in other forms of learning.

HBA has been tested in simple tasks such as Rock-Paper-Scissors and Prisoner's Dilemma with successful results. The next step in its validation, and the focus of the project, is its ability to scale successfully to more complicated games. The board game Othello was chosen as a suitable candidate task, with its larger game tree and richer strategies. From analysis of the strategies of Othello the hypothesised type space for a number of HBA agents was created. These agents (along with baseline agents) played in a number of semi and full ad hoc coordination simulations of Othello, as well as against a number of human players. The win-rates of these games was used as the primary indicator of performance. A series of further tests were also carried out to study the importance of other HBA variables such as the planning horizon.

The HBA agents performed significantly better than the baseline agents in the semi ad hoc simulations, achieving win-rates of up to 82% against agents with mixed, dynamic type distributions. The reduced performance in the full ad hoc coordination simulations, in which the true type spaces are not necessarily subsets of the hypothesised types spaces, highlighted the issue of how beliefs are updated when hypothesised types are incorrect. In addition, the results also highlighted the critical importance of task specific planning horizons. Whilst these issues will need to be addressed, the

results of this thesis provide support for the potential of HBA to be used in increasingly varied and complex applications.

Acknowledgements

I would like to thank my academic supervisor, Alex Lascarides, for the guidance provided throughout the project. Thanks is also given to Stefano Albrecht for providing advice and example code from his original HBA experiments.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree of professional qualification except as specified.

(Edward Stevinson)

Contents

List of Figures

Chapter 1

Introduction

1.1 Nature of Ad Hoc Coordination

When autonomous agents are utilised in real world tasks, whether in virtual or robotic settings, they must successfully interact with unfamiliar agents with whom strategies cannot be developed a priori. This is the framework of a*d hoc coordination* problems (Albrecht, 2015a; Stone, 2010). Success is dependent on the level of flexibility and efficiency they achieve in completing their tasks, i.e. achieving their goals in the presence of heterogeneous agents and with a good relation between their payoff and the time taken to complete goals.

Such problems possess characteristics that make traditional methods of multiagent interaction unsuitable. Facing agents with heterogeneous behaviours, agents in such systems need to overcome ignorance over what possible strategies an opponent might hold without the benefit of a lengthy learning time. Traditional attempts have been limited in their flexibility, being designed in the context of homogeneous agents, or require lengthy learning periods based on past observations, limiting their efficiency.

1.1.1 Harsanyi-Bellman Ad Hoc Coordination

A game-theoretic model called Harsanyi-Bellman Ad Hoc Coordination (HBA) has recently been proposed as an efficient and successful solution to such tasks (Albrecht, 2015b). Fundamentally, it uses a set of predefined *types* to characterise the behaviour of the agents in a system, and so acts to reduce the complexity of the system. It assumes that the assignment of types is governed by an

unknown distribution, and furthermore does not assume that opponent types are static, but rather that opponent strategies change dynamically throughout a game. HBA uses its interaction history to form posterior beliefs about their opponent's type distribution by comparing the type prediction with the opponent's actual actions. The beliefs and type predictions are then used in a planning procedure to reason about subsequent turns, permitting the agent to calculate its upcoming actions (Albrecht, 2015a).

1.2 Motivation

1.2.1 Advantages of HBA

HBA provides numerous proposed benefits over other forms of learning, such as traditional opponent modelling or reinforcement learning techniques. By committing to a predefined set of types it requires only a fraction of the observations that other methods require. An example is AlphaGo, the Google project that recently beat the world champion at the game Go. AlphaGo first required training from 30 million recorded historical games, and then played a large number of games against other instances of itself (Silver, 2016). In theoretical ad hoc coordination tasks, in which agents do not know how other agents behave, such numbers of training examples are not possible, and hence alternative methods are required.

Furthermore, as the types HBA utilises define complete behaviours, it can make predictions on unforeseen circumstances, unlike many other methods that rely on having experienced similar events. HBA is also able to respond quickly to significant changes in opposition behaviour as changes to the posteriors require relatively little information, whereas other methods may even require learning to start again. These capabilities satisfy requirements necessary in a host of real-world human-machine interaction tasks such as adaptive user interfaces, robotic care, and automated trading agents (Albrecht, 2015a).

1.2.1 Project Hypotheses

Scalability is one of the prominent challenges for multi-agent learning techniques, as any increase in the number of agent behaviours leads to large increases in the size and complexity of the network of interactions between agents in the system (Panait and Luke, 2005). This can lead to a multitude of problems, not least the rise of emergent behaviour: where unexpected global effects arise from simple per-agent behaviours.

To date HBA has been tested in a few simple domains, including Rock-Paper-Scissors and the Prisoner's Dilemma. This project aims to evaluate whether the technique scales to a more complicated game, namely the board game Othello. Othello was chosen as a suitable candidate environment as it possesses far richer strategies and a larger game tree than the aforementioned games. Whilst Othello is still a long way removed from the complexities of a real world situation, it represents a level up from what has been tested so far. To test HBA in this setting an Othello program was written in which the performance of an HBA agent against a number of agents was measured. The hypothesised type spaces of these agents were determined from the study of advanced strategies of the game.

1.3 Results

The HBA agents outperformed the baseline agents by a significant margin in the semi ad hoc coordination experiments, with a HBA agent using TR-posterior with general time weight achieving win-rates of around 80%. This suggests that the proposed benefits of HBA can be realised in more complex domains. However, when the distributions of the opposition agents were perturbed to generate full ad hoc coordination settings, the performance dropped below that of the best baselines, suggesting that even slight inaccuracies in hypothesised types can nullify the proposed benefits of HBA. Additionally, analysis of the posteriors in the human experiments and experiments into the optimal planning horizon suggest that adaptive behaviours that generate type spaces during gameplay, possibly with the use of other learning methods, may be required for HBA to deal with these problems.

1.4 Thesis Outline

The remainder of the document is laid out as follows:

- **Chapter 2** reviews the theory which is used in the project, including a formal layout of HBA and an overview of the strategies of Othello.
- **Chapter 3** explains the methodology used to create the Othello program and in the experiments run on it.
- **Chapter 4** lays out the results of the experiments performed, and the statistical analysis carried out on them.
- **Chapter 5** presents analysis of the performance, validity and assumptions of the HBA agent.
- **Chapter 6** concludes the project, discusses potential areas of future research, and highlighting shortfalls of the project.

Chapter 2

Literature Review

This chapter serves to lay out the theoretical foundations of the project and introduce the terminology used throughout the thesis. It begins with a brief overview of the rules and strategies of Othello, along with a review of previous Othello programs. After this an overview of the salient theory required to understand HBA is provided.

2.1 The Game of Othello

Othello is a two player game played between Black and White, each of which take turns to place disks on squares on the game board. The winner of the game is the player with the most disks of their colour on the board when neither player has any possible moves left.

2.1.1 Standard Play

As shown in Figure 2.1 the game begins with two white and two black disks placed on the middle squares. The game then proceeds sequentially by turn, with black always placing the first move.

2.1.2 Capturing

Players may only place disks on squares that ‘capture’ opponent disks. This is achieved by placing a disk on a square that is adjacent to the opponent’s disk. In addition the newly placed disk must flank

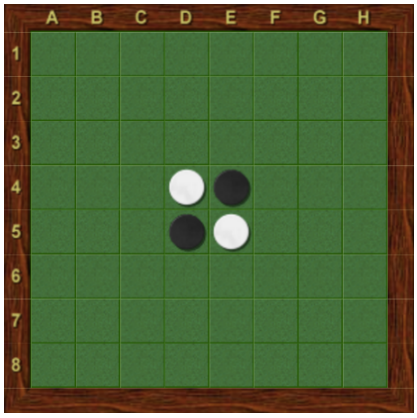

Figure 2.1 Initial board layout (Lazard, 1993)

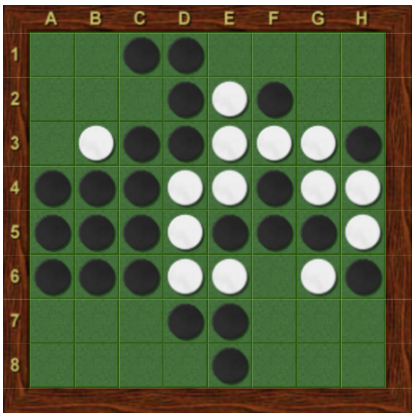

Figure 2.2 Mid-game board layout (Lazard, 1993)

one or several opponent disks between a disk of their own colour which is already on the board. By doing so they 'capture' opponent pieces, replacing them with their own pieces, and hence increase the number of their own coloured disks on the board and consequently decrease the number of opponent disks. If a player has no possible moves on their turn play returns to the other player. The game ends when neither player has any possible moves, which generally occurs when all the squares on the board are occupied with disks.

