Deep Synoptic Monte Carlo Planning in Reconnaissance Blind Chess

Gregory Clark
ML Collective, Google
gregoryclark@google.com

Abstract

This paper introduces deep synoptic Monte Carlo planning (DSMCP) for large imperfect information games. The algorithm constructs a belief state with an unweighted particle filter and plans via playouts that start at samples drawn from the belief state. The algorithm accounts for uncertainty by performing inference on “synopses,” a novel stochastic abstraction of information states. DSMCP is the basis of the program Penumbra, which won the official 2020 reconnaissance blind chess competition versus 33 other programs. This paper also evaluates algorithm variants that incorporate caution, paranoia, and a novel bandit algorithm. Furthermore, it audits the synopsis features used in Penumbra with per-bit saliency statistics.

1 Introduction

Choosing a Nash equilibrium strategy is rational when the opponent is able to identify and exploit suboptimal behavior [Bowling and Veloso, 2001]. However, not all opponents are so responsive, and computing a Nash equilibrium is intractable for many games. This paper presents deep synoptic Monte Carlo planning (DSMCP), an algorithm for large imperfect information games that seeks a best-response strategy rather than a Nash equilibrium strategy.

When opponents use fixed policies, an imperfect information game may be viewed as a partially observable Markov decision process (POMDP) with the opponents as part of the environment. DSMCP treats playing against specific opponents as related offline reinforcement learning (RL) problems and exploits predictability. Importantly, the structure of having opponents with imperfect information is preserved in order to account for their uncertainty.

DSMCP uses sampling to break the “curse of dimensionality” [Pineau et al., 2006] in three ways: sampling possible histories with a particle filter, sampling possible futures with upper confidence bound tree search (UCT) [Kocsis and Szepesvári, 2006], and sampling possible world states within each information state uniformly. It represents information states with a generally-applicable stochastic abstraction technique that produces a “synopsis” from sampled world states. This paper assesses DSMCP on reconnaissance blind chess (RBC), a large imperfect information chess variant.

2 Background

Significant progress has been made in recent years in both perfect and imperfect information settings. For example, using deep neural networks to guide UCT has enabled monumental achievements in abstract strategy games as well as computer games [Silver et al., 2016, 2017a,b, Schrittwieser et al., 2019, Wu, 2020, Tomašev et al., 2020]. This work employs deep learning in a similar fashion.

Recent advancements in imperfect information games are also remarkable. Several programs have reached superhuman performance in Poker [Moravčík et al., 2017, Brown and Sandholm, 2018, 2019, Brown et al., 2020]. In particular, ReBeL [Brown et al., 2020] combines RL and search by converting
Within 98k historical reconnaissance blind chess (RBC) games between non-random players, the win percentage tends to decrease as the number of possible states increases. The games were replayed while tracking up to one million states. Each bucket is labeled with an inclusive upper bound. The median of the maximum number of possible states encountered during a game is 4,869.

![Figure 1](link)

Playing RBC well requires balancing risks and rewards. (a) On the left, Penumbra moved the white queen to g8. After sensing at d2, Black could infer that the white queen occupied one of 25 squares. That uncertainty allowed the white queen to survive and capture the black king on the next turn. (b) On the right, Penumbra moved the black queen to h2. In this case, the opponent detected and captured the black queen.

The games are available online at [https://rbc.jhuapl.edu/games/120174](https://rbc.jhuapl.edu/games/120174) and [https://rbc.jhuapl.edu/games/124718](https://rbc.jhuapl.edu/games/124718).

Neural fictitious self-play (NFSP) [Heinrich and Silver, 2016] is an RL algorithm for training two neural networks for imperfect information games. Experiments with NFSP employed compact observations embeddings of information states. DSMCP offers a generic technique for embedding information states in large games. Dual sequential Monte Carlo (DualSMC) [Wang et al., 2019] estimates belief states and plans in a continuous domain via sequential Monte Carlo with heuristics.
3 Reconnaissance blind chess

Reconnaissance blind chess (RBC) is a chess variant that incorporates uncertainty about the placement of the opposing pieces along with a private sensing mechanism. As shown in Figure 1 RBC players are often faced with thousands of possible game states, and reducing uncertainty increases the odds of winning.

