Learning augmented planning for challenging multi-agent envronments

Charles Higgins, Odinaldo Rodrigues, Michael Luck
Kings College London
Center for Doctoral Training in Safe and Trusted AI

Abstract

Recent success in game-playing AI have relied on the combination of model-free and model-based forms of learning and reasoning respectively. This work suggests a development of the learning-augmented search which tailors the search depending on the perceived types of opponent. We provide experimental results showing it to be more robust to uncertain conditions and multiple opponent types against state-of-the art baselines. This work, we argue is among to first line of which which effectively extends reinforcement learning augmented search to larger and more complex environments .

Introduction

It is a truth universally acknowledged that a machine-learning agent in possession of sufficient data must be in want of converging to optimal rewards. However, given a non-stationary environment, such convergence is often not possible. Any environment where the relationship between state and optimal actions change over time is deemed non-stationary, hence, any open environment where other agents might enter or leave, or learn and update their policies in response to stimuli is deemed non-stationary.

Despite prodigious advances in recent years in games posing challenges through enormous state-spaces, requiring strategic thinking and planning, multi-agent systems pose a number of open questions. Crucially, a number of these problems revolve around safe and trustworthy action-selection: in short, how can an autonomous agent act in a way a human might trust? If an agent is in a non-stationary environment, an agent cannot be sure of the results of its action before making it, and hence, cannot reasonably act safely. In the pursuit of safe and trusted AI therefore methods to address this non-stationary are required.

One assumption one can reasonably make is that the non-stationary element of a multi-agent environment is (for the most part) due to other agents, as opposed to an intrinsic part of the wider world. One can then combine a model of another agent, designed to capture the non-stationary component of the environment, with a distinct (and stationary) environment model. Hence, an agent can control, or at the very least limit the negative effects of non-stationarity, and act with increasing certainty. This is the fundamental assumption made by a group of methods which are loosely referred to as opponent modelling. Opponent modelling methods construct a distinct internal model of other agents, and use this model in to adapt their actions to this other agent.

Opponent modelling, particularly within the area of game-playing AI, has had notable successes in closed environments, however solutions have yet to extend conclusively to open environments (Albrecht and Stone 2017). In order to combat some of the challenges posed by openness, an agent must learn not only to adapt to large numbers of opponents, but an agent must be able to perform opponent modelling *swiftly* and *robustly*: it must be able to swiftly reason over the nature of another agent (i.e. its intentions/reward function, observations, and likely actions) and, equally swiftly, use this information to act safely, and ideally, optimally.

Traditional or regular methods of action seleciton for intelligent agents tend to fall into one of two distinct camps: the reasoning or learning approach. The reasoning approach tends to conform to a state-space search style approach, and so fails to perform well in large or dynamic environments. In contrast, model-free learning agents tend to perform well in complex environments, however require a large training time (i.e. they have a low degree of sample-efficiency). Both therefore have significant flaws, and therefore neither are able to meet the criteria of a swift and robust agent (N.B. this line of reasoning is further explained in Section).

To meet this objective: contributing a trustworthy method of action selection which is swift and robust, we propose a multi-agent extension of a Partially Observable Monte Carlo Planning agent architecture (POMCP). A POMCP agent is a model-based agent which uses an internal model of the environment to select actions which is capable of performing in large uncertain environments. We augment this architecture with an opponent-weighted policy-augmented search component. Our contribution builds upon the state of the art in opponent modelling and game-playing AI, and integrates a data-driven learning component within a model-

based reasoning agent architecture. It proves to outperform both purely model-based and model-free counterparts as well as other state of the art baseline methods, given an uncertain ad-hoc coordination environment (i hope...). To the best of the author's knowledge, this is the first hybrid data-driven/model-based system to have been applied to an opponent modelling task.

The remainder of this paper is structured as follows: Section introduces the background theory for opponent modelling, data-driven action-selection in game-playing AI and reasoning approaches. Section formalises the system. Section introduces our contribution Section lists the upcoming experiments and expected results, Section concludes.

Background

This section provides an introduction to various basic concepts and intuitions required to understand the context and contribution of this line of work. It starts with an introduction into multi-agent environments and opponent modelling, before discussing various methods of action-selection and their associated benefits and weaknesses, before finally introducing the core algorithms upon which this piece of works builds.

