Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

POMCP with Human Preferences in Settlers of Catan

Mihai S. Dobre, Alex Lascarides
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB
Scotland
Email: m.s.dobre@sms.ed.ac.uk, alex@inf.ed.ac.uk

Abstract

We present a suite of techniques for extending the Partially Observable Monte Carlo Planning algorithm to handle complex multi-agent games. We design the planning algorithm to exploit the inherent structure of the game. When game rules naturally cluster the actions into sets called *types*, these can be leveraged to extract characteristics and high-level strategies from a sparse corpus of human play. Another key insight is to account for action legality both when extracting policies from game play and when these are used to inform the forward sampling method. We evaluate our algorithm against other baselines and versus ablated versions of itself in the well-known board game Settlers of Catan.

Introduction

This paper is concerned with learning to play highly complex games, where exact methods aren't feasible and online planning methods perform better. Even in this case, it is well known that time forbids sufficient search of the space, and that random sampling is not informative enough. There are a multitude of methods to extract policies from existing gameplay which can then be used to inform the sampling method: for instance, inverse reinforcement learning (Ng and Russell 2000), apprenticeship learning (Abbeel and Ng 2004), learning by demonstration (Argall and others 2009), learning to predict expert play via supervised learning (Maddison and others 2015) and reinforcement learning bootstrapped with policies extracted via supervised learning (Silver and others 2016). However, these assume that previous play is generated by experts, and/or have access to massive data sets or computational budget. Our focus is to extract policies from low quantities of mixed previous play in a very challenging environment: Settlers of Catan. Due to its challenges, previous work relied on complicated hand-crafted heuristics (Thomas 2004; Pfeiffer 2003) or focused only on portions of this environment (Szita, Chaslot, and Spronck 2010; Keizer and others 2017; Dobre and Lascarides 2017b).

In this paper, we present the first purely machine learning agent that can address the complete set of game rules, including trading and partial observability. The success of our model comes from two features. First, we exploit the game rules to impose structure on the learning task, enabling the agent to find lucrative parts of the game tree in the time available. Second, we mine a corpus of human game play to estimate which parts of the game tree to explore during online learning. In a major contrast from prior work, we show that learning from human data is beneficial even if the amount and quality of the data available is very low—in our case, we had only 60 games rather than millions. Nevertheless, a set of high-level preferences that are extracted from this data provide useful information in significantly improving the planning agent. Combining preferences with a modified version of POMCP resulted in the state of the art performance on this domain. Our agent can easily defeat previous implementations as well as a version of itself that uses a non-linear policy extracted via standard supervised learning methods. Furthermore, our agent required a fraction of the decision time compared to the latter. Finally, our ablation studies indicate that our agent heavily relies on the preferences extracted from the corpus.

Related work

Research on improving Monte Carlo Tree Search (MCTS) is very active. Work that focuses on the curse of history is related to the idea of clustering actions into types. The most popular methods in reinforcement learning that address this issue are the hierarchical methods, where Macro actions are defined e.g. (Dietterich 2000; Vien and Toussaint 2015). Macro actions impose a structure on the task which limits the possible policies that the agent can learn. To avoid this, we utilise the approach of Dobre and Lascarides 2017b, who cluster the actions into *types* based on the game rules. We further extend this work by also modifying the tree phase of MCTS and by extracting preferences from a corpus of mixed play. Dobre and Lascarides 2017b offer a detailed comparison to similar methods of improving the rollouts or the tree phase of MCTS. Other related work can be found in standard graph or tree search methods (Xie and others 2014; Lelis, Zilles, and Holte 2013), however these approaches cluster the nodes instead of the actions (edges). In real-time strategy games, Lelis 2017 partitions units controlled by the player instead of actions.

Clustering actions allows us to introduce preferences extracted from a very limited number of games. Biasing MCTS with knowledge that's mined from a corpus of game play is

a well explored path for addressing both the curse of history and the curse of dimensionality. Both tree and rollout policies can be improved via: initialising the action values based on common knowledge (Szita, Chaslot, and Spronck 2010; Silver and Veness 2010), using statistics from previous runs (Gelly and Silver 2007), combining with an opponent model learned via fictitious play (Heinrich and Silver 2015), weighting with a move-prediction model trained on a database of expert games (Graf and Platzner 2016) or with a policy trained via reinforcement learning (Silver and others 2016), or directly applying existing heuristics (Cowling, Ward, and Powley 2012; Dobre and Lascarides 2015).