2.1.3 Terminology

For strategic reasons that are explained in the following subsection, Othello has a certain terminology which needs to be explained, this includes:

- *C-squares* are the those horizontally or vertically adjacent to the corners. These are the A2, A7, B1, B8, G1, G8, H2 and H7 squares on Figure 2.1.
- *X-Squares* are the squares diagonally adjacent to the corner squares, specifically B2, B7, G2 and G7.
- *Frontier disks* are those adjacent to an empty square, whilst the others are called *internal disks.*

2.1.4 Othello Strategy

Knowledge of Othello game play is required to specify the hypothesised types that are the cornerstone of the HBA algorithm. Each type is formed from one of the basic strategies utilised by players of different skill levels. The first, and simplest, of these is the *maximum disk* strategy which consists of the current player choosing to place their disk on the square that captures the highest number of opponent disks in that move (Lazard, 1993). This is a poor strategy due to its lack of recognition of the concept of *stable disks*. These are disks that cannot be flanked, and subsequently provide assured points for that player who they belong to. The most obvious example of a stable disk is one that is placed on one of the corner squares. The implication of these stable disks is that there are some squares which are not good as they give the opponent the chance to take these stable positions, namely C and X squares.

This leads us to the *maximum value* strategy – assigning values to each type of square, and on your move choosing, from the feasible moves, the square with the greatest value. X-squares are assigned the lowest value, followed by C-squares, whilst central squares are given a neutral value. Edge squares which are not on the corner are given a positive value, whilst corners are given the highest.

Another strategy, the *mobility* strategy, is to limit an opponent's *liberties,* that is the number of available moves that the opposition has. As a player is obliged to play a move if they can, limiting a opponent's moves can be done in a way that forces the opponent to make a bad move (Nijssen, 2007). This can often be achieved by making one's frontier as small as possible. A *perfectly quiet* move is one that flips no frontier disks at all.

Being the last player to act is generally a major advantage as the disks they gain on this move are intrinsically stable. This is the concept of *parity.* As Black always acts first, White naturally obtains parity and is at an intrinsic advantage. However, if someone passes their turn, parity changes from White to Black.

Another advanced strategy is the use of *stoner traps*. A stoner trap is a series of moves which, on the

correct board layout, can lead to guaranteed capture of a corner. It requires a player to move into an X-square and then attack an opponent's *weak edge* (an edge including a C-square). The defender cannot respond by taking back the edge as this would flip back the X-square, meaning the attacker could take the corner.

2.1.5 Previous Othello Programs

Various Othello programs have been developed over the past few decades. Older programs are based on the heuristics humans use when playing the game, such as the disk-square and mobility-based evaluation strategies discussed above. More recent programs, however, use only local patterns in their evaluation functions, and utilise large databases of previous games played against experts to calculate statistics for each configuration in a game (Andersson, 2007). In this manner programs have moved on from hand crafted evaluation functions like that used in the program IAGO (1982) to the current best, such as Saio and Edax (Buro, 2003). The value of this project is in the study of HBA in a complex domain and not in the development of an optimal Othello program, and so it was decided not to include a recent program amongst the baseline agents.

2.2 Multiagent Learning

Multiagent learning is a wide ranging term which covers any attempt to create agents that are capable of learning to interact with other agents. As such there is a vast body of literature on the subject covering many different methods that have been theorised to date. This short subsection provides a brief introduction to the aspects of multiagent learning that are used in HBA.

2.2.1 Opponent Modelling

Opponent modelling is defined as any "attempt to learn a model of an agent's behaviour based on the agent's observed actions" (Albrecht, 2015a), and as such falls under the umbrella of multiagent learning. Opponent modelling methods have been used in a wide range of other games such as poker (Johanson and Bowling, 2009) and negotiation games (Hindriks and Tykhonov, 2008). Generally,

attempts in opponent modelling have used a standard behaviour class with parameters that are variable and set during implementation. Such an approach can be seen in Hindriks and Tykhonov's (2008) work on Bayesian learning in multi-issue negotiations. HBA is a type of opponent modelling, but one in which the hypothesised types are each essentially a model, and the parameters are the beliefs over them. This means that HBA it is not limited to a class of behaviours, but has a much greater flexibility and the ability to make predictions for unseen events (Albrecht, 2015b).

2.2.2 Bayesian Games

An incomplete game is one in which there is hidden information, so that not all players are privy to one another's strategies, preferences, or payoffs. Othello is an example of a game of incomplete information, as one does not know the opponent's strategy. Harsanyi (1967) first used the idea of Bayesian games, which use the assumption that the unknown information is governed by some distribution. Specifically, in a stochastic Bayesian game the type of a player $\theta_1^t, \dots, \theta_n^t$ are sampled from Θ_i^+ with probability $\Upsilon(t, \theta_1^t, \dots, \theta_n^t)$. This changes games of incomplete information into complete, but imperfect, information games. In turn, this means that they can be probabilistically analysed, and initial beliefs about types can be updated via Baye's rule. HBA uses such a type-based framework, but the type spaces and distributions are not assumed to be common knowledge, and rather is concerned with ascertaining beliefs over hypothesised type spaces.

2.3 HBA Algorithm

HBA pulls together aspects of opponent modelling and Bayesian games with a focus on performance maximisation. Intuitively, an HBA agent "chooses actions which maximise its expected long-term payoff with respect to what types it currently believes the other agents to have" (Albrecht, 2015b). The algorithm examines the game tree of all possible future moves by itself and the opponent and uses its current beliefs of the likelihood of opponent type and the predicted actions of these types to weight the trajectories.

HBA makes use of the following notation throughout the project:

- The letter i is used to denote the HBA agent, whilst j is used to denote the other player(s).
- H^t denotes the history of actions up to the current state.
- $\widehat{H}$ denotes the projected histories of the game tree.

2.3.1 Type Space

The *full type space* Θ_i contains all possible behaviours for player i, whilst the finite subset $\Theta_i^+ \in \Theta_i$ is the *true type space.* Each type $\theta_i \in \Theta_i$ in this true type space corresponds to a complete behaviour for player i, which means that, given a history, it will return a probabilistic prediction of the next action of the agent by means of its preferences via u_i. During a stochastic Bayesian game, the type of a player $\theta_1^t, \dots, \theta_n^t$ is sampled from Θ_i^+ from the probability distribution $\Upsilon(t, \theta_1^t, \dots, \theta_n^t)$. Albrecht (2015a) notes that this distribution can come in several forms. A type distribution, Υ, *static* if $\forall t, \hat{t}\ \forall \theta \in \Theta^+ :$ $\Upsilon(t, \theta) = \Upsilon(\hat{t}, \theta)$, i.e. that the distribution is not time dependent. Furthermore, a distribution is *pure* if $\exists \theta \in \Theta^+ : \Upsilon(\theta) = 1$, i.e. pure distributions specify one fixed type for each player, throughout the game. However, this project is interested in learning against dynamic players, i.e. distributions that change throughout the game (are *dynamic*) and which do not sample from only one type (are *mixed*).