Open environments & opponent modelling

As mentioned in Section 1, if the environment is stationary – that is, the relationship between the state of the environment and the eventual rewards, is fixed, a learning agent can eventually learn an optimal policy: a mapping between state and actions, which maximises discounted future rewards. Hence, given enough data, an agent can learn to act optimally, and therefore safely, in such environments. However when the relationship between state and optimal actions change, converging to an optimal policy (at least by the same mechanism) is not feasible. This poses a problem as in reality, most environments exhibit a degree of non-stationarity. For a number of these environments, non-stationarity comes as a result of other agents. Take for example, the challenge of an autonomous vehicle navigating down a corridor.

In a single-agent environment, there is a simple optimal policy. Given enough iterations, an agent can learn how best to optimise actions to eventually move (in as few moves as possible) towards the exit.

Given a more complex relationship between optimal action selection and state, like partial observability, or noisy movements (i.e. the relationship between state-action pairs and the successive state is probabilistic), the optimal policy should adjust, and will remain fixed.

Consider now, the same environment, but there exists at least one other agent. The optimal policy now depends on the state of the environment and the position of the other agent. Now, reasonably, one could image the other agent as being included in the state — and indeed, if the other agent has a fixed policy, the environment is still stationary, however if that other agent learns, or updates its policy, or enters and leaves, the environment is fundamentally changing, and so too will the optimal policy.

This non-stationarity poses problems for agents which are required to select actions safely and robustly. Simply put, if the results of an action depend on the actions taken by another agent, (i.e. the transition function which affects changes in the environment depends on more than a single agent's action), an agent must understand and predict the actions of another agent in order to preempt the likely resulting state. This idea fundamentally underpins opponent modelling — containing or compartmentalizing an element of non-stationary and uncertainty within another agent model, and constructing and using these models in order to reduce uncertainty and ultimately act optimally.

Methods of opponent modelling

There are two distinct challenges posed by opponent modelling. The first is how to *construct* an opponent model for a specific opponent (or set of opponents). The second involves *using* that opponent model — incorporating an opponent model into its action selection mechanism.

The first challenge is rather difficult to approach, as depending on the constraints and flow of information afforded by different environments and situations some are better suited to challenging conditions than others.

Arguably the simplest and most common form of opponent modelling is known as policy reconstruction. An agent attempting a form of policy reconstruction attempts to build an agent's policy, often with little former knowledge and simply building from observations alone. While this can be very effective given multiple observations, in a short period of time, this is unlikely to succeed. However, the policies can be very complex, and can contain elements of learning and constant updating (see for example (Leibo et al. 2017) for learning with opponent learning awareness, or (Mealing and Shapiro 2017) for a recent and sophisticated form of policy reconstruction under imperfect information).

A common and far swifter form of opponent modelling is type-based reasoning (see (Barrett, Stone, and Kraus 2011) for a number of recent well performing methods in a number of challenging environments). In type-based reasoning, an agent model is pre-built based on previous observations with other agents, or supplied by the user. These agent models are then attributed to an actual opponent at the moment of encountering an opponent. This reduces the task of opponent modelling to one of classification rather than one of policy reconstruction, and can allow for very sophisticated agent models to be built and used swiftly with minimal effort and observations necessary.

Type-based reasoners, and to an extent, policy-reconstruction models allow for the generalization of

an agent model as a black-box of sorts, and simply apply it to certain situations, without a huge amount of care as to how it really works. In contrast, arguably the most complex form of opponent modelling starts to encroach on a concept known as theory of mind: this is the attribution of latent mental states like beliefs, desires, and intentions to other agents in order to attempt to understand and predict their actions. This has shown to be particularly effective in predicting actions in scenarios involving incomplete information and deliberate deception, however this often leads to recursive cycles of beliefs about other agents beliefs — i.e. multiple orders of theory of mind (Panella and Gmytrasiewicz 2017). This tends to be computationally draining and fails to be of practical use in scenarios with limited interaction protocols.

The most applicable form of opponent modelling to an open environment, and one which crucially has the capacity to be swift, is that of type-based reasoning. It is pertinent to note at this point that a type-based reasoning form of opponent modelling can easily (and has recently (Albrecht and Ramamoorthy 2016)) encapsulate a form of policy reconstruction, or even a theory-of-mind agent.