A related extension is that of constraining the action space during rollouts (Subramanian and others 2016), but their options were generated via crowdsourcing and their constraints following a detailed manual analysis of varied human play. The most related work is that of Bitan and Kraus 2017, who interpolate with a probability distribution that describes general preferences from a database of gameplay over their own abstraction of the action space. There is also some work in combining preference learning with reinforcement learning (Fürnkranz and others 2012). None of the previous approaches integrate preferences conditional on action legality or combine with a type-based system.

Partial observability is a rarely tackled aspect due to its inherent difficulty. The most successful methods applied to POMDP's are based on sampling (Kurniawati, Hsu, and Lee 2008; Silver and Veness 2010) or by planning in abstract representation of the belief (Kaelbling and Lozano-Pérez 2013). In the MCTS literature, partial observability can be handled by "determinizing" the world (Bjarnason, Fern, and Tadepalli 2009), combining the results in a single tree by employing information sets (Cowling, Powley, and Whitehouse 2012) or by attempting to plan in the belief space (e.g. in Phantom games, Wang et al. 2015). Cowling, Powley, and Whitehouse 2012 present several methods for integrating opponents' beliefs by building the tree for each player, i.e. a multiple observer implementation. This is very related to the work in POMDPs that allow for reasoning about nested beliefs (Gmytrasiewicz and Doshi 2005). Even though Settlers of Catan is a multi-agent environment, we only implement a single observer algorithm and we leave such extensions to future work. We want to briefly mention that we do not attempt to learn or improve an existing model of the game as in Bayesian model-based reinforcement learning, where the agent may be uncertain of the environment dynamics (e.g. Guez, Silver, and Dayan 2013).

Settlers of Catan

Settlers of Catan (SoC for short) is a 4 player win-lose game. The rules of the game can be found at www.catan.com. Table 1 shows that it exhibits quite different features from Go, which has been extensively used as a benchmark for learning agents (e.g. Silver and others 2016, 2017). The figures in the table are determined by the game rules, except for the depth and branching factors which are estimated via game simulations where the agents are the state of the art rule-based agents. The table shows that SoC presents a more challenging problem than Go. Even though both have incomplete information (i.e. opponent types are unknown), SoC's state and action space is much larger, and it features imperfect information (i.e. opponents' resources and development cards are hidden) and chance events (i.e. it is stochastic). Further, while SoC has a smaller branching factor than Go, its depth becomes two orders of magnitude greater (11715 average depth) if random sampling is used. This massive depth highlights that random sampling is not appropriate in this domain. Finally, a player can execute multiple actions per turn (minimum of 2, i.e. roll dice followed by end turn) so deciding *when* to end your turn has a high strategic importance.

Property	GO	SoC
Incomplete Information	Yes	Yes
Stochastic	No	Yes
Partial Observable Moves	No	Yes
Imperfect Information	No	Yes
Avg. depth	250	250
Branching factor	250	65
Initial state space	1	$\approx 1.2 * 10^{15}$
Action Space	361	1882
Types of actions	2	8
Number of players	2	4
Number of actions per turn	1	≥ 2

Table 1: Comparison of Go and SoC. The depth of SoC was computed by simulating games with the heuristics agents or via random sampling.

Tracking Belief in Settlers of Catan

Popular methods for tracking an agent's beliefs include particle filters (Silver and Veness 2010), Dynamic Bayesian Networks (Russell and Norvig 2009), to abstract the game representation to only relevant aspects (Kaelbling and Lozano-Pérez 2013) or to approximate the POMDP with a factored representation (Paquet, Tobin, and Chaib-draa 2005; Williams 2005). We chose to use the latter, since a factored representation in SoC can be easily implemented just by following the game rules and without making any strong independence assumptions. All the planning agents described here will use a Factored POMDP (FPOMDP).