Game rules Pieces move in the same way in RBC as in chess. Players cannot directly observe the movement of the opposing pieces. However, at the beginning of each turn, players may view the ground truth of any 3×3 patch of the board. The information gained from the sensing action remains private to that player. Players are also informed of the location of all captures, but not the identity of capturing pieces. When a requested move is illegal, the move is substituted with the closest legal move and the player is notified of the substitution. For example, in Figure 2(a), if Black attempted to move the rook from h8 to f8, the rook would capture the queen on g8 and stop there instead. Players are always able to track the placement of their own pieces. Capturing the opposing king wins the game, and players are not notified about check. Passing and moving into check are legal.

Official competition This paper introduces the program Penumbra, the winner of the official 2020 RBC rating competition. In total, 34 programs competed to achieve the highest rating by playing public games. Ratings were computed with BayesElo, and playing at least 100 games was required to be eligible to win. Figure 2 shows ground truth positions from the tournament in which Penumbra voluntarily put its queen in danger. Players were paired randomly, but the opponent’s identity was provided at the start of each game which allowed catering strategies for specific opponents. However, opponents were free to change their strategies at any point, so attempting to exploit others could backfire. Nonetheless, Penumbra sought to model and counter predictable opponents rather than focusing on finding a Nash equilibrium.

Other RBC programs RBC programs have employed a variety of algorithms including Q-learning, counterfactual regret minimization (CFR), online outcome sampling (OOS), and the Stockfish chess engine. Another strong RBC program maintained a probability distribution for each piece. Most RBC programs select sense actions and move actions in separate ways while DSMCP unifies all action selection. also applied UCT to RBC and modeled the root belief state with a distribution over positions. Input to a neural network consisted of the most-likely 100 positions, and storing a single training example required 3.5MB on average which was large enough to hinder training. This work represents training examples with compact synopses which are less than 1kB without compression.

4 Terminology

Consider the two-player extensive-form game with agents \( P = \{ \text{self, opponent} \} \), actions \( A \), “ground truth” world states \( X \), and initial state \( x_0 \in X \). Each time an action is taken, each agent \( p \in P \) is given an observation \( o_p \in O \) that matches (\( \sim \)) the possible world states from \( p \)’s perspective. For simplicity, assume the game has deterministic actions such that each \( a \in A \) is a function \( a : X \to \mathcal{X} \) defined on a subset of world states \( X \subseteq \mathcal{X} \). Define \( A_x \) as the set of actions available from \( x \in \mathcal{X} \).

An information state (infostate) \( s \in S \) for agent \( p \) consists of all observations \( p \) has received so far. Let \( \mathcal{X}_s \subseteq \mathcal{X} \) be the set of all world states that are possible from \( p \)’s perspective from \( s \). In general, \( \mathcal{X}_s \) contains less information than \( s \) since some (sensing) actions may not affect the world state. Define a collection of limited-size world state sets \( \mathcal{L} = \{ L \subseteq \mathcal{X}_s : s \in S, |L| \leq \ell \} \), given a constant \( \ell \).

Let \( \rho : \mathcal{X} \to P \) indicate the agent to act in each world state. Assume that \( A_x = A_y \) and \( \rho(x) = \rho(y) \) for all \( x, y \in \mathcal{X}_s \) and \( s \in S \). Then extend the definitions of actions available \( A_x \) and agent to act \( \rho \) over sets of world states and over infostates in the natural way. A policy \( \pi(a|s) \) is a distribution over actions given an infostate. A belief state \( B(h) \) is a distribution over action histories. Creating a belief state from an infostate requires assumptions about the opponent’s action policy \( \tau(a|s) \). Let

---

\( 1 \) An infostate is equivalent to an information set, which is the set of all possible action histories from \( p \)’s perspective [Osborne and Rubinstein 1994].
Effective planning algorithms for imperfect information games must model agents’ choice of actions based on (belief states derived from) infostates, not on world states themselves. Deep synoptic Monte Carlo planning (DSMCP) approximates infostates with size-limited sets of possible world states in $\mathcal{L}$. It uses those approximations to construct a belief state and as UCT nodes [Kocsis and Szepesvari 2006]. Figure 3 provides a high-level visualization of the algorithm.