Ad-hoc coordination As the name might suggest, the majority of opponent modelling work has been developed in purely adversarial environments: e.g. Chess, Go, Poker (Brown and Sandholm 2017; Brown et al. 2020; Silver et al. 2016). In two player zero sum games, opponent modelling is challenging due to large state spaces and strategy profiles, however the intentions of agents are clear, and the solutions can often be reduced to equilibria and modified MiniMax pruning (as for instance in (Brown et al. 2020)).

In recent years there has been increasing interest in mixed-motive games, where agents must interact with friendly as well as adversarial agents (Barrett et al. 2017). Here solution concepts are arguably more challenging to derive due to the comparatively larger number of opponents/other agents and their mixed motives. One such challenge is referred to as AD-hoc coordination — playing a game with mixed motives with unknown agents. In these environments, an agent must discern whether other agents are cooperative, and how they might cooperate and compete to achieve aligned or conflicting objectives.

There are several similarities in testing grounds of this sort and open environments: large number of agents, agents with unknown motives and reward functions and considerable uncertainty. These ad-hoc coordination environments therefore provide a school of methods and well established challenges to adapt, develop and tune agents capable of performing well in an open environment.

Action-Selection

There are two broad approaches to action-selection for autonomous agents: a learning vs a reasoning approach[1]. The former starts with limited knowledge about the environment, and through multiple iterations maps a policy from state to action via rewards. The latter typically requires comparatively more domain knowledge, and reasons about actions before they are made in order to meet some objective/reach a desired state.

Learning vs Reasoning (high level) In general, data-driven approaches can perform well, given sufficient time and resources to learn an optimal policy within a stationary environment. This policy often takes a huge amount of time to learn and develop as the agent learns the effects its actions have upon the environment. However, once learned, typically the execution is very swift — depending on the exact architecture used times vary — usually a single look-up in a Q table, or a single forward pass through a neural network. They also require very little expert knowledge to implement, and given a stationary environment, can learn an optimal policy to maximise rewards.

In contrast, the reasoning approach requires very few, if any, learning iterations before performing well. Rather than learning from scratch, reasoning agents typically have a model of the environment (hence the name model-based) which they query, testing out the likely effect of each possible action, and subsequent actions. Typically, such agents are referred to as planners. These planning agents search for a chain of actions which take them from their initial state to a desired goal state. As the search space expands exponentially in the number of sequential actions, this search is costly, and hence, at run time, these agents tend to be very slow.

In designing a swift agent therefore, one would require the initial performance of a reasoning agent, however the low-latency and swift response of a learning/data-driven agent.

One must also address the concept of robustness, and so the consequential qualities of safety (and subsequent trust, or lack thereof): data-driven or learning agents are hard to predict — should the environment differ from that of their training, performance degrades significantly. Further, such are prone to unpredictable and inexplicable action-selection, limiting their applicability from fields which contain any real element of risk. In contrast, reasoning agents are often deemed 'safer'. They typically adapt well to changing circumstances, as their internal model already captures some of the environment's dynamics. Furthermore, as such agents they investigate the results of each action before they undertake it, given the assumption that their comprehension of the environment dynamics is correct, they act safely and robustly.

[1]While this section is deliberately high-level, it makes a crucial point as to the strengths and weaknesses of data-driven learning vs reasoning approaches.

Reinforcement learning augmented search In recent years, the combination of model-based search methods and deep reinforcement learning (via self-play) has achieved super-human performance in a number of benchmark challenges in game playing (adversarial) AI (Silver et al. 2016; Brown et al. 2020; Lerer et al. 2019). The major difficulties of these domains (Chess, Go, even Poker) is that of a truly enormous state-space. Research which has achieved notoriety has combined local state-space search with deep neural networks to approximate complex functions.

While each of the aforementioned techniques and successes contribute different novel features, the basic structure of these algorithms remains constant: a sample-based search algorithm augmented with two learning components, commonly referred to as a value network, and a policy network respectively.

Briefly, these algorithms structure the environment as an extensive form game (a search tree). Nodes represent states, and actions represent transitions from one state to another. Given a simple environment, a planning algorithm can search through the possible states, and select the optimal action in order to maximise rewards. However, given a complex environment, the possible state-space expands exponentially, swiftly rendering a typical search-based solution intractable. To address the issue of an enormous search space, a *sample-based* search algorithm is employed.