2.3.2 Formalisation

The HBA agent does *not* know the type distribution of the game, Υ, or true type spaces, Θ_i^+, Rather, HBA assumes a stochastic Bayesian game set up in which the HBA agent only has access to the *hypothesised type spaces*, Θ_i^*, and its history H^t. By comparing its probabilistic predictions of an opponent's actions with the observed actions, an HBA agent calculates posterior beliefs about the relative likelihood of the hypothesised types, $\Pr(\theta_j^* | H^t)$. In other words, given its interaction history, the agent creates a profile for each of its opponents $\theta_j^* = \theta_1^*, \dots, \theta_{i-1}^*, \theta_{i+1}^*, \dots, \theta_n^*$ (Albrecht, 2015a).

$$\Pr(\theta_j^* | H^t) = \frac{L(H^t | \theta_j^*) P_j(\theta_j^*)}{\sum_{\hat{\theta}_j^* \in \Theta_j^*} L(H^t | \hat{\theta}_j^*) P_j(\hat{\theta}_j^*)}$$

In the above equation, $L(H^t | \theta_j^*)$ is the likelihood of H^t if player j has type θ_j^* and $P_j(\theta_j^*)$ is the HBA

agent's prior belief that player j is of type θ_j^*. The likelihood is important as it determines how the agent incorporates its observations when it updates its posterior beliefs. Depending on the nature of the task, different forms can be used which change the posterior between the *product* posterior, the *sum* posterior, and the *correlated* posterior. The Othello HBA agent needs the capability of recognising changes in types in an opponent (i.e. learn a mixed type distribution), and so the sum posterior is formed from the following form of the likelihood.

$$L(H^t|\theta_j^*) = \sum_{\tau=0}^{t-1} \pi_j(H^\tau, a_j^\tau, \theta_j^*)$$

2.3.3 Algorithm

The posterior beliefs and the type predictions are then used to make predictions about future trajectories in the interaction, based on which the HBA computes an optimal response. This optimal response is not inspired by equilibrium attainment, but rather by a best-response rule which maximises the expected payoff, $E_s^{a_i}(\widehat{H})$.

$$E_s^{a_i}(\widehat{H}) = \sum_{\theta_{-i}^* \in \Theta_{-i}^*} \Pr(\theta_{-i}^*|H^t) \sum_{a_{-i} \in A_{-i}} Q_s^{a_i,-i}(\widehat{H}) \prod_{j \neq i} \pi_j(\widehat{H}, a_j, \theta_j^*)$$

where,

$$Q_s^a(\widehat{H}) = \sum_{s' \in S} T(s,a,s') \left[u_i(s,a,\alpha) + \gamma \max_{a_i} E_{s'}^{a_i}(\langle \widehat{H}, a_j, s' \rangle) \right]$$

By doing this HBA is maximising the agents long-term payoff based on what the agent currently believes the opponent's type to be. It does this by exploring the whole game tree of future interactions, weighting them by the current beliefs regarding the relative likelihood of types (Albrecht, 2015). This HBA implementation uses a general tree expansion formula, shown in Figure 2.3, with exhaustive tree expansion.

Repeat:

Observe current state s^t

For all $a_i \in A_i$ **do:**

$$\Omega(a_i) = \left\{ \langle s^t, a^t, \ldots, s^{t+l}, a^{t+l} \rangle \,|\, a_i^t = a_i \right\} \text{ where } l = \min[l^*, t^* - t] - 1$$

$$E(a_i) = \sum_{\omega \in \Omega(a_i)} \left[\prod_{\tau=t}^{t+l} \text{OPPSTRAT}(s^\tau, a^\tau) \sum_{\tau=t}^{t+l} u_i(s^\tau, a^\tau) \right]$$

Sample action $a_i^t \sim \arg\max_{a_i} E(a_i)$

Figure 2.3 Tree Planning Algorithm (Albrecht, 2015a)

2.3.4 Monte Carlo Tree Search

Due to the relative complexity of Othello compared to the previous HBA experiments, a more efficient implementation of HBA is required than exhaustive tree-expansion. To achieve this, *Monte Carlo Tree Search* (MCTS) will be used. MCTS samples a search space by completing many playouts of a game's search tree and using the results of these to weight the nodes, meaning that 'better' nodes are more likely to be selected in future simulations (Wikipedia, 2016). An advantage of MCTS is that it is goal orientated, so it does not require an explicit evaluation function which is good as Othello does not have a developed theory (Browne, 2012). Each round of MCTS can be broken down into four processes; *selection, expansion, simulation* and *backpropagation* as seen in Figure 2.4 (Chaslot, 2010).

- Selection: a selection strategy, balancing exploration and exploitation, is applied recursively until a leaf node, L, is reached. Many different strategies are available, including Upper Confidence bounds applied to Trees (UCT), Probability to be Better than Best Move (PBBM), and Objective Monte-Carlo (OMC).
- Expansion: unless L ends the game, this step adds nodes to the tree.
- Simulation: this step selects moves to play until the end of the game. Random moves may be used, or a pseudo-random strategy could be implemented that plays selected moves

according to heuristic knowledge.

- Backpropagation: propagates the result of the simulated game backwards from L through the nodes it had to traverse to reach L. The value of a node is recorded.

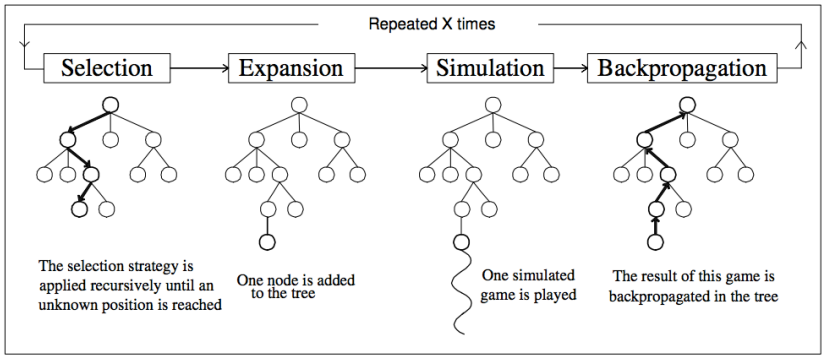

Figure 2.4 Stages of Monte Carlo Tree Search (Chaslot, 2010)

This project will utilise UCT as the selection algorithm. Without UCT, MCTS would select the nodes randomly, meaning that 'good' moves were only selected infrequently. However, by completing multiple playouts and recording the results of each move, UCT utilises this information to weight 'better' moves. UCT selects the child node k of parent p that satisfies the equation below:

$$k \in argmax_{i \in I}(v_i + C \times \sqrt{\frac{\ln n_p}{n_i}})$$

where v_i is the value of the node i, n_i is the visit count of i, and n_p is the visit count of p. C is a coefficient, which has to be tuned experimentally.

Chapter 3

Methodology

This chapter lays out the research methodologies used in this project, along with their justifications and a discussion of their limitations.

3.1 HBA

3.1.1 Formalisation of Types

The hypothesised type space is the primitive library of player types that HBA composes a complex type distribution from. These were decided upon from the theory on Othello strategy. Types must specify complete behaviours and so these ideas must be formalised into rigorous types so that they (i) necessarily return a move on each turn, and (ii) can be encoded. These are discussed one-by-one, and summarised in Table 1. Note that in all cases that if $A_i^t = \{\}$, i.e. there is no possible actions, player i does not make a move and play moves on to the opposition.

1. Maximum disk strategy. On one's turn, make the move that turns the greatest number of disks, selecting uniformly from moves that flip equal number of disks.
2. Maximum value strategy. Each square has a value associated with it, representing how 'good' a move it is. The agent selects uniformly from the squares that have the highest value. This was encoded using a HashMap of values and keys.
3. Mobility strategy. The next state is calculated for all possible moves, and the moves that leads to the state with the smallest opponent mobility (possible actions) are sampled from uniformly.

Note that Table 1 makes use of the following notation:

- A_i^t is a set of all the possible moves for player i at step t.
- g(a) is a function that returns the number of disks gained my move a.
- v(a) is a function that returns the 'value' of action a.
- $\hat{A}_i^t$ is a subset of A_i^t from which a_i^t is chosen in each timestep.