A bandit algorithm chooses an action during each node visit, as described in Algorithm 1. This bandit algorithm is similar to Smooth UCB [Heinrich and Silver 2015] in that they both introduce stochasticity by mixing in a secondary policy. Smooth UCB empirically approached a Nash equilibrium utilizing the average policy according to action visits at each node. DSMCP mixes in a neural network’s policy ($\pi$) instead. The constant $c$ controls the level of exploration, and $m$ controls how the policy $\pi$ is mixed into the bandit algorithm. For example, taking $m = 0$ always selects actions directly with $\pi$ without considering visit counts, and taking $m = \infty$ never mixes in $\pi$. As in [Silver et al. 2016], $\pi$ provides per-action exploration values which guide the tree search.

Approximate belief states are constructed as subsets $\hat{B} \subset \mathcal{L}$, where each $L \in \hat{B}$ is a set of possible world-states from the opponent’s perspective. This “second order” representation of belief states allows DSMCP to account for the opponent’s uncertainty. Infostates sampled with rejection (Algorithm 2) are used as the “particles” in a particle filter which models successive belief states. Sampling is guided by a neural network policy ($\hat{\pi}$) based on the identity of the opponent. To counter particle deprivation, if $k$ consecutive candidate samples are rejected as incompatible with the possible world states, then a singleton sample consisting of a randomly-chosen possible state is selected instead.
Algorithm 3 ChooseAction – UCT playouts

Given: $b, d, \ell, n_{v1}, z \in \mathbb{Z}^+$, synopsis func. $\sigma$, policy $\pi$, opp. policy $\bar{\pi}$, value func. $\nu$

Input: belief distribution $\hat{B}$, possible states $X$, visit counts $\mathbf{N}$, value totals $\mathbf{Q}$

$\hat{\pi} \leftarrow$ average of $\pi \circ \sigma$ over $b$ samples from $\hat{B}$

while time is left do
    $J \leftarrow$ random sample from $\hat{B}$
    $I \leftarrow$ random $\ell$ states in $X$ including one in $J$
    $a_0 \leftarrow$ Bandit($\hat{\pi}, \mathbf{N}_\text{root}, \mathbf{Q}_\text{root}$)
    for $t = 0 \to d$ do
        $x_t \leftarrow$ the one state in $I \cap J$
        $o \leftarrow$ observations when $a_t$ is played on $x_t$
        $I \leftarrow \{a(x) : x \in I, a \in \mathcal{A}_J, o_{self} \sim a(x)\}$
        $J \leftarrow \{a(x) : x \in J, a \in \mathcal{A}_J, o_{opp} \sim a(x)\}$
        $I \leftarrow$ random $\ell$ states in $I$ including $a_t(x_t)$
        $J \leftarrow$ random $\ell$ states in $J$ including $a_t(x_t)$
        $K_t \leftarrow I$ and $\mu \leftarrow \pi$ if $I$ is to act
        $K_t \leftarrow J$ and $\mu \leftarrow \bar{\pi}$ if $J$ is to act
        $d += 1$ if $\mathbf{N}_{K_t,a_t} > z$
    for $i = 0 \to n_{v1}$
        $a_{i+1} \leftarrow$ Bandit($\mu(\sigma(K_t)), \mathbf{N}_{K_t}, \mathbf{Q}_{K_t}$)
        $q \leftarrow \nu(\sigma(K_t))$
    for $i = 0 \to n_{v1}$
        $\mathbf{Q}_{K_t,a_t} += q$ if $\rho(K_t) = \rho(K_t)$ else $-q$
        $\mathbf{N}_{K_t,a_t} += 1 - n_{v1}$

return $a \leftarrow \arg\max_a \mathbf{Q}_\text{root}$

Algorithm 4 PlayGame – DSMCP

Given: $n_{\text{particles}} \in \mathbb{Z}^+$

$\mathbf{N}_{x,a} \leftarrow 0$ // Visits $(\text{sample, action}) \in \mathcal{L} \times \mathcal{A}$

$\mathbf{Q}_{x,a} \leftarrow 0$ // Values $(\text{sample, action}) \in \mathcal{L} \times \mathcal{A}$