A sample based search algorithm constructs a search tree of immediately reachable states (rather than all possible states), given available actions. As there are a huge number of possible sequences of actions, rather than exhaustively searching the entire search-space, a sample-based search tree expands and samples only some branches (possible options), and averages across the expected rewards of immediate actions to select the best action at a given node in the tree at a given time.

Even with a sampling-based approach, the search space is often still prohibitively large, and so data-driven learning methods can be used to restrict the search space further to improve performance. Two distinct learning components (deep neural networks) are typically employed.

Both networks are trained through a combination of supervised learning and reinforcement learning via self-play (playing oneself). At run-time, the policy network guides the search through the search tree (limiting the branching factor from areas of the state-space which are unlikely/unfavorable). The value network truncates the depth of the search tree by approximating the value (i.e. the expected discounted reward achievable) for a given state. In short, the learning parameters allow for a tractable approximation of vast state-space, while allowing for explicit reasoning over the short-term, or immediate search-space.

While arguably the most famous result was DeepMind's Go-playing AI AlphaGo (Silver et al. 2016), which rather dramatically defeated Lee Sidol, both Chess and Go are perfect information games — the state of the game is certain and visible at all times. In a development of earlier Poker-playing bots, Brown et al. (Brown et al. 2020) tackled the problem of imperfect information games (namely head's up no limit texas hold 'em poker) by adapting the notion of state to a public-belief state. In short, they define a mechanism for transforming an imperfect information game into a continuous state-space perfect information game, where the state contains a probabilistic distribution over all agents beliefs (a public-belief state). Through this transformation, a similar algorithm to Silver et al's AlphaGo can be used, with policy and value networks guiding and truncating search.

The following Section describes the building blocks of all algorithms mentioned, and formalizes the environment for the sake of common notation.

Formalisation

Partially observable Markov Decision Process POMDP

The environment can be phrased as a partially observable Markov decision process (POMDP). A POMDP is defined as a tuple:

$$\langle S, A, O, P, Z, R, \gamma, b_0 \rangle \tag{1}$$

S is the set of states, A the set of actions, O the set of observations;

$P = Pr(s_{t+1}|s_t, a_t)$ is a stochastic transition function, mapping a state and action to a new state.

$Z = Pr(o_{t+1}|s_{t+1}, a_t)$ is the stochastic observation function — observations are not certain, hence the it is defined as the probability of observing a certain observation having taken an action in a given state.

$R = R(s_t, a_t)$ is the reward function;

$\gamma \in [0, 1]$ is the discount factor (discounting future rewards);

b_0 is the belief about the initial state.

In every timestep t, an agent receives a reward (r_t) and an observation (o_t) and takes/selects an action. The agent aims to derive a policy π which maximises its total discounted reward given its beliefs in a certain state.

Crucial to efficient planning and action selection for a model-based agent is minimising or eliminate the mismatch between the agent's belief/encoded model of the environment and the dynamics of the real environment. This is challenging given that observations are not perfect and only display part of the environment, and the transition between each state is not certain, and doesn't solely depend on a single agent's action.

To simplify, let the state-transition function be interpreted as $s_{t+1} \sim f(s_t, a_t)$, and the observation function be $o_{t+1} \sim g(s_{t+1}, a_t)$.

Given that the state can be changed by other agents, one can view the environment as having a joint action-space between the actions of an agent and all other agents present. Hence, with the assumption that another agent is present, one can then reform the state-transition function and observation functions to take into account the joint actions of all other agents and the environment dynamics, which shall be represented by Θ:

$$s_{t+1} \sim f(s_t, a_t|\Theta) \tag{2}$$
$$o_{t+1} \sim g(s_{t+1}, a_t|\Theta) \tag{3}$$

Opponent modelling can be seen as an attempt to improve this model Θ, by explicitly modelling an opponent, or group of opponents. The quality of an agent's internal model, and therefore the quality of the overall policy derived, is dictated by the quality of the state-transition and observation functions.

One can view Θ as parameters to be tuned for an encapsulated (internal) environment model, which when fed to a black box environment simulation (the function $f(s_t, a_t|\Theta)$ provides a projected next state.

