# Converging to a Player Model in Monte-Carlo Tree Search

Trevor Sarratt
University of California at Santa Cruz
Santa Cruz, California 95064
tsarratt@soe.ucsc.edu

David V. Pynadath
USC Institute for Creative Technologies
Los Angeles, California 90094
pynadath@usc.edu

Arnav Jhala
University of California Santa Cruz
Santa Cruz, California 95064
jhala@soe.ucsc.edu

***Abstract*—**

**Player models allow search algorithms to account for differences in agent behavior according to player's preferences and goals. However, it is often not until the first actions are taken that an agent can begin assessing which models are relevant to its current opponent. This paper investigates the integration of belief distributions over player models in the Monte-Carlo Tree Search (MCTS) algorithm. We describe a method of updating belief distributions through leveraging information sampled during the MCTS. We then characterize the effect of tuning parameters of the MCTS to convergence of belief distributions. Evaluation of this approach is done in comparison with value iteration for an iterated version of the prisoner's dilemma problem. We show that for a sufficient quantity of iterations, our approach converges to the correct model faster than the same model under value iteration.**

## I. Introduction

Evaluating a player's actions and infering a strategy or player type are useful capabilities for an agent within a game. When human players interact in a virtual environment, they can quickly deduce, via their experience and expectations, both what another player is intending to do and the reasons behind such actions. This allows for quick reactions and counters in competitive games as well as cohesive teamwork in collaborative settings. It is natural, then, that AI in games can benefit from having similar reasoning ability.

It is no surprise that humans can act in unexpected ways within a game, often in a manner considered sub-optimal from the perspective of an AI, given a relatively straightforward goal such as navigating to the end of a level, defeating an opponent, or maximizing a score. Players may have ulterior motives within a virtual environment based on their own preferences and goals. For games, the use of models with explicit representation of beliefs and preferences of agents can allow agents to predict such behavior. Player modeling is a common pursuit in artificial intelligence, one with many challenges such as how to learn, predict actions of, plan against, and adapt to changes in another agent's behavior.

In this paper, we are primarily concerned with the capability of an agent to efficiently plan under the uncertainty of a player's model as well as converge to a correct player model based on the actions taken by the player. Considering many potential opponent models can restrict game tree pruning, which in turn can impact performance [1]. Fast convergence to the correct model diminishes the uncertaintly, allowing the agent to plan more effectively.

Model recognition in multi-agent domains exists as an alternative to building a model during play. Given a set of possible models, observations on actions taken by a target are used to inform an agent of the target's likely model. Model recognition can then be performed by using a classifier [2], [3] or by updating a probability distribution over models [4]–[6]. In the latter case, one must be able to evaluate the probabilities of observed actions given the candidate models.

This paper investigates the application of player models to Monte-Carlo Tree Search, a sampling-based, anytime planning algorithm that been proposed as a promising technique for agent planning in board and video games [7]. We characterize how various parameters of MCTS affect the convergence of beliefs over models. Furthermore, we describe how information used in MCTS can be leveraged for belief updates with modifications to the algorithm. This provides the benefit of inferring which model an agent may be using while simultaneously planning in the game's state space.

For evaluation, we adapt an iterated form of the prisoner's dilemma (IPD) such that an agent can have hidden incentive to favor one of the two actions, collusion or betrayal. We test the ability of our MCTS adaptation to converge correctly and efficiently to the appropriate model. We also compare the results against an established social simulation tool, PsychSim [8], which employs value iteration—a full-width search—for the same purpose. IPD provides an established, well-studied game with discrete actions, useful for anlayzing belief convergence, whereas a more complex domain may add confounding factors or noise to our results as well as limit the application of value iteration for comparison.

## II. POMDPs

In the case where agent models are finite, static, and deliberate in action, the problem of uncertainty in an agent's true model can be conceptualized as a single-agent partially observable Markov decision process (POMDP) [9]. The POMDP representation has been applied in a wide variety of domains [10] and has frequenly been used for modeling players in games [11]–[14]. For convenience and consistency, we will

978-1-4799-3547-5/14/1.00 

adopt the representation of a POMDP while discussing belief distributions as applied to MCTS.