$X_0 \leftarrow \{x_0\}$

$\hat{B}_0 \leftarrow \{X_0\}$

while the game is not over do
    $t \leftarrow$ current turn
    $o \leftarrow$ current observation
    // Track all possible world states
    $X_t \leftarrow \{a(x) : x \in X_{t-1}, a \in \mathcal{A}_x, o_{self} \sim a(x)\}$
    // Filter belief states with the new information
    for $i = t-1 \to 0$ do
        $X_i \leftarrow \{x \in X_i : \exists a \in \mathcal{A}_x \text{ such that } a(x) \in X_{i+1}\}$
    // Repopulate belief states with new particles
    while opponent to act or $|\hat{B}_i| < n_{\text{particles}}$ do
        $i \leftarrow$ smallest $i > 0$ s.t. $|\hat{B}_i| < n_{\text{particles}}$
        $I \leftarrow \text{DrawSample}(\hat{B}_{i-1}, X_i, \mathbf{N}, \mathbf{Q})$
        $\hat{B}_i \leftarrow \hat{B}_i \cup \{I\}$
    if self to act then
        $a \leftarrow \text{ChooseAction}(\hat{B}_i, X_i, \mathbf{N}, \mathbf{Q})$
        Perform action $a$

The tree search, described in Algorithm 3, tracks an approximate infostate for each player while simulating playouts. Playouts are also guided by policy ($\pi$ and $\bar{\pi}$) and value ($\nu$) estimations from a neural network. A synopsis function $\sigma$ creates a fixed-size summary of each node as input for the network. The constant $b$ is the batch size for inference, $d$ is the search depth, $\ell$ is the size of approximate infostates, $n_{v1}$ is the virtual loss weight, and $z$ is a threshold for increasing search depth.

Algorithm 4 describes how to play an entire game, tracking all possible world states. Approximate belief states ($\hat{B}_i$) are constructed for each past turn by tracking $n_{\text{particles}}$ elements of $\mathcal{L}$ (from the opponent’s point of view) with an unweighted particle filter. Each time the agent receives a new observation, all of the (past) particles that are inconsistent with the observation are filtered out and replenished, starting with the oldest belief states.

5.1 Synopsis

One of the contributions of this paper is the methodology used to approximate and encode infostates. Games that consist of a fixed number of turns, such as poker, admit a naturally-compact infostate representation based on observations [Heimrich and Silver 2015, 2016]. However, perfect representations are not always practical. Game abstractions are often used to reduce computation and memory requirements. For example, imperfect recall is an effective abstraction when past actions are unnecessary for understanding the present situation [Waugh et al. 2009, Lanctot et al. 2012].

DSMCP employs a stochastic abstraction which represents infostates with sets of world states and then subsamples to a manageable cardinality ($\ell$). Finally, a permutation-invariant synopsis function $\sigma$ produces fixed-size summaries of the approximate infostates which are used for inference. An alternative is to run inference on “determinized” world states individually and then somehow aggregate the results. However, such aggregation can easily lead to strategy fusion [Frank et al. 1998]. Other alternatives include evaluating states with a recurrent network [Rumelhart et al. 1986, one-at-a-time or using a permutation-invariant architecture [Zaheer et al. 2017, Wagstaff et al. 2019].
Given functions $g_i : X \rightarrow \{0, 1\}$ for $i = 0, \ldots, F$ that map states to binary features, define the $i^{th}$ component of a synopsis function $\sigma : L \rightarrow \{0, 1\}^F$ as

$$\sigma_i(X) = \bigwedge_{j=0}^{F} g_i(x_j) * i \bigotimes_{j \neq i} g_i(x_j)$$

where $X = \{x_0, x_1, \ldots, x_F\}$ and $\ast_i$ is either the logical AND ($\land$) or the logical OR ($\lor$) operation. For example, if $g_i$ encodes whether an opposing knight can move to the d7 square of a chess board and $\ast_i = \land$, then $\sigma_i$ indicates that a knight can definitely move to d7. Figure 4 shows an example game, and Figure 5 shows an example output of Penumbra’s synopsis function, which consists of 104 bitboard feature planes each with 64 binary features. The appendix describes each feature plane.