By way of an example, an agent would enter an environment with a belief over the likely parameter(s) of Θ. As interactions continue, an agent can update the value(s) of Θ based on observations: the values of Θ are not immediately obvious however they can be inferred from observations/prior knowledge and some form of reasoning. Hence, the belief of an agent at time t can be seen as a joint distribution over:

$$b_t(s, \Theta) = Pr(s_t, \Theta|h_t) \tag{4}$$

where h_t is the history of observation, reward, action triples ($h_t = [(a_0, o_0, r_0), (a_1, o_1, r_1)...(a_{t-1}, o_{t-1}, r_{t-1})]$) taken to reach the state s_t. In other words, the opponent model is included in an uncertain state-representation. Hence, in reasoning over the state, the agent also reasons over possible opponent models given an action history.

Parameterisation of Θ

The previous section provides a general framework of simulation-based action-selection which makes use of an internal agent model to better inform an observation and state-transition function. This serves to show how an opponent model might fit into the action-selection pipeline. [2].

In the simplest form, Θ refers to a vector containing agent models: $\theta_0, \theta_1, ..., \theta_n \in \Theta$.

[2]The definition of Θ, and methods to accurately estimate Θ, where theta refers to an number of opponent models in an open environment is completed by a simulation based component known as a particle simulator (this is explained further in Section)

Type-based reasoning Each $\theta_i \in \Theta$ is an agent present in the environment. A popular and swift method of opponent modelling is type-based reasoning. Rather than building an assumed policy from scratch at run time, a modelling agent assigns a pre-specified agent model to an observed agent. This has often been supplied by the user, or (recently) has been learned over repeated interactions with other (similar) agents. This leads to a (potentially) large number of agent *types*: $\phi \in \Phi$.

Hence, the parameter of ϕ in θ_i refers to agent model θ_i as being of type ϕ.

Hence, in terms of the belief updating, one can rewrite the belief over state and model parameters, given an observation history to the belief over state and type of model, given an observation history.

$$b_t(s, \Theta) = Pr(s_t, \theta_i = s_t, \phi|h_t) \tag{5}$$

The building blocks: basic algorithms

The following section defines the basic methods upon which the contribution builds. It begins by describing a weighted particle filter for probabilistic state-estimation, a Monte Carlo Search Tree, and finally describes the architecture of a (modified) Multi-Agent Partially Observable Monte Carlo Planner (Hayashi et al. 2020; Silver and Veness 2010).

Particle Filtering: iterative approximate Bayesian Updates

Traditionally the use of Bayes Law is used iteratively in each timestep to reason over uncertain information to update the posterior probability of being in a certain state.

$$b_{t+1}(s') = \frac{O^{s' a_t z_{t+1}} \Sigma_{s \in S} P^{s a_t s'} b_t(s)}{\Sigma_{s'' \in S} O^{s'' a_t z_{t+1}} \Sigma_{s \in S} P^{s a_t s''} b_t(s)} \tag{6}$$

In complex environments (particularly those where there are multiple opponents), performing a complete Bayesian update can be so costly as to be intractable. Hence, recent works in opponent modelling have used deep-learning to approximate the update (see for example (Doucet and Johansen)). However deep-neural nets require training, and can take many iterations before being successful. A recent strategy for opponent modelling (adapted by (Hayashi et al. 2020)), however proposed to expand Monte-Carlo planning to partially observable domains is a sampling based approach which makes use of a generative environment model. This environment model (G) can be used to perform Bayesian Filtering.

The belief state (i.e. the belief distribution over possible states) is sampled for a history h_t by K particles, with index i and a weighting of w

$$k \in K, k = \langle i, w \rangle; B_t^i \in S, 1 < i < K \tag{7}$$

Each sample represents a possible state, and the belief-state is the weighted sum over all particles.

Particles are iteratively removed and reinvigorated at each timestep. At the start of a game/iteration, K particles are sampled from the initial belief-state. The agent selections an action $a_{t,real} \in A$, and receives an observation $o_{t+1,real}$.