A POMDP is a generalization of an Markov decision process (MDP) where some aspect of the world state is not directly observable to an agent. In this paper, we restrict uncertainty to the model used by a deterministic agent in the game. In this framework, a POMDP can be represented as a tuple $\langle S, A, \Omega, B, R \rangle$, where

- $S$ is the set of world states, comprising of both the game state and the type of the agent in question. Due to the latter uncertainty, an agent cannot directly observe the full world state.
- $A$ is the set of actions available to the agents.
- $\Omega$ defines a set of observations on actions occuring in the game. In this case, we allow for all actions to be observable at all times.
- $B$ describes the set of belief states possible given an initial state and a set of observed actions.
- $R$ relates the common definition of a reward function in MDPs, where $R : S \times A \to \mathbb{R}$.

In order to plan successfully in an POMDP, agents must utilize an observation history to update their beliefs over time. As we define observations to be derived from actions taken in the game, a history can be defined as $h_t = \langle a_1, a_2, ...a_t \rangle$. Beliefs are then described as a distribution over possible states given those histories, or $b_t = Pr(s|h_t) \quad \forall s \in S$. The goal of planning in a POMDP is to find a policy $\pi$ maximizing the agent's expected reward, as given by $J^\pi(b_0) = \sum_{t=0}^{\infty} E[R(s_t, a_t)|b_0, \pi]$. This can be done by searching through the belief space to identify the optimal value function, $V^*(b_t) = \max_{a \in A} \left[ R(b_t, a) + \sum_{o \in O} \Omega(o|b_t, a) V^*(b_{t+1}) \right]$, where $\Omega$ is a probability distribution over observations.

## III. Monte-Carlo Tree Search

Monte-Carlo Tree Search is a search algorithm based on Monte-Carlo simulations within a sequential game. Through evaluating potential states by averaging simulation outcomes, MCTS holds several advantages over other search methods. By sampling, it bypasses the curse of dimensionality of large numbers of state transitions. Black box simulations can be used for problems too complex to represent fully, and it can be used effectively without prior domain knowledge [15]. In addition, it converges to an optimal policy in fully observable and partially observable problems, given an appropriate exploration function [16], while also being an anytime approach.

As previously mentioned, MCTS has been proposed as a promising approach for developing an action policy within the state spaces of board and video games [7] and has already seen successful application to games such as computer Go [17], General Game Playing [18], poker [19], and Ms. Pac-Man [20]. In 2009 FUEGO, an MCTS Go program, beat a top human professional at 9x9 Go [21].

MCTS performs a large number of simulations of a game, from the current player state to the end of the game. As a new simulation begins searching through the game tree, MCTS considers information gathered from previous playthroughs.

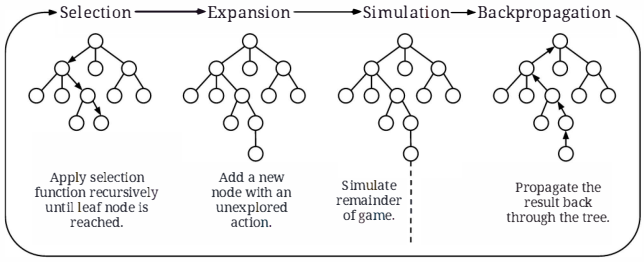


Fig. 1: Outline of Monte-Carlo Tree Search

Specifically, in each step of the game, MCTS selects the next action with a bias toward those with a higher success rate for the player. When an action is taken for the first time, the rest of the game is played out randomly. The resulting score or reward value is then back-propogated through the explored nodes. Over many simulations, the program focuses on better moves, leading to farther look-ahead without giving as much consideration to inferior moves. We will now describe this process in more detail.

In an MCTS game tree, every node represents a state of the game. Each node aggregates statistics from the simulations that have previously passed through that particular node. Specifically, the nodes contain two pieces of information:

- *Value* The value of the game state. Typically, this is the average of the cumulative rewards of all simulations that visited the node.
- *Visit Count* The number of simulations that have reached the game state represented by the node.