In a FPOMDP, the belief is not represented as a distribution over a single state variable. Instead states are represented with M random variables $s = X_1 = x_1, \ldots, X_M = x_M$ and a belief is represented as a joint probability distribution over these variables $b = P(X_1, \ldots, X_M)$. One approach is to assume complete independence $b = \prod_k^M P(X_k)$ (Paquet, Tobin, and Chaib-draa 2005), but this may not work very well in domains such as ours since *sets* of variables determine actions' legality as well as their effects. In line with our general approach of exploiting structure imposed by game rules during learning, our independence assumptions are informed by these rules. For instance, the players' resources and development cards can be tracked independently for the following reasons: *(a)* the partially observable effects of actions modify either what resources a player has (i.e. stealing and discard-

ing) or what development cards a player has (i.e. buying a development card), and *(b)* action legality depends on either what resources or what development cards are owned. On the other hand, we keep track of the joint distribution of which resources each player has (this allows us to accurately model the effects of stealing and discarding).

Since this is a 4-player game, tracking a belief over a complete description of the state would result in an explosion of possible states given the large number of combinations between each possible resource hand and development hand for each player. But, we only require the information for the current player to reason over the next legal moves. Therefore we track each player's resource hands individually and we make the assumption that these are independent. Given that players steal resources from each other, this is clearly a relaxed implementation as it will create a small number of additional impossible states. Nonetheless, this factored belief model provides a cheap belief transition function $b' = \tau(b, a, o)$. A more detailed description of the belief model can be found in Dobre 2018. This approach is specific to SoC, but similar approximate representations can be easily designed for most complex domains.

As Paquet, Tobin, and Chaib-draa (2005) observe, there are three major benefits in factorising the belief in this way: belief update is faster, enumerating possible legal actions is also faster and belief representation requires a lower memory. The first and the second benefits are mostly due to how these only target specific portions of the state as well as specific players. The latter is true because (as with Bayes Nets) the factored approach offers a highly compact representation of the joint probability distribution over all possible states.

Partially Observable Monte Carlo Planning

Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness 2010) was originally proposed as an online planning method in POMDPs. (Silver and Veness 2010) have applied POMCP to single-player environments; so unlike SoC the agent doesn't need to reason over the other agents' actions. In SoC, we need to reason over what opponents can and cannot do in the current belief state, because a failed action would inform the other players of what resources or development cards the agent doesn't have. Following the game rules, the observable game model provides a function, $\prod(s, a) = \{0, 1\}$, that we use to mask illegal actions depending on the state sampled at the root node.

Trading creates cycles that are dificult to manage with random rollouts. To address this issue we exploit typed rollouts (Dobre and Lascarides 2017b). This approach assumes the actions can be clustered into types, e.g. road building is a different type of action to city building. Sampling then proceeds in two steps. First an action type is selected from a list of types T with a policy $\pi_t(T)$. Then a specific action is sampled from the set of actions A_t of chosen type t, with a policy $\pi_a(A_t)$—for instance, if sampling chose the action type of road building, you now sample where on the board to build it. These lists are provided by the game model. In this paper, we'll show that estimating the policy π_t from a corpus of human play performs better than assuming uniform sampling. In the original POMCP algorithm, rollouts are performed with a hand-crafted policy defined in terms of the history h_t consisting of action and observation pairs $h_t =< (a_1, o_1), (a_2, o_2), \ldots, (a_t, o_t) >$. However, our policies (whether mined from the human corpus or made uniform) don't require hand-crafting nor the dependence on history. For us, the rollouts are performed in the observable game.

Negotiations also affect the selection step of the original algorithm since it represents nodes in the tree using the history. For example, the same belief state can be encountered following different trajectories and therefore the histories are different. This is caused by players exchanging resources that the belief model was already certain they have. The most simple degenerate case is where two opposite trades are made one after another and the game is back in the original belief state. Therefore, representing nodes with histories is not efficient as the algorithm will duplicate a lot of effort to relearn the best actions. We overcome this by representing a belief state node with a set of features for both the observable state and the belief. The features are computed using an abstraction function $\omega(b, s)$ and the belief features are bounds for the values of the belief model factors. This representation has proven sufficient to ensure the uniqueness of the belief states and share statistics between nodes.

The obvious downside is that $\omega(b, s)$ requires updating the belief as we traverse the tree. But we only need to perform this update during the selection phase of the algorithm since the rollouts are performed in the observable game. As we will show later, the time required for running the algorithm can be tolerated by a human player. The other improvements presented in this paper do not depend on how an agent's belief is tracked.