Each particle is then evaluated. From the state in K, an agent uses the internal generative model to provide a sample of a successor observation:

$$o_{t+1,possible} = G(k_i, a_t) \tag{8}$$

This possible state is then compared to the real observation via a distance metric to return a value of the accuracy of this particle $V(k)$, which in turn is used to update the value of each particles weighting (w_k) (by a learning rate denoted γ)

$$V(k_i) = o_{t+1,possible} - o_{t+1,real} \tag{9}$$

$$w_{k_i} \leftarrow w_k + (-\gamma * V(k_i)) \tag{10}$$

The weaker particles are pruned based on a hyperparameter depending on the relative strength of weightings. Finally, new particles are created by sampling and adding Gaussian noise to existing particles.

The result is that as the history increases (i.e. more observations are evaluated) the belief-state should converge as inaccurate state hypotheses are filtered, and more accurate hypotheses increasingly suggested.

Monte Carlo Tree Search — an approach to search in a large state-space

Monte Carlo Tree Search is a sample-based search algorithm. Similar to a particle filter, it requires a generative model of the environment (G) to evaluate possible actions and future states in order to converge to an optimal policy online (i.e. re-evaluating from each state).

An agent begins in a state ($s_0 \in S$). From this state, it constructs a sample-based search tree of possible successor states.

$$T \sim \langle N, E \rangle \tag{11}$$

$$N \sim \langle s, q, v \rangle \tag{12}$$

In this search tree, each node ($n \in N$) represents a possible state ($s \in S$), and transitions (edges ($e \in E$) are possible (probabilistic) actions, mapping a node to a probable successor node given a state and action pair.

In each iteration, the tree is traversed in a best-first search procedure. Typically, the value of selecting a node is estimated by the upper-confidence trees algorithm (an adapted upper confidence bound UCB1 bandit algorithm). This weights the relative value of the explored node with an exploration value (derived by the expression on the far right of equation 13) which weights the number of times that node has been explored relative to its parent node, by the scalar constant c (which typically takes the value of 2).

$$V(n \in N) = V(n) + c\sqrt{\frac{log(parent(n)_{visits})}{n_{visits}}} \tag{13}$$

Given a sufficiently large tree, this approach has shown to converge (eventually) to an optimal policy (Ross et al. 2011). Having reached a leaf node (i.e. a node in the tree T with no children), children are generated by *expanding* the node. In this case, all possible successor states of the node are created and added to the tree by use of the generative environment model G.

$$s_{t+1,possible} \leftarrow G(s_t, a_t) \tag{14}$$

Having generated the successor states, the agent estimates the value of each state by performing random *rollouts* to a predetermined depth. These *simulations* (sometimes referred to as *rollouts*) take the form of a random sequence of actions, and return the rewards for the successive states. The reward of potential future states are sampled from these rollouts by taking the mean eventual reward for a single starting state. For x number of rollouts:

$$Q(n \in N) = \frac{1}{x}\Sigma_{i=0}^{x} R^i \tag{15}$$

where R^i is the reward over the i^{th} rollout.

Finally, the estimated values of the leaf nodes are backpropagated throughout the tree, in order to allow for potential future states to add value to their parent states, until finally all the possible moves available in the initial state (the root node) have accurate values, from which one can select an optimal action.