For MCTS, the root node represents the starting state of the game and is the only node present in the tree initially. By repeating four steps outlined in Figure 1, the tree is explored until a number of maximum simulations has been reached or there is no time remaining for exploration. We will describe these steps in detail as well as discuss some of the implementations we have chosen for this paper:

- *Selection* Potential actions at each node are selected in a fashion that balances exploration and exploitation. The idea behind such a heuristic is to progress deeper into the game tree through actions with higher expected value (exploitation) while periodically trying less favorable moves that may have yet unseen value (exploration). We use Upper Confidence Bounds applied to Trees (UCT), a popular, efficient algorithm for guiding MCTS [15]. In this search technique, an action is selected according to the following criteria:

$$a^* = argmax_{a \in A(s)} \left\{ V(s,a) + C\sqrt{\frac{2 \ln n_p}{n_a}} \right\}$$

where $a \in A(s)$ represents an action within the set of available actions for state $s$, $V(s,a)$ is the average value for action $a$ stored in the tree node representing state $s$, $n_p$ is the number of visits for the current node, and $n_a$ is the visit count for the child node resulting from the

action. $C$ is a parameter to tune the search, where higher values encourage more exploration.

- *Expansion* When a leaf node is reached, a new action is selected for expansion. The resulting state is added as a new node to the tree. In this way, the tree is expanded by one node for each simulation.
- *Simulation* From a newly added node, the remainder of the game is performed by selecting actions according to a set policy until the game ends. For this paper, we simulate the remainder of the games by selecting actions randomly.
- *Back-propagation* Once the end of the simulated game is reached, we update each node that was traversed during the simulated playthrough of the game. The visit count for each node is incremented, and the endgame values are incorporated into each node's cumulative average.

When the maximum number of simulations has been reached, an action is returned. This can be the action with the highest expected value or that with the highest visit count. We choose the latter approach.

## IV. Alterations to MCTS

MCTS in its basic form does not account for partial observability in a world state of a game. Various adaptations of MCTS have been proposed to handle types of partial observability. Cowling et al. [22] apply MCTS to games with hidden information and uncertainty, but do not address belief convergence and inference. Silver and Veness [16] introduced a variation of UCT search for general application to POMDPs using a particle filter to update beliefs. The approach was empirically demonstrated on the games battleship and partially observable PacMan. A comprehensive survey of alterations to Monte-Carlo Tree Search for diverse application can be found in [23]. We outline our enhancements to the algorithm in this section.

### *Reward Scaling*

In common applications of MCTS, a binary win/loss result is a sufficient value to track and maximize throughout the space exploration. However, as we intend to work with domains with continuous rewards, it is clear that a static exploration factor in MCTS may have reduced effectiveness over time with a cumulative, unbounded reward. To keep both exploration and exploitation in balance, it is necessary to either adjust the expected values within the game tree or devise a method of choosing an exploration factor dynamically.

To solve this problem, before initiating MCTS, we sample the game with random moves a set number of times. This gives us a rough distribution of possible reward values. We scale expected values by the difference of the maximum and minimum values sampled beforehand. These scales are calculated per model, as each model may have reward values of an arbitrary magnitude.

### *Node Information*

In MCTS, each node records the average value for the game state. For multi-agent problems, this value corresponds to the expected value for the agent who acted immediately prior to the current game state. The heuristic used to select an action can then maximize the expected value for each agent during their respective turns. Under the uncertainty of which model an agent is using, it is necessary to reason over the results of each action in the context of each potential model.Therefore, rather than retaining a single expected value for an agent, nodes keep a vector of values corresponding to the set of rewards achieved by each independent preference model of an agent.

### *Node Selection*

Since each node now contains multiple values corresponding to the multiple possible agent models, we must reconsider the method of selecting actions while exploring the game tree. It is desirable to retain the benefits provided by UCT, namely balancing exploitation of previously observed actions with exploration of potentially inferior moves. Yet, the agent must also consider such actions in the context of various models. In order to accomplish both objectives during the selection phase, we propose sampling a single model according to the current belief distribution, then performing the UCT selection over actions using expected values specific to the chosen model.