Similar to previous work on Go, we only update with the rewards received at the end of the game, 0 for a loss and 1 for a win, and we do not discount these (the backpropagate function is omitted for the sake of space). A further required clarification is that our game model G can provide a list of all possible actions given a belief b. This is just an optimisation to reduce the memory footprint; alternatively one can return all actions that can be executed in the game. Our modified POMCP method is described in Algorithm 1.

Combining with human preferences

Our end goal is to combine the planning method with a method that extracts a policy from a human corpus. The corpus contains mixed play and is made of a total of 60 games (Afantenos and others 2012). These games have been collected by an omniscient server which had access to the exact state description s. We do not have access to the players' beliefs, however we will look into estimating these beliefs and integrating this estimation in future work. One obvious way to improve the rollouts is by improving the policy used to select action types $\pi_t(T)$ over the standard uniform policy. This can be achieved by learning a preference distribution over action types from the game play stored in the corpus. The simplest approach is to estimate this distribution via Maximum Likelihood Estimation (MLE) which counts the number of times an action type was selected in the cor-

Function `Search(`ω, G, $Tree$`)`**:**
  create root node n
  while *within budget* **do**
    $s \sim b$
    n,s,$b \leftarrow$ TreePolicy(s, b, ω, n, G, $Tree$)
    $s \leftarrow$ Expand(s, b, n, G)
    $r \leftarrow$ Rollout(s, G)
    Backpropagate(r, $Tree$)
  end
  $a \leftarrow \arg\max_{a_l} Q(\omega(b,s), a_l)$
  return a;
Function `TreePolicy(`s, b, ω, n, G, $Tree$`)`**:**
  while n *is expanded node* **do**
    $a \leftarrow$
    $\arg\max_{a_l} Q(\omega(b,s), a_l) + C\sqrt{\frac{\log N(\omega(b,s))}{N(\omega(b,s),a_l)}}$
    where $a_l \in \{a | \amalg(s,a) = 1\}$
    s,$o \leftarrow G.step(s,a)$
    $b \leftarrow \tau(b,a,o)$
    $n \leftarrow Tree(\omega(b,s))$ # also creates new node
  end
  return n,s,b
Function `Rollout(`s, G`)`**:**
  while s *is non-terminal* **do**
    $T \leftarrow G.action\_types(s)$
    $t \sim \pi_t(T)$
    $A_t \leftarrow G.actions(s,t)$
    $a \sim \pi_a(A_t))$
    $s \leftarrow G.step(s,a)$ # o is not needed in rollouts
  end
  return $G.reward(s)$
Function `Expand(`s, b, n, G`)`**:**
  $A \leftarrow G.possible\_actions(b)$
  initialise statistics in n for actions A
  select random action a legal in s
  $s \leftarrow G.step(s,a)$ # o is not needed here
  return s

Algorithm 1: Modified POMCP

pus over the number of times this action type was legal. We use a function C to represent the counts, a function $type(a)$ to represent the type of the chosen action a and a function $legal(t)$ to indicate that a type t is legal (i.e. there is at least one legal action of type t) or not (there are no legal actions of type t):

$$\pi^u_t(T) = P(T = t) \approx \frac{C(type(a) = t)}{C(legal(t) = True)} \quad (1)$$

Ideally, one would also like to condition on the state representation, such that this preference would resemble an action value function $Q(s,a)$. Using a tabular representation is impossible in a game of SoC's size when we have access to such a small set of samples. However, we exploit action legality to reduce the effects of the extremely sparse data. Specifically, we condition on the set of n legal action types in the current state s as shown in Equation 2, where the function $\Gamma(s) = \{legal(t_1), legal(t_2), \ldots, legal(t_n)\}$ represents this set of legal types.

$$\pi^c_t(T, \Gamma(s)) = P(T = t | \Gamma(s)) \approx \frac{C(type(a) = t)}{C(\Gamma(s))} \quad (2)$$

This approach captures the fact that a human player's policy includes preferences of executing certain types over other types when both options are legal. The unconditioned distribution may be very skewed towards the more common types, which are more likely to be legal and more likely to be executed in a game. Such a skewed distribution will not reflect the fact that certain types that are known to be good are less likely to be encountered in a game. A simple example in SoC would be to consider how often the player selected to build something over trading when both options are available. Trading is a means to acquire the requirements for executing other actions which in turn will get the player closer to winning the game. The conditioned distribution encapsulates these preferences as well as others such as preferences (or indifference) of building settlements over building cities.