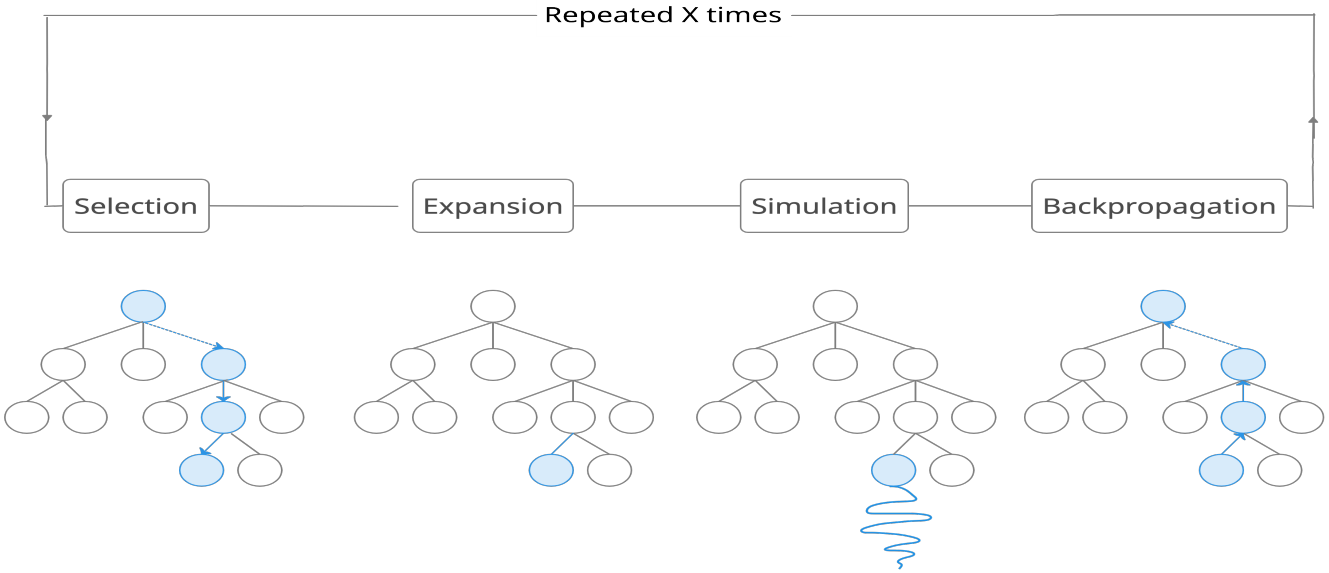
For a fixed time-period or for a fixed number of simulations, the cycle of *tree-traversal*: traversing the tree T; *expansion*: simulating all possible actions in a given state to add all possible successor states to the tree; *Simulation/rollouts*: performing random rollouts to estimate the value of the state; and *backpropagation*: propagating expected rewards back up the tree. This cycle then continues for a determined period, until a new action is selected, and a new state observed.

This approach, typically when paired with a strong heuristic and an accurate transition/environment simulator, has shown to be an effective and powerful approach to action-selection in complex environments with large search spaces — this algorithm forms the basis for most of the recent successes in adversarial AI (as described in more detail in Section)

Aadapting Monte Carlo Tree Search to partially observable environments Despite the success of a sample-based approach, the extension of planning to uncertain (partially observable) environments still poses problems. In such an environemt (due to inperfect or incomplete observations), an agent is (initially) uncertain of the actual state. This causes considerable complexity, and has ramifications for the time and space requirements for computing an optimal policy.

In an n-stateful environment, an agent must compute a distribution over n states representing its belief over the true state of the environment (also known as a belief-state).

Figure 1: Graphical representation of the four stages of a single iteration a Monte Carlo Tree Search, where X represents the number of samples taken per timestep.



It must compute this in addition to the generic planning complexities of all possible actions, transitions and resulting states — hence unless facing a very trivial problem in an exceedingly simple environment, the exponential complexity of partially observable environments tend to render planners inapplicable.

Silver and Veness (Silver and Veness 2010) combined a sampling-based approach to belief updating (particle-filtering) with a monte-carlo tree search style algorithm which allows for partially observable planning. Among a number of alterations, each node in the tree is based on an observation history rather than a state, reflecting the agents belief over the state.

In small state spaces, the belief-state (i.e. distribution over the possible state) can be perfectly calculated by applying Bayes rule — in large state spaces this can be computationally demanding, and a compact representation of the transition model (in terms of likelihoods) might not be available. To address this problem Silver and Veness contributed an algorithm, Partially Observable Monte Carlo Planning (POMCP), which uses a particle-filter: generating a number of small unweighted particles, each representing a possible state, and evaluating them based on expected vs received observations (as described in detail in Section and Section). They use this sampling-based approach to update the belief about the likely history at each time-step, and iteratively converge onto the true state.

As with typical Monte Carlo methods, the sampling approach greatly limits the search space as only (likely) reachable states are evaluated, and the belief over states can be efficiently computed. In short, sampling based methods have shown to allow for swift approximation over complex belief spaces, and the combination of a particle filter with a Monte Carlo Tree Search allows for accurate estimation of the belief-state, and accurate approximation of a near-optimal policy despite complex environments.