Table 1 Hypothesised Types

Type	Definition
Maximum Disk	$\hat{A}_i^t = \max_{a \in A_i^t} g(a), \quad a_i^t \sim U(\hat{A}_i^t)$
Maximum Value	$\hat{A}_i^t = \max_{a \in A_i^t} v(a), \quad a_i^t \sim U(\hat{A}_i^t)$
Mobility	$\hat{A}_i^t = \min_{a \in A_i^t} \lvert A_j^{t+1} \rvert \quad a_i^t \sim U(\hat{A}_i^t)$

3.1.2 HBA Algorithm with MCTS

A novel implementation of the HBA algorithm was required to account for MCTS, the game's extensive form, and other particulars of the game Othello. The HBA agent takes the inputs `sim_time` and `prior` which specify the length of time to perform MCTS and the prior distribution to use. HBA makes use of two key variables. The first, `opp_strategies`, stores the moves that each type would make in the current state (the predictions), and the second, `prob_history`, stores a binary value according to whether the observed action in the last state matched the prediction made for each type. These are created on the first run of the algorithm, whilst on other iterations they are updated. `updatePosterior()` encodes the likelihood function which manipulates `prob_history` to form a matrix of weights, which is used on `prob_history`. The likelihood is

then multiplied by the prior and normalised to form the posterior.

When the HBA agent has its posterior beliefs it moves on to MCTS. For each playout it probabilistically selects the opponent type from its current beliefs using `chooseType()`. `expandTree()` then performs the selection and expansion tasks, traverses the tree from a given root node until each of the child nodes has been visited, before using the `bestNode()` method to implement UCT. Having selected the 'best' move `simulate()` plays the game until completion. In this implementation, the HBA agent's moves are selected randomly whilst the opponent's moves are selected according to the type previously chosen by `chooseType()`. The *planning horizon* of the agent determines how many steps into the future it assumes that the agent acts according to this predicted type, before resuming with random move selections. The results of the simulation are propagated up the tree by `backPropagate()`, incrementing the play count of every node on the path taken, and incrementing the win count if the simulation led to a win. These variables are then used in the UCT equations. Finally, the node that was visited the most is selected as the best action, with ties being settled by choosing the action that won the most. This algorithm is summarised in pseudo-code in Figures 3.1 and 3.2.

```
Input: prior, sim_time
Output: HBA strategy profile
if step = 1 then
    /* Initiate Workspace                                    */
    posterior = prior;
    CreateOppositionStrategies();
    CreateProbabilityHistory();
else
    /* Update Workspace                                      */
    ExtendProbabilityHistory(last_action);
    UpdatePosteriors(prob_history, likelihood);
end
while time() - now < sim_time do
    /* run MCTS                                              */
    ChooseType(posterior);
    expandTree(state);
    simulate(game_state, opposition_type);
    backPropagate();
    bestAction();
    UpdateOpponentStrategies();
end
```

Figure 3.1 Pseudo Algorithm for HBA Agent

```
expandTree(rootNode);
while true do
    if not legal_moves then
        /* Game is won or turn passes to next player          */
    else if len(curr_node.children ) < len(legal_moves ) then
        /* expand nodes                                       */
    else
        /* Iterate over child nodes and use UCT formula to
           select best node                                   */
    end
end
```

Figure 3.2 Pseudo Algorithm for Simulation and Expansion Method

3.1.3 Choice of Posterior

Sum and product posteriors are only for *independent* type distributions, as opposed to *correlated* distributions which specify constraints on type combinations, such as not allowing a player to be two different types. The types used in this project are all basic strategies that hold no such requirements and as such independent type distributions are assumed. This project is interested in dynamic, mixed distributions and as such the product posterior is not suitable as if the case arises that $\Pr(\theta_j|H^t) = 0$ for a type θ_j, then even if player j's type changes to θ_j some time in the future τ it will still assign $\Pr(\theta_j|H^\tau) = 0$ for that type. To avoid this problem a sum posterior was used, which forms its likelihood out of the sum of probabilities and not the product. Furthermore, as we are assuming that players will change their types throughout a game, the quick reassignment of probabilities will be needed. To facilitate this I used *temporally reweighted posteriors*, as shown in the equation below. The time weight used was the *general time weight*, which has $f(\xi) = \max[0,\ a - b(\xi - 1)^c]$. Different values of the constants a, b and c were experimented with, with the results shown in Chapter 5.

$$\mathrm{L}(H^t|\theta_j) = \sum_{\tau=0}^{t-1} f(t-\tau)\pi_j(H^\tau, a_j^\tau, \theta_j)$$

where $f(\xi) \geq 0$ and $f(\xi) \geq f(\xi + 1)$, for all $\xi \in N^+$.

3.1.4 Choice of Priors

In Bayesian inference, the prior is the distribution that conveys one's beliefs before evidence is taken into account, and its importance has long been the focus of discussion (Kalai and Lehrer, 1993). In the case of these experiments, the prior is the HBA agent's initial judgement as to the relative likelihood of types. Albrecht (2015a) explored the use of a variety of priors in HBA, including the uniform (which sets $P_j(\theta_j^*) = |\Theta_j^*|^{-1}$ for all $\theta_j^* \in \Theta_j^*$), random priors, and priors formed from expected cumulative payoff. Albrecht (2015a) conducted experiments on a set of 78 games, and despite beliefs changing quickly after only a few trial runs, found that in certain cases the formulation of the prior can have a significant impact on performance.

The first case was that the long term performance of HBA can be affected by the prior. He concluded that this was due to the fact that a different prior mean the HBA agent takes different actions at the beginning of the game, which in turn affects the beliefs of the other player as to how HBA plays. This recursive effect can lead to play following different trajectories. However, most of the experiments conducted in this report are simulations against agents that are not learning but rather using a mixed, dynamic type distribution.

The next case was that the depth of the algorithm, the number of steps for which the HBA agent predicts the action of the opponent, can both diminish and amplify the impact of the priors. This occurs because the effects of different types can become much more noticeable after a period of time, meaning that a deep planning horizon can lead to HBA choosing different initial actions. As the first case can be ignored, and the second is explored separately, the impact of the prior was deemed minimal, and for ease of computation, a uniform prior was used throughout the simulations.

3.2 Othello Program

3.2.1 Encoding

The simulations were prototyped on MATLAB due to ease of use: the desktop environment allowing one to work interactively with data. However, MCTS requires recursive program flow which meant a true object orientated language would be more suitable. MATLAB also does not give direct memory control, and as such a more computationally efficient platform was needed and so Python was used to encode the final program.

The `Othello` class was encoded to enforce the rules of the game, with methods such as `getValidMoves()` and `makeMove()`, whilst the `Board` class was used to return state information. Each of the hypothesised types had their own class, whose methods defined what their possible game actions were. In addition the HBA agent utilised a `Workspace` class to maintain the information needed for HBA. The code that implemented MCTS was adapted from that used by Salerno (2016) in his application of MCTS in extended board games. The data representation was as follows: a game state was a tuple comprising of a 8x8 matrix displaying the disk positions of both players and a constant representing whose turn it is. `legal_moves` returned a list of all possible moves, whilst an agent's `get_actions()` method returned a list of possible actions for that agent, on the current state. This is shown in Figure 3.3.

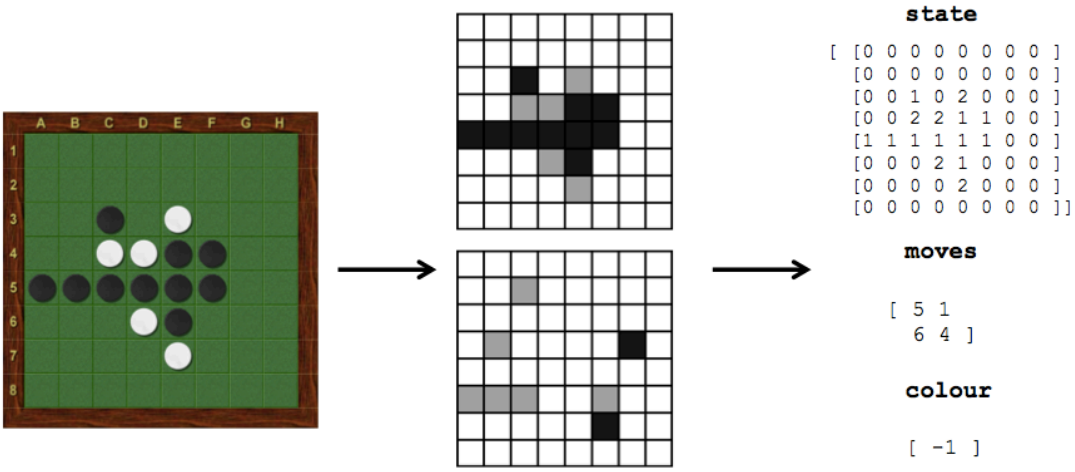

Figure 3.3 Illustration showing how an Othello board is turned into a tuple comprising of the state and colour variables. In this example it is assumed that it is White's turn, and that they utilise a pure Maximum Disk strategy.