Despite the fact that the action type space is small, we still do not have enough examples to learn a good estimation for each possible condition. Therefore we learn both the unconditioned and the conditioned policies and when we encounter an unseen condition or we do not have any counts of the action being selected when the conditioning is true, we fall back to the unconditioned case. We have not explored any smoothing strategies (other than the backoff strategy just given) to account for fewer or no counts for certain types. Our data is sufficient to at least have some counts for every type in the unconditioned policy. We will refer to the 3 different implementations as:

- *uniform* which uses the typed rollouts with a uniform policy over types $\pi_t(T)$;
- *unconditioned* which uses the typed rollouts with the policy over types $\pi^u_t(T)$ learned using Equation 1;
- *conditioned* which uses the typed rollouts with the policy over types $\pi^c_t(T, \Gamma(s))$ learned using Equation 2, and falling back to $\pi^u_t(T)$ from Equation 1 when the condition was not encountered in the corpus (i.e. $C(\Gamma(s)) = 0$).

Illiciting user preferences is known to be very useful in decision making and a very popular method for representing them is a Conditional Preference Net (CP-Net) (Boutilier and others 2004). Simple preferences can be represented as: $a \succ b$ (a is strictly preferred over b), $a \succeq b$ (a is equally or more preferred to b) or $a \sim b$ (the agent is indifferent to either a or b). CP-Nets are graphs that can represent conditional preferences of the form $a : b \succ c$ (if a is true then the agent prefers b over c). The conditioned implementation provides a distribution over a preference ordering similar to the output of a Probabilistic CP-Nets (PCP-Net) (Bigot and others 2013). The fallback strategy π^u_t provides preferences of the form $a \succ \neg a$ (agent prefers executing action a over not executing action a), while π^c_t provides preferences of the form $ab : c \succ \neg c$ (if a and b are also legal, then the agent prefers doing c over the other actions). The benefits of PCP-Nets is that they account for possible noisy preferences or when the whole set of variables that affect the user's preferences is unknown. In our case, a probabilistic representa-

tion accounts for the fact that there are multiple players that generated the data or these players could have employed a mixed strategy. Their preferences are collapsed to a single preference representation for a standard player.

We also use the 3 implementations to bias search in the tree phase of the algorithm. We extend the PUCT variant used in alphaGo (Silver and others 2016) to bias the search with these policies (see Equation 3). First of all we adapt the formula to use belief states (with a minor abuse of notation, we replace $\omega(b, s)$ with b). Secondly, we only consider the legal actions a and a'. Finally, we compute $P(s, a) = P(a|type(a) = t) = \frac{P(T=t)}{N(t)}$, where $N(t)$ is the number of legal actions of type t and $P(T = t)$ is computed using one of the policies extracted via MLE. Here, s is the state sampled by POMCP at the root node. These distributions can get very peaked so to avoid aggressive pruning in the tree phase, we increase the temperature of each probability before normalising the distribution in order to smooth it. We will refer to the agent that only informs the tree phase with the conditioned type distribution as *POMCP-TS* (Type Selection). Our strongest agent uses the conditioned policy to bias both the tree phase and the rollout phase of POMCP. We refer to this agent simply as *POMCP-TS-CR* (Conditioned typed Rollouts). As a side note, we found that the PUCT policy performs better than the standard UCT even with a uniform distribution over the legal actions. So we use the PUCT policy even for the standard POMCP agent.

$$PUCT(b, a) = Q(b, a) + C \cdot P(s, a)\sqrt{\frac{\sum_{a' \neq a} N(b, a')}{1 + N(b, a)}} \tag{3}$$