### 5.2 Network architecture

Penumbra uses a residual neural network [He et al. 2016] as shown in Figure 6. The network contains 14 headsets, designed to model specific opponents and regularize each other as they are trained on different slices of data [Zhang et al. 2020]. Each headset contains 5 heads: a policy head, a value head, two heads for predicting winning and losing within the next 5 actions, and a head for guessing the number of pieces of each type in the ground truth world state. The Top policy head and the All value head are used for planning as $\pi$ and $\nu$, respectively. The other heads (including the SoonWin, SoonLose, and PieceCount heads) provide auxiliary tasks for further regularization [Wu 2020, Fifty et al. 2020]. While playing against an opponent that is “recognized” (when a headset was trained on data from only that opponent), the policy head ($\pi$) of the corresponding headset is used for
Figure 6: Penumbra’s network contains a shared tower with 10 residual blocks and 14 headsets. Each headset contains 5 heads for a total of 70 output heads. The residual blocks are shown with a double border, and they each contain two $3 \times 3$ convolutional layers and batch normalization. All of the convolutional layers in the headsets are $1 \times 1$ convolutions with the exception of the one residual block for each policy head. Each headset was trained on a separate subset of the data, as described in Table 1. The policy head provides logits for both sense and move actions.

the opponent’s moves while progressing the particle filter (Algorithm 2) and while constructing the UCT tree (Algorithm 3). When the opponent is unrecognized, the Top policy head is used by default.

5.3 Training procedure

The network was trained on historical game data as described by Table 1. The reported accuracies are averages over 5 training runs. The All headset was trained on all games, the Top headset was trained on games from the highest-rated players, the Human headset was trained on all games played by humans, and each of the other 11 headsets were trained to mimic specific opponents.

10% of the games were used as validation data based on game filename hashes. Training examples were extracted from games multiple times since reducing possible state sets to $\ell$ states is non-deterministic. A single step of vanilla stochastic gradient descent was applied to one headset at a time, alternating between headsets according to their training weights. See the appendix for hyperparameter settings and accuracy cross tables. Training and evaluation were run on four RTX 2080 Ti GPUs.

5.4 Implementation details

Penumbra plays RBC with DSMCP along with RBC-specific extensions. First, sense actions that are dominated by other sense actions are pruned from consideration. Second, Penumbra can detect some forced wins in the sense phase, the move phase, and during the opponent’s turn. This static analysis is applied at the root and to playouts; playouts are terminated as soon as a player could win, avoiding unnecessary neural network inference. The static analysis was also used to clean training games in which the losing player had sufficient information to find a forced win.

Piece placements are represented with bitboards [Browne, 2014], and the tree of approximate infostates is implemented with a hash table. Zobrist hashing [Zobrist, 1990] maintains hashes of piece placements incrementally. Hash table collisions are resolved by overwriting older entries. The tree was not implemented until after the competition, so fixed-depth playouts were used instead ($m = 0$). Inference is done in batches of 256 during both training and online planning. The time used per

---

[1] The games were downloaded from rbmc.jhuapl.edu in June, 2019 and rbc.jhuapl.edu in August, 2020. Additionally, 5,000 games were played locally by StockyInference.
Table 1: Summary of headset training data and resulting validation accuracies