3.3 Experiments

Experiments were conducted so as to test the performance of various forms and aspects of HBA against baseline agents. These were conducted in a number of different settings.

3.3.1 Agents

In the non-human simulations, agents were required to play against the HBA and baseline agents. As discussed, these agents had to display mixed, dynamic type distributions. Three such-like agents, `agent4`, `agent5`, and `agent6`, were created and each played the same number of games against each of the HBA and baseline agents. They were defined as follows:

- `agent4`: In each game this agent plays the Maximum Disk strategy for the first 7 moves of the game, the Mobility strategy for the next 7 moves, and finally plays the Maximum value strategy until the game ends.
- `agent5`: This agent changed its type every 5-8 moves to random, but different, type.

- `agent6`: This agent started with a static, mixed distribution but upon losing a game would change its distribution.

The above listed agents all draw their actions directly from the hypothesised types, i.e. $\Theta^{+} = \Theta^{*}$, meaning that the hypothesised types were correct. This is a case of semi ad hoc coordination. Full ad hoc coordination, where the type spaces are not necessarily correct, is a more realistic, and therefore more interesting, domain to study. To simulate a full ad hoc coordination scenario, the above agents were each adapted so 10% of the time they acted randomly, instead of acting according to their prescribed type, creating `agent4_perturb` and so on.

Finally, a small number of experiments were conducted against human participants. As these take much longer, and are more time limited, than simulations, the participants were only pitted against the HBA(gtw) and pure Maximum Value strategy (see next subsection) agents.

3.3.2 HBA Agents

The above agents played against three different forms of HBA agents, these were:

- HBA(gtw): HBA using a sum posterior, with a general time weight likelihood.
- HBA(correct): HBA which was informed of the current true type of the opponent at each time step.

3.3.3 Baseline Agents

Baseline agents were also needed so as to perform statistical analysis and to give reference results. These were chosen to be:

- MCTS: An agent that uses MCTS to expand the game tree for the same `sim_time` as the

HBA agents, but does not implement HBA.

- Pure Maximum Value strategy (MV). Simple Othello programs often use this as its AI, so it was chosen to give a reference performance value.

3.4 Statistical Tests

In order to test the significance of the obtained results, statistical tests were used. Paired, one-sided t-tests were carried out to determine whether the difference in win-rates between agents was significant. This significance level corresponds to the probability of observing such an extreme value by chance. T-tests are used for small sample sizes when the normal distribution cannot be assumed. Two hypotheses are established: the null hypothesis which assumes the two are statistically similar, and the alternative hypothesis, that the two results are not equal. The mean difference between win-rates, $\bar{d}$, the standard deviation of the differences, s_d, and the standard error of the mean difference, $SE(\bar{d}) = \frac{s_d}{\sqrt{n}}$ are all calculated. Using the t-distribution tables the t-statistic, $T = \frac{\bar{d}}{SE(\bar{d})}$, is compared to the distribution, giving the p-value for the test.

-

Chapter 4

Experimental Results

This chapter summarises the setup, results and statistical significance of each of the experiments laid out in the previous chapter, whilst analysis of the results is left for the next chapter. It begins with the semi ad hoc coordination simulations, followed by the full ad hoc coordination simulations and the human experiments. In ends with the experiments conducted on the effects of varying the planning horizon.

4.1 Semi Ad Hoc Coordination Simulations

4.1.1 Setup

For these experiments I used the agents whose dynamic, mixed distributions sample their types directly from the hypothesised types, i.e. $\Theta^{+} = \Theta^{*}$. To ensure that the advantages of parity were accounted for, each HBA agent (and the MCTS baseline) played 150 games against each of the agents, half as White and half as Black, accounting for 450 games each for HBA(cor), HBA(gtw) and MCTS. The MV baseline, whose simulation runs much faster, played 1000 games as both white and black against each agent, accounting for a further 6000 games. The values that the general time weight posterior took were a = 10, b = 0.01, c = 5. This gave relatively strong weight to recent events, meaning the HBA was able to quickly reassign probabilities, and has a fast drop off for events more than 4 steps in the past. The UCT exploration constant used was $\sqrt{2}$. The `sim_time` variable was set to 10 for each agent that has a MCTS component. This meant that on each turn the HBA agent ran simulations for 10 seconds, after which in uses UCT to decide its best action. This value was deemed a sensible choice as it is approximately the same amount as the average time taken by a human to

decide upon a move. For reasons previously discussed, a uniform prior over types was used. Each of the simulations used a planning horizon that plans until the end of the game, but only assumes player types follow current beliefs for the next 10 turns, after which it assumes a random distribution of types. Draws occurred a negligible amount and roughly an equal amount for each colour, and so for ease of collecting results a draw counted as a white win.

4.1.2 Results

The win rates were recorded as follows:

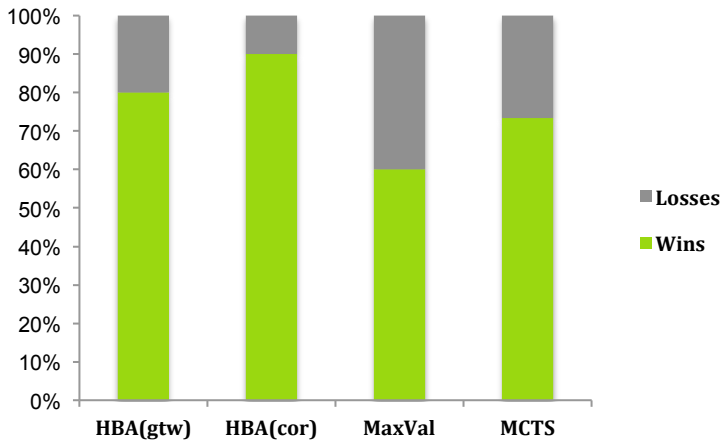

Figure 4.1 Average win-rates against all agents with $\boldsymbol{\theta}^{+} = \boldsymbol{\theta}^{*}$

Please see Appendix A.1 to see the table of statistical significance results. All differences were statistically significant with significance level of (at most) 15%.

4.2 Full Ad Hoc Coordination Simulations

4.2.1 Setup

To simulate a full ad hoc simulation situation, the perturbed agents were used, which acted randomly instead of according to type 10% of the time. Again, each HBA agent (and the MCTS baseline) played 150 games against each of the agents, half as White and half as Black, and 1000 for each

colour for the MV baseline. The general time weight posterior took the same weights, the UCT constant was kept at $\sqrt{2}$, the `sim_time` variable remained equal to 10, and a uniform prior over types was used.