Non-linear move prediction

We compare the above MLE method for extracting preferences against a non-linear function approximation that learns a policy from the same data (Dobre and Lascarides 2017a). They train a Deep Neural Network (DNN) in a supervised manner, using hand-picked features and only the observable parts of the states to make their estimates from the sparse data. So the NN estimates a policy given the belief of an agent which ignores all the unknowns: $\pi_{NN}(b) = P(a|b)$. We adapt the PUCT equation to use this policy by setting $P(b, a) = \pi_{NN}(b)$ (see Equation 4). In addition, we added a normalisation term $\mu = \frac{1}{\sum_a \prod_{(s,a)} P(b,a)}$ since the masking performed to account for illegal actions given the sampled state s would cause $\sum_a P(b, a) \neq 1$. This has a negative impact on the exploitation-exploration trade-off of the standard PUCT algorithm and we found that normalising resulted in a 10% absolute increase in the win rate of this agent. The value of each possible action is computed only once, during the expansion step of the algorithm. As with the MLE extracted policies, the output distribution is quite peaked so we increase the temperature of the softmax output layer to avoid prunning the game tree. The agent that informs the tree search of POMCP with the NN prediction while the rollouts are uniform typed, is known as *POMCP-NN*. We only use the DNN during the selection phase, since it is too expensive to apply it in the rollouts.

$$PUCT(b, a) = Q(b, a) + C \cdot \mu \cdot P(b, a)\sqrt{\frac{\sum_{a' \neq a} N(b, a')}{1 + N(b, a)}} \tag{4}$$

Results

The performance of an agent is measured by running 2000 games between 4 players: one of the players is the (modified) agent we are evaluating and the other 3 are baseline agents (all of the same type). So, a player that is of equal strength to the baseline agent would win 25% of the games. We tested the significance of win rates against this null hypothesis using the z-test and a threshold $p < 0.01$. This makes any win rate between 22.5-27.5% not significantly different from the null hypothesis (i.e. a win rate of 25%). Running several simulations in an evaluation is necessary because of the stochastic nature of the game: even a weak player can get lucky! To avoid situations where the modified agent has a successful strategy only because it focuses on aspects of the game that are ignored by the baseline agent, we also evaluate the performance of the baseline agent versus 3 of the modified agents during the tournament experiment.

The first baseline agent is the *Stac* agent (Guhe and Lascarides 2014): this is a hand-coded decision tree, and at the time was the state of the art player for the full version of SoC. We will then evaluate the two preference extraction methods versus a uniform bias by running experiments where each agent is the baseline. Finally, we evaluate our *POMCP-TS-CR* agent versus ablated versions of itself and versus *POMCP-NN* in a tournament style experiment. Even though *Stac* proved challenging for human players (Keizer and others 2017), its performance was under 22.5% win rate versus against all our planning agents except for one (to be discussed shortly). So we do not include these results. All experiments were run on a cluster of CPU machines with 4 cores: the 4 threads were shared with the NN prediction step in *POMCP-NN*. We also limited trading to 3 offers per turn for all agents (to do more would annoy human opponents).

We first focus on the performance of the POMCP agent compared the *POMCP-TS-CR* agent versus 3 Stac agents. This experiment shows how the algorithm scales with the number of planning iterations. The results in Table 3 indicate that the bias from the MLE trained policies is useful even if the algorithm is allowed to perform a large number of iterations. Our *POMCP-TS-CR* agent performs better than an uninformed POMCP agent that runs 2 times more iterations. Increasing the number of iterations makes it impractical to run all our experiments, so we have capped the number of iterations allowed to 10k in the remaining experiments. Since random is too weak of a baseline, we have tuned the parameters of all planning agents versus the *Stac* agents, before pitching them against each other. The best exploration parameter C for an uninformed POMCP is 0.5, while for any biased agent that uses PUCT as selection policy is 4.

In Table 4, we illustrate the benefits of conditioning the typed policy on the type legality. The modified agents are

Modified	Baseline				
	Stac	POMCP	POMCP-NN	POMCP-TS	POMCP-TS-CR
POMCP	33.45%	–	19.65%	22.05%	31.50%
POMCP-NN	34.89%	30.69%	–	24.80%	24.70%
POMCP-TS	40.56%	29.53%	24.82%	–	24.71%
POMCP-TS-CR	40.70%	47.50%	33.30%	30.73%	–

Table 2: Agents' performances in a tournament style evaluation.

Agent	10k	20k	30k	40k
POMCP	33.45%	42.23%	44.5%	47.94%
POMCP-TS-CR	40.70%	49.00%	52.65%	53.65%

Table 3: Win rates of POMCP and POMCP-TS-CR agents against 3 Stac agents, while varying the number of iterations.

specified on the first column and each row contains their performance versus 3 baseline agents labelled on the second row (e.g. POMCP with conditioned typed rollouts wins 34.75% of the games when all opponents are Stac). One POMCP agent with rollouts using the conditioned type distribution to sample action types can easily defeat any of the other agents, while POMCP with the unconditioned type distribution is unable to even defeat the Stac agent (25.63%). Interestingly, an agent that uses a uniform distribution is able to win over 25% of the games versus 3 agents that sample the action types from the conditioned distribution. Even though the 27.19% win rate is not significantly better, it illustrates why we also need to evaluate the performance of baseline agents versus the modified agents. Despite this result, an agent using the conditioned distribution is better since it wins more than 27.19% (i.e. 32.10% win rate) versus 3 agents using the uniform distribution.