### *Back-propagation*

Rather than propagating results of one model of an agent back through the game tree, each model's rewards are tracked through the simulation of the game. Each of these results is then propagated to the visited nodes for the purpose of updating their estimated values.

### *Belief Update*

As in existing approaches to mental modeling [6], we employ a Bayesian approach to updating belief distributions over said models. Starting with an initial distribution of model probabilities, we update according to Bayes' Theorem, as seen in Equation 1 where $m$ represents a model and $a$ indicates an action.

$$P(m|a) = \frac{P(m) \times P(a|m)}{\sum_i P(m_i) \times P(a|m_i)} \tag{1}$$

The difficulty with this particular update strategy is determining $P(a|m)$, the probability of an action given a particular model. Several methods of calculating such probability can be found in [6]. For fairness of comparison with PsychSim in our test case, we adopt a distribution model given by:

$$P(a|m) = \frac{e^{V_a(m)}}{\sum_{a_i \in A} e^{V_{a_i}(m)}} \tag{2}$$

## V. Evaluation

To evaluate our implementation, we tested the approach on a sequential version of the iterated prisoner's dilemma problem with full visibility only of actions taken, i.e. no direct observation of the other player's value model. The goal of an agent, then, is to infer the correct player model of the other agent and adapt its own strategy accordingly.

In this setup, two players must complete the prisoner's dilemma scenario twenty times. Each player has two actions, defect or remain silent. If only one player betrays his partner, he receives a high reward while his partner suffers a low reward. If both players betray the other, both receive a moderately low reward. If both players remain silent, they each receive a moderately high reward. The specific payoffs used in our tests are given in Table I.

TABLE I: Payoff matrices.

Base game.

| | Betray | Silent |
|---|---|---|
| Betray | 2,2 | 4,1 |
| Silent | 1,4 | 3,3 |

Incentive for left player to betray.

| | Betray | Silent |
|---|---|---|
| Betray | 3,2 | 5,1 |
| Silent | 0,4 | 2,3 |

Incentive for left player to remain silent.

| | Betray | Silent |
|---|---|---|
| Betray | 1,2 | 3,1 |
| Silent | 2,4 | 4,3 |

*Agents*

The game is setup with two agents of heterogeneous capabilities. The uncertain agent, who always plays first in each scenario, must assess his opponent's model over the course of the game and predict his decisions effectively. The second agent only assesses the current scenario, but he also has a hidden additional reward either for betraying the first player or staying silent, depending on the corresponding assigned model. Furthermore, while each model's expected reward distribution results in a pure strategy from a game theoretic standpoint, we instead sample the second agent's action according to the distribution given by Equation 2. The stochastic sampling of actions allows for selection of suboptimal actions for the second agent, which will hinder the first agent's ability to surmise the second's true model.

*PsychSim*

We compare MCTS with an existing social simulation tool, PsychSim. PsychSim employs value iteration to solve for agent policies that maximize expected reward based on its goals [24]. At each time point, an agent, $i$, computes a value, $V_a(b^t)$, of each action $a$, given its beliefs, $b^t$. A transition function projects the effects of actions of other agents, $\pi_{\neg i}$, in future states, and each state is evaluated against the agent's goals, $g$.

$$V_a(b_i^t) = g_i \cdot b_i^t + \sum_{b^{t+1}} V(b^{t+1}) P(b^{t+1} | b_i^{t+1}, a, \pi_{\neg i}(b^{t+1}))$$

The depth at which agents model the game state is bounded by a finite horizon, limiting their lookahead. As value iteration is a full width algorithm, the agents complete an exhaustive analysis of all reachable states within the limited horizon.