<table>
<thead>
<tr>
<th>Name</th>
<th># Games</th>
<th># Train</th>
<th># Actions</th>
<th># Validation</th>
<th>Actions</th>
<th>Multiplicity</th>
<th>Training Weight</th>
<th>Stop Gradient</th>
<th>% Top-1 Accuracy</th>
<th>% Top-5 Accuracy</th>
<th>% Winner Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>85.6k</td>
<td>16.8M</td>
<td>1.8M</td>
<td>4</td>
<td>15</td>
<td>No</td>
<td></td>
<td></td>
<td>33.6</td>
<td>62.0</td>
<td>74.8</td>
</tr>
<tr>
<td>Top</td>
<td>49.5k</td>
<td>16.0M</td>
<td>1.6M</td>
<td>∼9.6</td>
<td>13</td>
<td>No</td>
<td></td>
<td></td>
<td>43.1</td>
<td>73.4</td>
<td>82.4</td>
</tr>
<tr>
<td>StrangeFish</td>
<td>13.3k</td>
<td>8.1M</td>
<td>0.8M</td>
<td>∼20.7</td>
<td>10</td>
<td>No</td>
<td></td>
<td></td>
<td>45.9</td>
<td>74.3</td>
<td>86.6</td>
</tr>
<tr>
<td>LaSalle</td>
<td>3.7k</td>
<td>4.3M</td>
<td>0.4M</td>
<td>32</td>
<td>4</td>
<td>No</td>
<td></td>
<td></td>
<td>36.4</td>
<td>68.2</td>
<td>69.7</td>
</tr>
<tr>
<td>Dyn. Entropy</td>
<td>7.8k</td>
<td>7.8M</td>
<td>0.8M</td>
<td>32</td>
<td>7</td>
<td>No</td>
<td></td>
<td></td>
<td>50.0</td>
<td>76.7</td>
<td>78.9</td>
</tr>
<tr>
<td>Oracle</td>
<td>20.3k</td>
<td>17.6M</td>
<td>1.9M</td>
<td>32</td>
<td>9</td>
<td>No</td>
<td></td>
<td></td>
<td>49.3</td>
<td>77.4</td>
<td>81.4</td>
</tr>
<tr>
<td>StockyInfer.</td>
<td>10.7k</td>
<td>13.5M</td>
<td>1.4M</td>
<td>16</td>
<td>9</td>
<td>No</td>
<td></td>
<td></td>
<td>45.2</td>
<td>73.6</td>
<td>68.3</td>
</tr>
<tr>
<td>Marmot</td>
<td>10.2k</td>
<td>3.9M</td>
<td>0.4M</td>
<td>16</td>
<td>8</td>
<td>No</td>
<td></td>
<td></td>
<td>24.6</td>
<td>55.4</td>
<td>80.8</td>
</tr>
<tr>
<td>Genetic</td>
<td>5.6k</td>
<td>2.3M</td>
<td>0.2M</td>
<td>16</td>
<td>8</td>
<td>No</td>
<td></td>
<td></td>
<td>40.4</td>
<td>70.5</td>
<td>77.9</td>
</tr>
<tr>
<td>Zugzwang</td>
<td>10.9k</td>
<td>3.1M</td>
<td>0.3M</td>
<td>∼12.0</td>
<td>5</td>
<td>No</td>
<td></td>
<td></td>
<td>47.0</td>
<td>71.6</td>
<td>80.1</td>
</tr>
<tr>
<td>Trout</td>
<td>18.0k</td>
<td>5.1M</td>
<td>0.5M</td>
<td>16</td>
<td>5</td>
<td>No</td>
<td></td>
<td></td>
<td>41.8</td>
<td>63.4</td>
<td>79.8</td>
</tr>
<tr>
<td>Human</td>
<td>6.3k</td>
<td>1.5M</td>
<td>0.2M</td>
<td>16</td>
<td>2</td>
<td>No</td>
<td></td>
<td></td>
<td>24.9</td>
<td>51.9</td>
<td>73.3</td>
</tr>
<tr>
<td>Attacker</td>
<td>15.3k</td>
<td>1.7M</td>
<td>0.2M</td>
<td>16</td>
<td>4</td>
<td>No</td>
<td></td>
<td></td>
<td>45.1</td>
<td>61.9</td>
<td>80.1</td>
</tr>
<tr>
<td>Random</td>
<td>17.0k</td>
<td>0.6M</td>
<td>76k</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
<td></td>
<td></td>
<td>4.5</td>
<td>20.7</td>
<td>94.7</td>
</tr>
</tbody>
</table>

The action is approximately proportional to the time remaining. The program processes approximately 4,000 nodes per second, and it plays randomly when the number of possible states exceeds 9 million.

6 Experiments

This section presents the results of playing games between Penumbra and several publicly available RBC baselines [Gardner et al., 2020, Bernardoni, 2020]. Each variant of Penumbra in Table 2 played 1000 games against each baseline, and each variant in Table 3 and Table 4 played 250 games against each baseline. Games with errors were ignored and replayed. The Elo ratings and 95% confidence intervals were computed with BayesElo [Coulom, 2008] and are all compatible. The scale was anchored with StockyInference at 1500 based on its rating during the competition.