Modified	Baseline			
	Stac	unif.	uncond.	cond.
unif.	33.45%	–	22.25%	27.19%
uncond.	25.63%	23.55%	–	17.25%
condit.	34.75%	32.10%	34.80%	–

Table 4: Win rate of the POMCP agents while varying the distribution over types used in the rollout phase.

We have also performed experiments with informing the tree phase of POMCP with the three different type distributions (uniform, unconditioned and conditioned), but we do not include a table due to space limitations (see Dobre 2018 for details). We observed that the unconditioned and the conditioned distributions have comparable results versus Stac ($\approx$ 40% win rate) and versus POMCP ($\approx$ 29% win rate), while pitching the two agents versus each other produced win rates that were not significantly different to the baseline performance (i.e. differences were $\pm$ 0.5%). Stac or POMCP were unable to defeat either improvement. Comparing with Table 4, informing the tree phase resulted in better performance versus the Stac agent, while informing the rollout phase resulted in a better performance against the POMCP agent. So both methods have their merits.

To evaluate which of our agents is the best one, we performed a set of experiments where each agent was the modified agent (see Table 2). POMCP-TS-CR can easily defeat 3 opponents of any other agent type (see performance on bottom row). Interestingly, POMCP is able to defeat 3 POMCP-TS-CR agents, but it is winning fewer games in total (31.50% compared to 47.50%) and it has a weaker performance against the other agents. Another unexpected result is that the performance of seeding with the type distribution (POMCP-TS) is comparable to seeding with a Neural Network (POMCP-NN). POMCP-NN requires 3.3 seconds on average to take a decision, while a standard POMCP agent requires only 2.1 seconds on average. Informing POMCP with types in either rollouts or tree phase of the algorithm slightly increases this average to 2.2 seconds. So this agent has a comparable performance while it requires 2 thirds of computation time. Finally, informing both stages of the algorithm has proven fruitful since POMCP-TS-CR is able to defeat 3 ablated versions of itself (30.73% versus POMCP-TS), while the ablated version is unable to achieve a performance significantly different to 25% win rate versus 3 *POMCP-TS-CR* agents.

Future work and Conclusion

We presented an extension to POMCP to address the challenges of complex games. We exploited the structure imposed by game rules (i.e. action types) to limit the search space and we showed the importance of action (type) legality. Conditioning on the type legality permitted a Maximum Likelihood Estimation approach to learn a high-level policy and be more successful than deep learning techniques. This result emphasizes the need for simpler and more robust methods in complex low-resource scenarios. Our strongest agent heavily relies on the conditioned policy extracted via MLE. Therefore, the human data has proven essential in creating the strongest agent despite its limitations.

There are two extensions that we consider fruitful paths to explore. The POMCP agent with a uniform prior, even though it's not the strongest agent, has proven very adaptable. We believe accounting for opponent types might aid our strongest agent to balance the strong bias from the extracted conditional policy. A possible approach is to use the policy as a prior in a Bayesian setting, where we allow the agent to update it depending on the opponents' types. A related extension is to account for the opponent beliefs, however this requires ensuring that the final agent doesn't exceed a time limit tolerable by human opponents.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *21st ICML*. ACM Press.

Afantenos, S., et al. 2012. Developing a corpus of strategic conversation in the settlers of catan. In *1st GAMNLP*.

Argall, B. D., et al. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469 – 483.

Bigot, D., et al. 2013. Probabilistic conditional preference networks. In *29th UAI*.

Bitan, M., and Kraus, S. 2017. Combining prediction of human decisions with ISMCTS in imperfect information games. *CoRR* abs/1709.09451.

Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding klondike solitaire with monte-carlo planning. In *ICAPS*. AAAI.

Boutilier, C., et al. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR* 21:135–191.