*Information Consideration*

It should be noted that MCTS has a distinct information advantage over value iteration by simulating the remainder of a game beyond the most recently expanded node. This benefit comes at the cost, as simulating the entire length of a game requires more time. The time requirements are linear in the number of passes through the tree and length of the game. The rollout process is fast, however, and can be halted at a finite horizon if runtime is a concern. Value iteration, on the other hand, has a complexity linear in the number of actions and quadratic in the number of states, which quickly becomes intractable for application beyond a small game. Furthermore, we show here that even if value iteration could incorporate information in the form of a random action simulation beyond its finite horizon, belief convergence would be unaffected.

**Lemma 1.** *For the game IPD and the Boltzmann distribution of action probabilities, the addition of a random action simulation phase beyond the finite horizon considered in value iteration does not affect the expected Bayesian belief update over models.*

*Proof.* The value of an action can be decomposed into the cumulative, discounted reward accumulated up to the finite horizon and the reward achieved during the rollout phase.

$$E[V_a] = E[V_h] + E[V_r]$$

Under this observation, Equation 2 becomes

$$\begin{aligned} P(a|m) &= \frac{e^{E[V_h(m)]+E[V_r(m)]}}{\sum_{a_i \in A} e^{E[V_{h_i}(m)]+E[V_{r_i}(m)]}} \\ &= \frac{e^{E[V_h(m)]} e^{E[V_r(m)]}}{\sum_{a_i \in A} e^{E[V_{h_i}(m)]} e^{E[V_{r_i}(m)]}} \end{aligned}$$

Furthermore, we observe that for each round of IPD, the agent is in a world state independent of the actions taken up until that point. Additionally, given that during the rollout procedure, actions are sampled uniformly, we know that the expected reward for rollouts of equivalent length (in rounds) is identical. This leads to the following reduction:

$$\begin{aligned} E[V_{r_i}(m)] &= E[V_{r_j}(m)] \text{ where } |r_i| = |r_j| \text{ therefore,} \\ e^{E[V_{r_i}(m)]} &= e^{E[V_{r_j}(m)]} \\ P(a|m) &= \frac{e^{E[V_h(m)]} e^{E[V_r(m)]}}{e^{E[V_r(m)]} \sum_{a_i \in A} e^{E[V_{h_i}(m)]}} \\ P(a|m) &= \frac{e^{E[V_h(m)]}}{\sum_{a_i \in A} e^{E[V_{h_i}(m)]}} \end{aligned}$$

With $P(a|m)$ shown to be unchanged by a random simulation phase in value iteration, it is trivial to show that the updated $P(m|a)$ given by Equation 1 is similarly unaffected.

| | | Horizon (VI) - Nodes Searched (MCTS) | | | | |
|---|---|---|---|---|---|---|
| Model | Approach | $1-4$ | $2-8$ | $3-16$ | $4-32$ | $5-64$ |
| Loyal | MCTS | $13.14(\pm 0.86)$ | $12.87(\pm 0.79)$ | $13.12(\pm 0.63)$ | $13.48(\pm 0.48)$ | $12.62(\pm 0.38)$ |
| Loyal | VI | $13.2(\pm 0.64)$ | $13.19(\pm 0.60)$ | $13.16(\pm 0.65)$ | $13.03(\pm 0.62)$ | $13.07(\pm 0.66)$ |
| Betray | MCTS | $12.95(\pm 0.7)$ | $12.92(\pm 0.82)$ | $12.05(\pm 0.88)$ | $12.98(\pm 0.92)$ | $12.56(\pm 0.39)$ |
| Betray | VI | $12.33(\pm 0.71)$ | $12.18(\pm 0.78)$ | $12.5(\pm 0.79)$ | $12.53(\pm 0.61)$ | $12.49(\pm 0.87)$ |

TABLE II: Cumulative reward (and standard deviation) at the end of the game by opponent model, approach, and states explored.

*Experimental Setup*

In this section, the MCTS implementation with belief updates over models is evaluated alongside a value iteration implementation provided by PsychSim. For comparison, the MCTS variant was restricted to expanding a number of nodes equal to the number of reachable game states within a set finite horizon used by PsychSim. The results of each test were compiled over fifty games.