Table 2 gives ratings of the baselines and five versions of Penumbra. PenumbraCache relied solely on the network policy for action selection in playouts (m = 0), PenumbraTree built a UCT search tree (m = ∞), and PenumbraMixture mixed in the network policy during early node visits (m = 1). The mixed strategy performed the best. PenumbraNetwork selected actions based on the network policy without performing any playouts. PenumbraSimple is the same as PenumbraMixture with the static analysis described in Section 5.4 disabled. PenumbraNetwork and PenumbraSimple serve as ablation studies; removing the search algorithm is detrimental while the effect of removing the static analysis is not statistically significant. Unexpectedly, Penumbra played the strongest against StockyInference when that program was unrecognized. So, in this case, modeling the opponent with a stronger policy outperformed modeling it more accurately.

Two algorithmic modifications that give the opponent an artificial advantage during planning were investigated. Table 3 reports the results of a grid search over “cautious” and “paranoid” variants of DSMCP. The caution parameter κ specifies the percentage of playouts that use ℓ = 4 for the opponent instead of the higher default limit. Since each approximate infostate is guaranteed to contain the correct ground truth in playouts, reducing ℓ for the opponent gives the opponent higher-quality information, allowing the opponent to counter risky play more easily in the constructed UCT tree.

The paranoia parameter augments the exploration values in Algorithm 1 to incorporate the minimum value observed for each action. With paranoia φ, actions are selected according to

$$\arg\max_a \left(1 - \phi \frac{\hat{q}_a}{\hat{n}_a} + \phi \hat{m}_a + c\pi_a \sqrt{\frac{\ln n}{\hat{n}_a}} \right)$$

where $\hat{m}_a$ contains the minimum value observed for each action. This is akin to the notion of paranoia studied by Parker et al. [2006, 2010].
Table 2: Bot Elo scores

<table>
<thead>
<tr>
<th>Bot</th>
<th>Recognized as</th>
<th>Elo score</th>
</tr>
</thead>
<tbody>
<tr>
<td>PenumbraMixture</td>
<td>unrecognized</td>
<td>1747±11</td>
</tr>
<tr>
<td>PenumbraSimple</td>
<td>unrecognized</td>
<td>1739±10</td>
</tr>
<tr>
<td>PenumbraCache</td>
<td>unrecognized</td>
<td>1727±10</td>
</tr>
<tr>
<td>PenumbraTree</td>
<td>unrecognized</td>
<td>1641±9</td>
</tr>
<tr>
<td>StockyInference</td>
<td>Trout</td>
<td>1610±7</td>
</tr>
<tr>
<td>StockyInference</td>
<td>StrangeFi.</td>
<td>1528±8</td>
</tr>
<tr>
<td>StockyInference</td>
<td>Genetic</td>
<td>1512±8</td>
</tr>
<tr>
<td>StockyInference</td>
<td>StockyInf.</td>
<td>1500±8</td>
</tr>
<tr>
<td>StockyInference</td>
<td>unrecognized</td>
<td>1474±8</td>
</tr>
<tr>
<td>PenumbraNetwork</td>
<td>unrecognized</td>
<td>1376±9</td>
</tr>
<tr>
<td>AggressiveTree</td>
<td>unrecognized</td>
<td>1134±15</td>
</tr>
<tr>
<td>FullMonte</td>
<td>Trout</td>
<td>1005±22</td>
</tr>
<tr>
<td>Trout</td>
<td>unrecognized</td>
<td>997±22</td>
</tr>
</tbody>
</table>

Table 3: Caution and paranoia grid search results

<table>
<thead>
<tr>
<th>Caution</th>
<th>Paranoia</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>1691±18</td>
</tr>
<tr>
<td>10%</td>
<td>1688±18</td>
</tr>
<tr>
<td>20%</td>
<td>1670±18</td>
</tr>
<tr>
<td>30%</td>
<td>1666±18</td>
</tr>
</tbody>
</table>

Table 4: Exploration strategy grid search results

<table>
<thead>
<tr>
<th>Exploration ratio c</th>
<th>UCB1</th>
<th>aVoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1698±19</td>
<td>1696±18</td>
</tr>
<tr>
<td>2</td>
<td>1686±18</td>
<td>1680±18</td>
</tr>
<tr>
<td>4</td>
<td>1695±18</td>
<td>1695±18</td>
</tr>
</tbody>
</table>

Figure 7: The mean historical win percentage and the mean network value assigned to (a) train and (b) validation synopses tend to decrease as the number of world states given to σ increases.