Cowling, P. I.; Powley, E. J.; and Whitehouse, D. 2012. Information set monte carlo tree search. *IEEE TCIAIG* 4(2):120–143.

Cowling, P. I.; Ward, C. D.; and Powley, E. J. 2012. Ensemble determinization in monte carlo tree search for the imperfect information card game magic: The gathering. *IEEE TCIAIG* 4(4):241–257.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR* 13:227–303.

Dobre, M., and Lascarides, A. 2015. Online learning and mining human play in complex games. In *IEEE CIG*.

Dobre, M., and Lascarides, A. 2017a. Combining a mixture of experts with transfer learning in complex games. In *AAAI Spring Symposium*.

Dobre, M., and Lascarides, A. 2017b. Exploiting action categories in learning complex games. In *IEEE IntelliSys*.

Dobre, M. 2018. *Low-resource learning in complex games*. Ph.D. Dissertation, School of Informatics, University of Ediburgh.

Fürnkranz, J., et al. 2012. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning* 89(1):123–156.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in uct. In *24th ICML*, 273–280.

Gmytrasiewicz, P. J., and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *JAIR* 24:49–79.

Graf, T., and Platzner, M. 2016. *Using Deep Convolutional Neural Networks in Monte Carlo Tree Search*. Leiden, The Netherlands: Springer International Publishing. 11–21.

Guez, A.; Silver, D.; and Dayan, P. 2013. Scalable and efficient bayes-adaptive reinforcement learning based on monte-carlo tree search. *JAIR* 48(1):841–883.

Guhe, M., and Lascarides, A. 2014. Persuasion in complex games. In *18th SEMDIAL*, 62–70.

Heinrich, J., and Silver, D. 2015. Smooth uct search in computer poker. In *24th IJCAI*, 554–560. AAAI Press.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *IJRS* 32(9-10):1194–1227.

Keizer, S., et al. 2017. Evaluating persuasion strategies and deep reinforcement learning methods for negotiation dialogue agents. In *EACL*.

Kurniawati, H.; Hsu, D.; and Lee, W. S. 2008. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *RSS*.

Lelis, L. H.; Zilles, S.; and Holte, R. C. 2013. Stratified tree search: A novel suboptimal heuristic search algorithm. In *AAMAS*, 555–562.

Lelis, L. H. 2017. Stratified strategy selection for unit control in real-time strategy games. In *IJCAI*, 3735–3741.

Maddison, C. J., et al. 2015. Move evaluation in go using deep convolutional neural networks. In *ICLR*.

Ng, A. Y., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *17th ICML*, 663–670.

Paquet, S.; Tobin, L.; and Chaib-draa, B. 2005. An online pomdp algorithm for complex multiagent environments. In *AAMAS*, 970–977. New York, NY, USA: ACM.

Pfeiffer, M. 2003. Machine learning applications in computer games. Master's thesis, Institute for Theoretical Computer Science, Graz University of Technology.

Russell, S. J., and Norvig, P. 2009. *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall.

Silver, D., et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484–503.

Silver, D., et al. 2017. Mastering the game of go without human knowledge. *Nature* 550:354–359.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large pomdps. In *NIPS*, 2164–2172.

Subramanian, K., et al. 2016. Efficient exploration in monte carlo tree search using human action abstractions. In *NIPS*.

Szita, I.; Chaslot, G.; and Spronck, P. 2010. Monte-carlo tree search in settlers of catan. In van den Herik, H., and Spronck, P., eds., *ACG*. Springer. 21–32.

Thomas, R. 2004. *Real-time Decision Making for Adversarial Environments Using a Plan-based Heuristic*. Ph.D. Dissertation, Comp. Science Dep., Northwestern University.

Vien, N. A., and Toussaint, M. 2015. Hierarchical monte-carlo planning. In *29th AAAI*.

Wang, J.; Zhu, T.; Li, H.; Hsueh, C. H.; and Wu, I. C. 2015. Belief-state monte-carlo tree search for phantom games. In *IEEE CIG*, 267–274.

Williams, J. D. 2005. Factored partially observable markov decision processes for dialogue management. In *4th Workshop on Knowledge and Reasoning in Practical Dialog Systems*, 76–82.

Xie, F., et al. 2014. Type-based exploration with multiple search queues for satisficing planning. In *28th AAAI*, 2395–2402.