4.2.2 Results

The win-rates were recorded as follows:

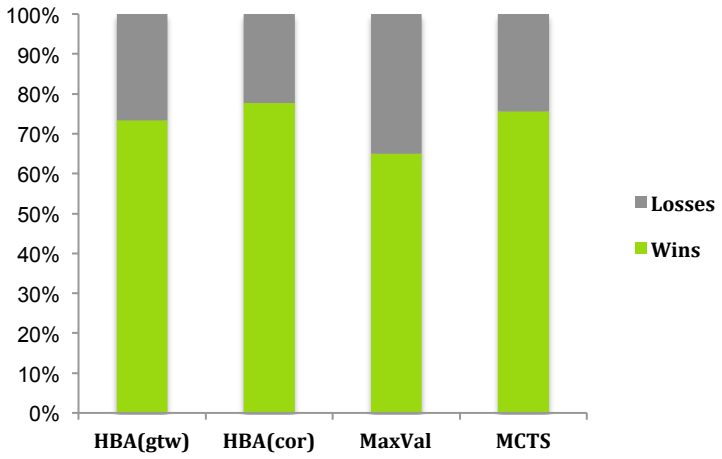

Figure 4.2 Average win-rates against all agents with $\boldsymbol{\Theta}^{+} \approx \boldsymbol{\Theta}^{*}$

4.3 Human Experiments

4.3.1 Setup

A small number of experiments were conducted that pitted human players against the HBA(gtw) agent and the MV baseline. Only 4 participants were studied, each playing 4 games against each agent, for a total of 32 game results. Of the four participants, 3 had a low Othello skill level and 1 had a high skill level. Whilst not significant to draw generalizable results, it provided another realisation of a full ad hoc coordination scenario, and provided interesting data on the HBA agent's posterior belief development against real players. Again C = $\sqrt{2}$, a = 10, b = 0.01, c = 5, `sim_time` = 10 and

a uniform prior used.

4.3.2 Results

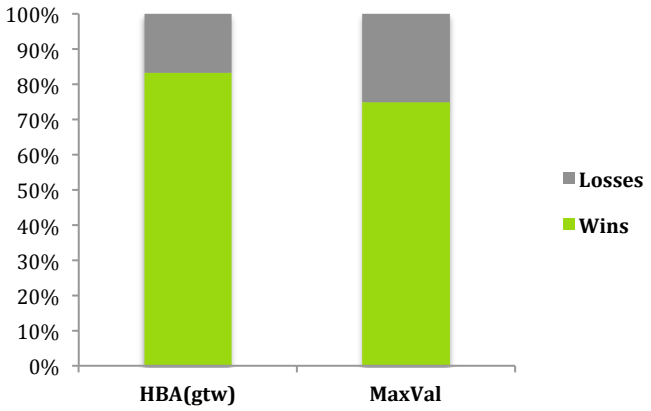

Figure 4.3 Average win-rates against human participants

The posteriors were tracked over the human experiments, and used to create the following graphs. Figure 4.4(a) shows the values that the posteriors of the HBA(gtw) agent took over one game against human opposition, in a game that the human won. Figure 4.4(b) represents statistics captured on (i) the number of types used by a human during a single game and (ii) the average duration of a type. The number of types was obtained by calculating the number of times that the HBA agent assigned its highest belief to a specific type for consecutive turns. The duration was calculated by dividing the length of a game (30 sequential 'turns') by the number of types used in a game. Along with the mean values, the graph also shows the maximum and minimum values these took.

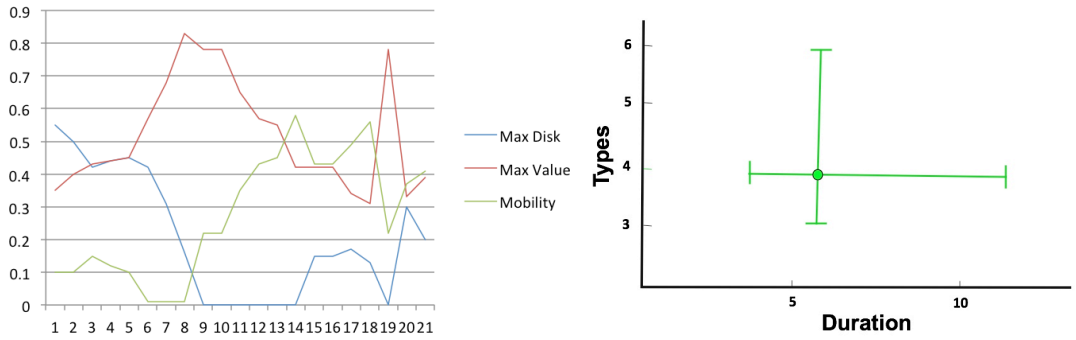

Figure 4.4 (a) Posterior belief changes throughout a game (b) Average number of types and duration during human experiments

4.4 Planning Horizon

In the HBA algorithm, during each MCTS simulation playout, it is assumed that the opponent is a specific type, chosen probabilistically based on current beliefs. The number of steps that it is assumed they remain this type is variable, but of importance. The generation of playouts are weighted towards (what the HBA agent believes) are the most likely types. However, we are also assuming that player distributions are dynamic, i.e. they change over time. As the consequences of being a particular type can diverge as the game progresses, the playouts generated will vary differently from what will occur if the player changes its type during the game. The number of steps that the HBA predicts player actions for before MCTS reverts to finishing the simulation with random actions is what I call the *time horizon* of the algorithm. For these experiments the HBA(gtw) agent was played against `agent4` for 150 games as both Black and White, with a planning horizon of infinity (i.e. until the game ended) and again with a planning horizon of 5. This value was selected as it coincided roughly with the frequency with which both the programmed agents and the humans in the experiments changed their types. All other constants were kept the same as in the previous simulations. The win-rate results are shown below:

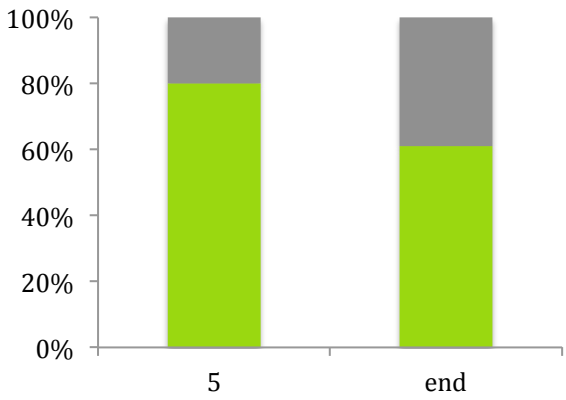

Figure 4.5 HBA(gtw) win rates against agent4 with varying planning horizons

Related to this concept are the general time weight constants, a , b, and c, used by the likelihood function. The effect these have is to weight more recent events differently, as shown in Figure 4.7(b). For these experiments the same settings were used to the above experiment, but the planning horizon kept at 5 steps.

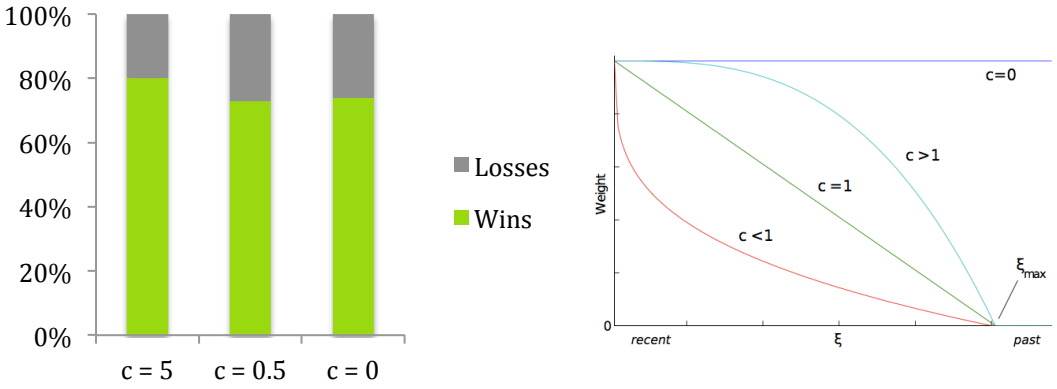

Figure 4.6 (a) Win-rates with varying the GTW parameter c (b) The GTW weight with parameter c varied (b and c remain constant) (Albrecht, 2015a)

Chapter 5

Discussion and Analysis

This chapter reviews and discusses the results laid out in the previous chapter.