In our first experiment, we vary the number of expanded nodes used by MCTS before returning a decision. MCTS is a technique which improves with an increasing number of iterations, though at the cost of time. Given the binary decisions–betray or remain silent–at each stage, we restrict the number of possible nodes to powers of 2. These values then correspond to the horizon used by PsychSim, in that 32 nodes would be comparable to value iteration exploring all states up to a horizon of 4. Two scenarios are tested, one for each possible model of the second player.

The second parameter tested in this work is the effect of the exploration factor on the updated belief distributions. Lesser values of the exploration factor diminish its impact in affecting which nodes are explored, resulting in a more greedy search. Similarly, a large exploration factor would emphasize exploration, expanding nodes higher in the game tree, though at the cost of accuracy for longterm play. As before, tests were completed for each of the models of the second player. Since value iteration is a full-width planning algorithm, no corresponding parameter exists, and therefore PsychSim is omitted from these tests.

## VI. Results

End game results for MCTS and value iteration are displayed in Table II. As value iteration finds optimal policies for discounted reward, finite horizon POMDPs, the simulated games with PsychSim provide an approximation of the highest score we can expect any approach to achieve given both the uncertainty of model and stochastic selection of action by the opponent. The results serve simply to illustrate that the MCTS agent achieves a comparable score.

In both tested models, where the second agent either has added incentive to be loyal to or betray the first agent, we observe similar belief convergence patterns. The number of nodes used to search the game's state space has a direct impact on the rate of belief convergence. As increased sampling provides more accurate expected values for actions, the belief update strategy is able to converge more readily. In comparison, as the calculations for PsychSim's value iteration end at a finite horizon, no information regarding the remainder of the game is included. As each iteration of MCTS samples the entire duration of the game, the disparity between resulting scores of the two simulated models is likely larger than that calculated by the shorter horizon of value iteration. This, in turn, is reflected in the estimated probability of an action given a model (Equation 2), which is used to calculate the new belief distribution. We see this effect in this sharp convergence of more highly sampled games (nodes $\geq 16$).

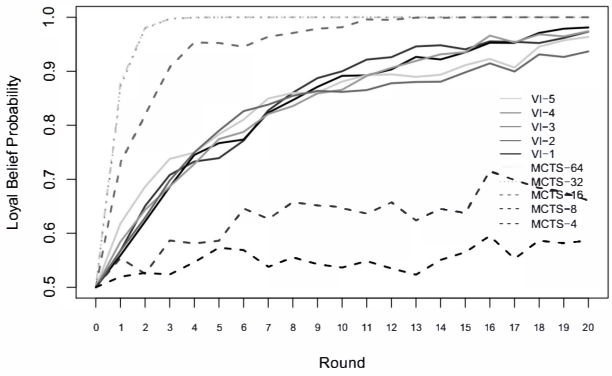


Fig. 2: Average convergence of beliefs when the opponent behaves loyaly. Note: MCTS-64 and MCTS-32 overlap.

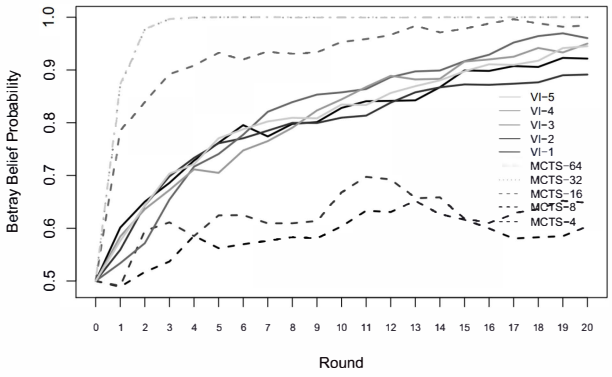


Fig. 3: Average convergence of beliefs when the opponent has incentive to betray. Note: MCTS-64 and MCTS-32 overlap.