7 Per-bit saliency

Saliency methods may be able to identify which of the synopsis feature planes are most important and which are least important. Gradients only provide local information, and some saliency methods fail basic sanity checks [Adebayo et al., 2018]. Higher quality saliency information may be surfaced by integrating gradients over gradually-varied inputs [Sundararajan et al., 2017] [Kapishnikov et al., 2019] and by smoothing gradients locally [Smilkov et al., 2017]. Those saliency methods are not directly applicable to discrete inputs such as the synopses used in this work. So, this paper introduces a saliency method that aggregates gradient information across two separate dimensions: training examples and iterations. Per-batch saliency (PBS) averages the absolute value of gradients over random batches of test examples throughout training. Similarly, per-bit saliency (PbS) averages the absolute value of gradients over bits (with specific values) within batches of test examples throughout training. Gradients were taken both with respect to the loss and with respect to the action policy.

Figure 8 shows saliency information for input synopsis features used by Penumbra. In order to validate that these saliency statistics are meaningful, the model was retrained 104 times, once with each feature removed [Hooker et al., 2018]. Higher saliency is slightly correlated with decreased performance when a feature is removed. The correlation coefficient to the average change in accuracy is −0.208 for loss-PBS, and −0.206 for action-PbS. Explanations for the low correlation include noise in the training process and the presence of closely-related features. Ultimately, the contribution of a feature during training is distinct from how well the model can do without that feature. Since
Figure 8: (a) The loss per-batch saliency (PBS) and (b) the action per-bit saliency (PbS) are taken on test examples during training. These graphs show the saliency of feature plane #8, dark squares, for each headset in one training run. The large gradients with respect to the loss suggest that the Genetic headset has overfit. (c) The loss PBS and (d) the action PbS provide insight about which synopsis features are most useful. The top-five most-salient feature planes and the least-salient feature plane for the Top headset from one training run are shown.

some features are near-duplicates of others, removing one may simply increase dependence on another. Still, features with high saliency — such as the current stage (sense or move) and the location of the last capture — are likely to be the most important, and features with low saliency may be considered for removal. The appendix includes saliency statistics for each feature plane.

8 Discussion

Broader impact DSMCP is more broadly applicable than some prior algorithms for imperfect information games, which are intractable in settings with large infostates and small amounts of shared knowledge [Brown et al., 2020]. RBC and the related game Kriegspiel were motivated by uncertainty in warfare [Newman et al., 2016]. While playing board games is not dangerous in itself, algorithms that account for uncertainty may become effective and consequential in the real world. In particular, since it focuses on exploiting weaknesses of other agents, DSMCP could be applied in harmful ways.

Future work Planning in imperfect information games is an active area of research [Russell and Norvig, 2020], and RBC is a promising testing ground for such research. Penumbra would likely benefit from further hyperparameter tuning and potentially alternative corralled bandit algorithms [Arora et al., 2020]. Modeling an opponent poorly could be catastrophic; algorithmic adjustments may lead to more-robust best-response strategies [Ponsen et al., 2011]. How much is lost by collapsing infostates with synopses is unclear and deserves further investigation. Finally, the “bitter lesson” of machine learning [Sutton, 2019] suggests that a learned synopsis function may perform better.

Acknowledgements

Thanks to the Johns Hopkins University Applied Physics Laboratory for inventing such an intriguing game and for hosting RBC competitions. Thanks to Ryan Gardner for valuable correspondence. Thanks to Rosanne Liu, Joel Veness, Marc Lanctot, Zhe Zhao, and Zach Nussbaum for providing feedback on early drafts. Thanks to William Bernardoni for open sourcing high-quality baseline bots. Thanks to Solidmind for the song “Penumbra”, which is an excellent soundtrack for programming.
References


Viliam Lisy, Marc Lanctot, and Michael Bowling. Online monte carlo counterfactual regret minimization for search in imperfect information games. volume 1, 05 2015.


Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael Bowling. A practical use of imperfect recall. 01 2009.