5.1 Semi Ad Hoc Simulations

HBA(cor) achieved the highest win-rate in the semi ad hoc coordination simulations, winning an average of approaching 90% of its games against the agents. This is to be expected as this is a measure of the potential of HBA as it can plan optimally, in the knowledge of what moves the opponent will make given specific board layouts. HBA(gtw) had a win-rate of 81%. This drop in performance can be put down to two factors: (i) an overlap in type action predictions, meaning that an HBA cannot always correspond a move to a type, and (ii) a delayed response in beliefs to real changes in opposition type. The MCTS baseline achieved a better win-rate than expected, winning 73% of its games, whilst the MV baseline won 61%, showing that a pure Maximum Value strategy performed better, on average, than the dynamic, mixed distributions that the agents used. Both HBA agents and the MCTS baseline gave statistically significant improvements in performance compared to the MV baseline, and in turn the HBA agents gave significant improvements on the MCTS baseline. For a full layout of the t-scores, please see Appendix 2.

5.2 Full Ad Hoc Simulations

The results from the full ad hoc coordination simulations were less expected, and show how damaging using incorrect types can be. The performance of the baselines against the agents did not change significantly, suggesting that the perturbed strategies were not significantly worse than their

unperturbed counterparts. However, the performance of HBA(gtw) against the agents dropped by 10%. The deviated actions were taken as a change in type by the HBA agent, which adapted its beliefs accordingly, leading to the reduced performance. Whilst this random perturbation may not mirror how real agents may deviate, it does highlight the importance of the correct type space. Ominously, the MCTS baseline outperformed the HBA(gtw) agent. This is significant as it suggest that, not only can the benefits of HBA be voided, but that HBA can decrease the performance of an agent if its hypothesised type space is different enough from the true type space.

5.3 Human Experiments

The results of the human experiments must be treated with the most scepticism, as the sample size severely limits the generalisability of any conclusions drawn from them. However, some conclusions can be drawn. Firstly, the rate of the baseline was higher, suggesting that the level of play was lower than the programed agents. The HBA(gtw) won all of its games, apart from some against the more experienced player. The analysis of the posteriors gave insights as to how players approached the game. It can be seen that in an average game a player would change between types 4 times. Players tended to play a maximum value strategy at the beginning of the game, and would adopt mobility and maximum disk strategies later in the game.

5.4 Planning Horizon

The performance of the HBA(gtw) agent dropped dramatically when it used an infinite planning horizon. HBA's advantage stems from its use of beliefs over types so it can predict opponent moves. However, if its assumptions of the opponents type are incorrect then this will lead the agent to game states that will not occur, and subsequently inhibit performance. If the opponent changes their type, then the agent's beliefs will be out-dated and a hindrance. Conversely, an HBA agent with a myopic planning horizon will not receive all the benefits from the HBA algorithm. Further work could implement the horizon parameters as variables so that task specific horizon parameters are learnt during play.

The c values affected the way in which the HBA agent's beliefs were formed. Having c = 0 meant that the beliefs were influenced equally by all past actions and lead to less extreme belief values (which tended to lie in the range 0.3-0.4). This affected performance as beliefs informed by recent actions are required if an opponent has a dynamic type distribution (and is therefore is changing their type). The parameter value c = 0.5 implements general time weight, but with a far shallower drop off than c = 5 that was implemented in all the other experiments. This meant that beliefs tended to be weaker and potential type ambiguities more frequent.

5.5 Further Discussion

5.5.1 Correctness of Posterior Beliefs

The posterior beliefs dictate the actions chosen by the HBA agent and so are extremely important, but can the correctness of the posteriors obtained during the simulations be measured? Overlap describes the similarity of the types, where $\mathrm{AO}_j(H^t) = 0$ means that player j's types (on average) never choose the same action in history H^t, whereas $\mathrm{AO}_j(H^t) = 1$ means they behaved identically (Albrecht, 2015a). Albrecht (2015a) continues by showing that that if the overlap converges to zero as t goes to infinity, the sum posterior is guaranteed to converge to any mixed type distribution.

$$\mathrm{AO}_j(H^t) = \frac{1}{t}\sum_{\tau=0}^{t-1}[\left|\Lambda_j^\tau\right| \geq 2]_1 \sum_{\theta_j^* \in \Theta_j^*} \pi_j(H^\tau, a_j^\tau, \theta_j^*)\left|\Theta_j^*\right|^{-1}$$

$$\Lambda_j^\tau = \{\theta_j^* \in \Theta_j^* \mid \pi_j(H^\tau, a_j^\tau, \theta_j^*) > 0\}$$

where $[b]_1 = 1$ if b is true, else 0.

The type spaces used in this project have a high overlap, due to the nature of the game Othello, meaning that often different types result in similar trajectories. As the HBA agent's only means of capturing information is through its observations of opponent actions, if there is ambiguity in which type led to an action, this will introduce redundancies and possibly lead to the HBA agent learning

incorrect type distributions. As the type spaces are made by design, it is hoped that they can be created in such a way to avoid this, but these experiments show that this is not necessarily the case in all problem domains. This issue will only amplify as they increase in complexity.

5.5.2 Correctness of Hypothesised Types

The question remains, how close to the true type spaces do the hypothesised type spaces need to be? The results from the full ad hoc coordination simulation suggest that it is closer than one might expect. If small random perturbations can lead to a drop in win-rate in this domain of 10%, one would expect this effect to be magnified in a more complex one. More worryingly, an incorrect type space has the ability to lead an HBA agent into believing that it is choosing the right actions to solve a task whereas in actual fact it is performing far from optimal actions. Such a type space is called a critical type space (Albrecht, 2015a). Othello, whilst having its nuances and subtleties, can be reduced to types more easily than problem domains that require interpretation or other human eccentricities. It is easily imagined that as the task gets more complicated, the ability to accurately form types get much harder. Further research should explore advanced methods for measuring the extent that the types in use can be trusted, both before and during tasks.

5.6 Other Observations

5.6.1 Othello observations

The experiments showed that playing as White increased one's win-rate by around 3%, or an average of 4 more disks. Also, the HBA agent can be seen to perform some of the more higher-level Othello strategies not directly encoded, which gives credence to it as an Othello program.

The following graph shows the minimum, maximum and mean disk counts of the semi ad hoc simulations. It shows that score is not a great indicator of performance and explains why win-rate was chosen as the primary measure in the experiments.

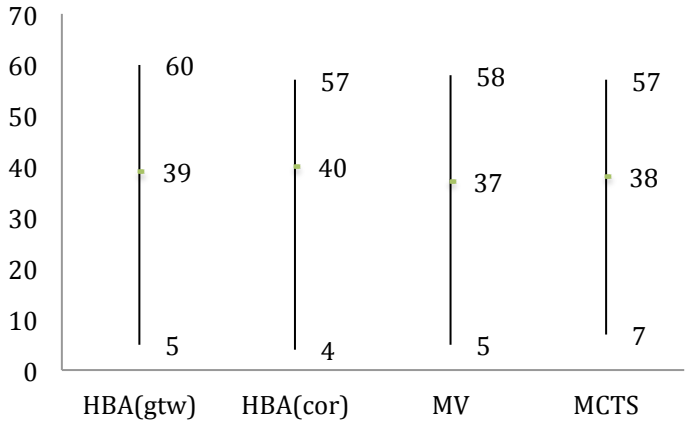

Figure 5.1 Maximum, minimum, and mean disk counts of different agents in the semi ad hoc coordination simulations

5.6.2 Runtime Analysis

As with any learning agent, the computations get more expensive as the complexity of the domain increases. Whilst HBA is computationally cheap in comparison to many reinforcement learning techniques, performing searches over complex domains gets expensive quickly. The most computationally expensive aspect of this project was the generation of the MCTS playouts, the gain being that the more runs the agent completed, the better its performance. In the 10 second run time per turn, the HBA(gtw) agent completed 338 simulations run to completion in the early stages, 582 simulations in the middle stages, and 1712 in the later stages, when there are not many moves left. For MCTS, which ran less code, 3004 simulations were run to completion in the early stages, 3526 simulations in the middle stages, and 20970 in the later stages. Note these figures were taken from a sample game, and are not averages.

Further techniques could have been used to improve the search speed. Alpha-beta pruning, or a similar algorithm that recognises ‘killer moves’ and focus their search on these, would improve speed. Preliminary shallow searches of around 5 steps into the future take negligible time and could be used to weed out moves that are obviously bad.