When the exploration value is varied, belief convergence is

directly affected. With large exploration factors, more nodes are added earlier in the game tree. This has two main effects. First, MCTS expends more resources examining branches that are irrelevant in the course of the game. Inflating the likelihood of these branches directly affects the resulting values for the agents and, in turn, the probabilities used to update the belief distribution. Second, exploration of nodes earlier in the game inflates the proportion of randomly selected moves for the simulation phase of MCTS. This shift produces more noise in resulting outcomes, again affecting the probabilities used to calculate the belief update. These factors force MCTS to converge more slowly when exploration of actions is emphasized.

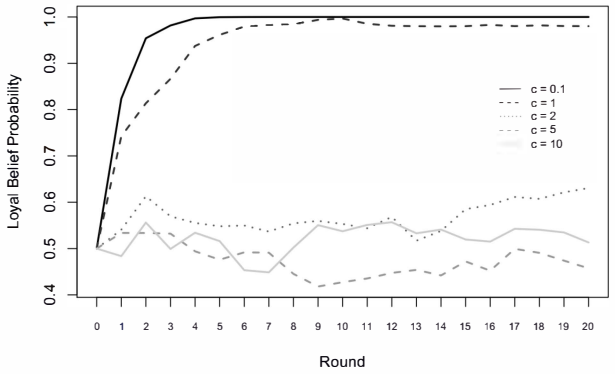


Fig. 4: Average convergence of beliefs with various explore factor values when exposed to loyal behavior.

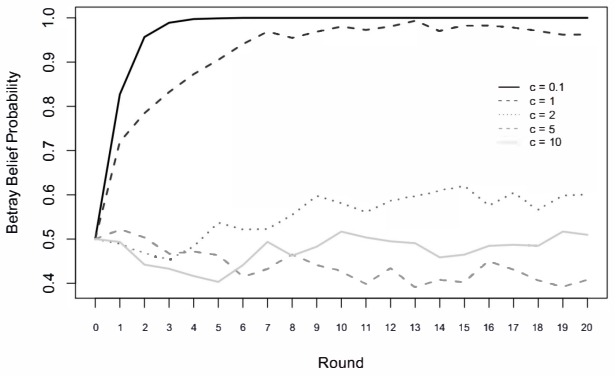


Fig. 5: Average convergence of beliefs with various explore factor values when exposed to betrayal behavior.

## VII. Discussion

It is important to note the potential pitfall of fast convergence to a single model. The caveat in this setup is that once the distribution aligns to a model, the probabilities for opposing models converge to zero (with numeric round-off). Under our Bayesian update scheme, it would be difficult to recover if the belief distribution converged to an incorrect model. However, in addition to tuning the number of nodes searched and the exploration factor, a potential implementation could also adopt a more conservative estimation of the probability of an action given a model, $P(a|m)$. Alternative estimations can be found in [6]. Similarly, one could adjust the update mechanism to incorporate polynomial weights. This process has been shown to be near optimal in [25].

### *Future Work*

Fast identification of a player model is a rich problem full of areas for exploration. We have shown here how adapting information present within MCTS can be leveraged to identify a player's model given his or her actions. But how well can an agent perform in cases where the set of models is incomplete or when the models considered are themselves imperfect descriptors of behavior? As previously noted, game designers and AI researchers alike must take precautions to avoid prematurely converging to an incorrect model given a potentially noisy set of actions or observations.

Additionally, we can consider relaxing various assumptions present in this work. How does this analysis change under a set of models that are not static over time? How do the specifics of a domain relate to choice of formulation for $P(a|m)$?

Furthermore, we envision the eventual necessity to support fully recursive mental modeling, allowing application to domains involving *Theory of Mind*. An agent may someday need to consider not simply what a player might do according to his or her own preferences, but also how the player's beliefs on the agent's behavior affect his or her planning. This line of reason extends to an arbitrary depth of recursive belief modeling. Adding support for this type of reasoning is non-trivial, however, as it significantly increases the complexity of the problem [26] and, therefore, may limit its potential for application in games.

## VIII. Conclusion

This paper adapts Monte-Carlo Tree Search to the task of planning under uncertainty of a player's model, as well as characterizes how the exploration/exploitation parameter and number of simulations affects belief convergence. We demonstrate how information already tracked in MCTS can be used for updating such belief distributions, given a sufficient number of passes. The results show that asymmetric tree growth allows for more informative inference over full-width approaches. Moreover, further tuning of the convergence rate is made possible by the parameters specific to MCTS.

## References

[1] D. Carmel and S. Markovitch, "Pruning algorithms for multi-model adversary search," *Artificial Intelligence*, vol. 99, no. 2, pp. 325–355, 1998.

[2] G. Synnaeve and P. Bessiere, "A bayesian model for plan recognition in rts games applied to starcraft." in *Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2011)*, 2011.

[3] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in *IEEE Symposium on Computational Intelligence and Games*. IEEE, 2009, pp. 140–147.

[4] J. Zhang, "Building opponent model in imperfect information board games," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, no. 3, 2014.

[5] P. Riley and M. Veloso, "Recognizing probabilistic opponent movement models," in *RoboCup 2001: Robot Soccer World Cup V*. Springer, 2002, pp. 453–458.

[6] J. Y. Ito, D. V. Pynadath, and S. C. Marsella, "A decision-theoretic approach to evaluating posterior probabilities of mental models," in *AAAI-07 workshop on plan, activity, and intent recognition*, 2007.

[7] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2008)*, 2008.

[8] S. C. Marsella, D. V. Pynadath, and S. J. Read, "Psychsim: Agent-based modeling of social interactions and influence," in *Proceedings of the international conference on cognitive modeling*. Citeseer, 2004, pp. 243–248.

[9] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[10] K. P. Murphy, "A survey of pomdp solution techniques," UC Berkeley, Tech. Rep., 2000.

[11] M. Ramírez and H. Geffner, "Goal recognition over pomdps: Inferring the intention of a pomdp agent," in *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*. AAAI Press, 2011, pp. 2009–2014.

[12] O. Macindoe, L. P. Kaelbling, and T. Lozano-Pérez, "Pomcop: Belief space planning for sidekicks in cooperative games." in *Proceedings of the Eighth Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2012)*, 2012.

[13] B. Ng, C. Meyers, K. Boakye, and J. Nitao, "Towards applying interactive pomdps to real-world adversary modeling." in *Innovative Applications in Artificial Intelligence (IAAI)*, 2010, pp. 1814–1820.

[14] C. T. Tan and H.-l. Cheng, "An automated model-based adaptive architecture in modern games." in *Proceedings of the Sixth Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE 2010)*, 2010.

[15] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*. Springer, 2006, pp. 282–293.

[16] D. Silver and J. Veness, "Monte-carlo planning in large pomdps," in *Advances in Neural Information Processing Systems*, 2010, pp. 2164–2172.

[17] H. Yoshimoto, K. Yoshizoe, T. Kaneko, A. Kishimoto, and K. Taura, "Monte carlo go has a way to go," in *AAAI*, vol. 6, 2006, pp. 1070–1075.

[18] H. Finnsson and Y. Björnsson, "Simulation-based approach to general game playing." in *AAAI*, vol. 8, 2008, pp. 259–264.

[19] M. J. Ponsen, G. Gerritsen, and G. Chaslot, "Integrating opponent models with monte-carlo tree search in poker." in *Interactive Decision Theory and Game Theory*, 2010.

[20] D. Robles and S. M. Lucas, "A simple tree search method for playing ms. pac-man," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 249–255.

[21] M. Enzenberger, M. Muller, B. Arneson, and R. Segal, "Fuegoan open-source framework for board games and go engine based on monte carlo tree search," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 4, pp. 259–270, 2010.

[22] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 2, pp. 120–143, 2012.

[23] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[24] D. V. Pynadath and S. C. Marsella, "Psychsim: Modeling theory of mind with decision-theoretic agents," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, vol. 5, 2005, pp. 1181–1186.

[25] A. Blum and Y. Monsour, "Learning, regret minimization, and equilibria," in *Algorithmic Game Theory*. Cambridge University Press, 2007, ch. 4.

[26] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.