5.7 Othello - a good choice of game?

The game of Othello was chosen due to its increased complexity in comparison to those domains that HBA has been used in previously. It proved to be an interesting domain and one that facilitated the use of HBA quite well. With hindsight, however, there were some issues with it. These included the fact that there is not a large variety in the potential strategies, which in turns limited the number of types that would both be realistic and successful. The limited move space strategies intrinsically led to overlap in types, which raised issues with the posterior beliefs.

Othello is a specific game, and as such the generalisability of these results is restricted. To draw further conclusions as to HBA's potential it will need to be used a wide range applications. With ad hoc coordination being key to many innovative tasks that are at the forefront of much of the research in artificial intelligence, there are high hopes that HBA can prove an important component in their solutions.

Chapter 6

Conclusions and Recommendations

This chapter enumerates limitations of the project and potential avenues of future research, before concluding the thesis.

6.1 Limitations and Recommendations

Observations from the experiments highlight several limitations of the project, some self-imposed and some that were unavoidable. Examination of these shortfalls leads to recommendations for future research. These include, but were not limited to the following:

- No consideration was given to the fact that players learn as they play. In reality, during games a player will model you, just as you are modelling them (Vidal and Durfree, 2003). This may lead to them adopting non-standard play - game-play that would not be considered under normal circumstances. Analysis on my results did not account for this phenomena, with all results being averaged to collect the win-rates. It was also not considered in the hypothesised type spaces.
- This leads on to idea of conceptual types, which were not touched upon during this project. HBA is capable of including methods for opponent modelling, so that whilst playing a game it can build novel player types from experience. This may be of critical importance if the hypothesised type spaces and true type spaces are significantly different. These would serve to improve the performance of the agent as well as provide information as to how opponents act by revealing hidden player types. Conceptual types are essentially a way of ensuring that

type spaces are eventually uncritical.

- The addition of reinforcement learning procedures into my algorithm could have made the HBA implementation more efficient. For example, Q-learning would increase the efficiency of the tree search by storing information about 'good' states that can be used in future runs, resulting in better performance (Wiering and van der Ree, 2013). In addition, a database of commonly occurring board layouts from games of Othello played against expert opposition could be accumulated. Layouts could be deemed identical if they are translations of other layouts to minimise search time (Stern et al., 2006).
- The number of human experiments was far below that required to draw reliable conclusions. To collect a meaningful test size future research could take the form of an online application to ensure a meaningful sample size.

6.2 Conclusion

The aim of this project was to test whether HBA scales to a more complex domain, and how it responds to the challenges this poses. The experiments performed showed that HBA can indeed scale successfully to a more complex domain, specifically that of the game Othello. This was shown in the significantly higher win-rates that it achieved in comparison to those of the baseline agents also used to play the game. This provides optimism for HBA, and by showing it has potential, this project provides support for HBA to be implemented in an increasing number and variety of ad hoc coordination applications. The project also serves to highlight areas which need attention going forward, namely how best to determine robust hypothesised type spaces and flexible planning horizons. In these increasingly complex tasks it will be used in, HBA will be required to cope with inaccuracies in types and uncertain task information, without suffering from reduced performance.

Appendix

A.1 Statistical Test Results

The following are the statistical significance results from the semi ad hoc coordination simulations.

		HBA(gtw)	HBA(cor)	MV	MCTS
HBA(gtw)	t-statistic	-	1.54	14.18	4.29
	p-value	-	0.124	0.0035	0.026
HBA(cor)	t-statistic	1.54	-	21.4	6.13
	p-value	0.124	-	0.0014	0.015
MV	t-statistic	14.18	21.4	-	3.06
	p-value	0.0035	0.0014	-	0.0175
MCTS	t-statistic	4.29	6.13	3.06	-
	p-value	0.026	0.015	0.0175	-

A.2 Code

Please find all the associated source code in the software file, along with a README file. A copy of the README file is included here.

READ ME

This directory includes code created by Edward Stevinson for the thesis entitled 'Utilising Policy Types to Achieve Effective Ad Hoc Coordination in the Game Othello'. It includes a MATLAB file and a PYTHON directory. The code is all commented for ease of understanding.

PYTHON:

Includes the Python code that was used for the experiments. To run, use run_game.py, and on line 93 choose the agents you want the game to simulate, and the number of games to simulate. The basic MCTS capabilities were taken from Salerno (2016), and where code is reused it is noted. The most important files enclosed are:

othello.py - Class which enforces the rules of the game.

board.py - Class that

workspace.py - Class that manipulates the information needed for HBA

hba_agent - agent which chooses its moves via HBA

human_agent - Human agent

max_disk_agent - agent which chooses the move that flips the most disks

MATLAB:

MATLAb was used for prototyping during the early stages of the project. The structure is inspired by Stephen Albrecht's experiments and where code is reused it is noted. The most important files are:

simulate_othello.mat - The main script that runs the simulations.

tester.mat - Provides the inputs for simulate_othello.mat.

Bibliography

Albrecht, S, 2015a, Utilising Policy Types for Effective Ad Hoc Coordination in Multiagent Systems, PhD Thesis, University of Edinburgh.

Albrecht, S, Crandall, J, & Ramamoorthy, S, 2015b, Belief and Truth In Hypothesised Behaviours, Preprint submitted to *Artificial Intelligence*.

Albrecht, S, Crandall, J, & Ramamoorthy, S, 2016, E-HBA: Using Action Policies for Expert Advice and Agent Typification, Association for the Advancement of Artificial Intelligence.

Andersson. G, 2007, Writing an Ohello Program, Available at <http://radagast.se/othello/howto.html> Accessed: 14 August 2016 .

Browne, C, Powley, E, et al, 2012, A Survey of Monte Carlo Tree Search Methods, *IEEE Transactions on Computational Intelligence and AI in Games 4.*

Buro, M, 2003, The Evolution of Strong Othello Programs, in : *Entertainment Computing – Technology and Applications,* Nakatsu and Hoshino, J, pp. 81-88.

Chaslot, G, 2010, Monte-Carlo Tree Search, Dutch Research School for Information and Knowledge Systems.

Harsanyi, J, 1967, Games with Incomplete Information played by 'Bayesian' Players, *Management Science*, vol. 14 pp. 159-182

Hindriks, K, and Tykhonov, D, 2008, Opponent Modelling in Automated Multi-Issue Negotiation using Bayesian Learning, Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 1 pp. 331-338.

Johanson, M, and Bowling, M, 2009, Data Biased Robust Counter Strategies, In: *Proceedings of the 12th International Conference of Artificial Intelligence and Statistics*.

Kalai, E, and Lehrer, E, 1993, Rational Learning leads to Nash Equilibria, Econometrica, vol. 61(5) pp. 1019-1045

Nijssen, J, 2007, Playing Othello Using Monte Carlo, MSc Thesis Maastricht University.

Panait, L, and Luke, L, 2005, Cooperative Multi-Agent Learning: The State of the Art, George Mason University PhD dissertation.

Salerno, A, 2016, Winning Reversi with Monte Carlo Tree Search, Available at <https://andysalerno.com/2016/03/Monte-Carlo-Reversi>, Accessed on 15 August 2016.

Silver, D, Huang, A, Maddison, C, et al., 2016, Master the Game of Go with Deep Neural Networks and Tree Search, Nature, vol. 529 pp. 484-489.

Stern, D, Herbrich, R, and Graepel, T, 2006, Bayesian Pattern Ranking for Move Prediction in the Game of Go. In Proceedings of the 23rd International Conference on Machine Learning, pp. 873-880.

Stone, P, Kaminka, G, Kraus, S, and Rosenschein, J, 2010, Ad Hoc Autonomous Agent Teams: Collaboration without pre-coordination, In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pp. 1504-1509.

Vidal, J, and Durfree, 2003, E, Predicting the Expected Behaviour of Agents that Learn about Agents: the CLRI framework, *Autonomous Agents and Multi-Agent Systems*

Wiering, M, and van der Ree, M, 2013, Reinforcement Learning in the Game of Othello: Larning Agaisnt a Fixed Opponent and Learning from Self-Play, *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

Wikipedia, 2016, Monte Carlo Tree Search, Available at < https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>, Accessed on 16 August 2016.