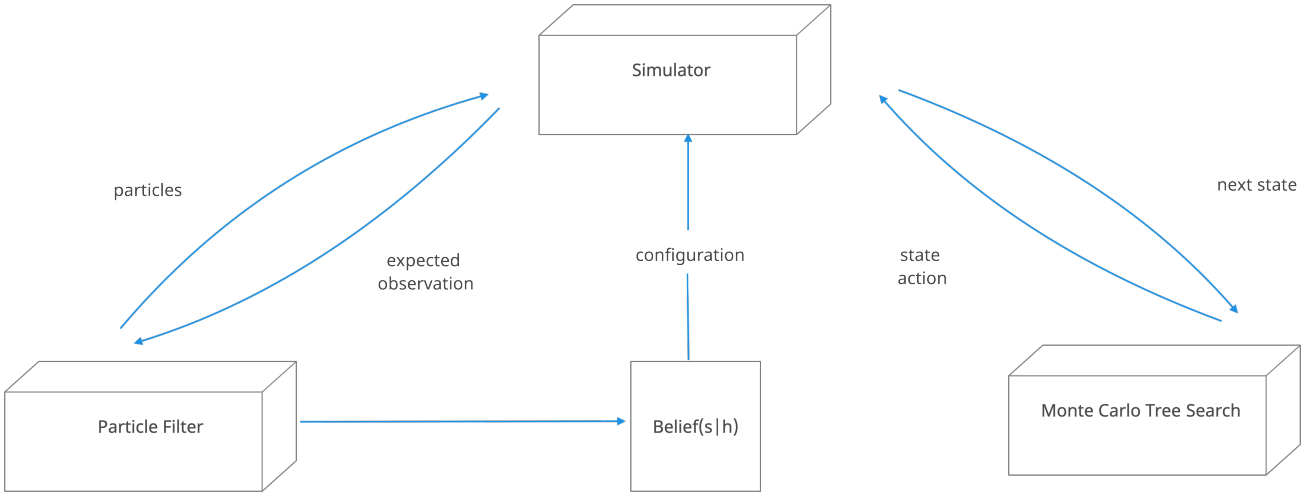
Improving on POMCP — updating the environment simulation Despite the theoretical success of POMCP, belief-based planning still poses challenges. While known for its efficacy in dealing with large state-spaces, MCTS is limited by the accuracy of the transition model. Simply put, if the model of the environment provided to the planner is not sufficiently expressive as to capture the environment dynamics, despite efficient sampling, the performance of an agent will be poor.

In recent years, there have been two notable works which have augmented belief-based planners with a learning capacity in order to overcome an incorrectly specified model.

Katt et al (Katt, Oliehoek, and Amato 2017) augmented the notation of a POMDP to take account of a number of extra features, to allow for a Bayesian updating of the environment model — in effect the transition dynamics were updated as experience increased in a Bayesian way, allowing for more realistic rollouts and better action selection. In short, they incorporated a learning element into the model of the environment to allow for an incorrectly or incomplete black-box environment simulation.

With a similar aim, Hayashi et al (Hayashi et al. 2020) augmented a POMCP planner with a deep-recurrent neural network as a mechanism for particle reinvigoration (sug-

Figure 2: Graphical representation of the POMCP architecture: the Simulator/Generative model is used both to reason over the apparent state based on an action history, but also to reason over the best possible action absed on a sample-based planner



gesting possible states to evaluate), which suggested better candidate states (particles) to be computed allowing noisy or incorrectly specified environments to be used without resulting in poor agent performance. They phrased the environment simulator as a black-box with various parameters which required fitting. This fitting took place as experience grew, meaning that the black-box simulator could be tuned online, allowing the world-model to be iteratively updated. Crucially, they applied this to an opponent modelling task, the level-based foraging domain (Papoudakis et al. 2020; Barrett and Stone 2015), in which agent types were parameters within the black-box simulator.

Incoporating agent models into MCTS As mentioned previously, a MCTS involves several stages: 1) traversing a tree via a best-first (heuristic) until reaching a leaf node; 2) Upon reaching a leaf node, it performs a rollout, which involves selecting an action, and sampling an environment simulator for a possible next state and expected reward. It continues this rollout to an arbitrary depth. 3) Finally, having reached a maximum depth, the reward is propagated back up the tree to the root node.

In a multi-agent environment, the transition function responsible for mapping one state to another depends on a *joint action* taken by all agents. Hence, in a rollout, the simulator must assume a joint action (i.e. infer/suggest likely actions taken by *other* agents). In this way, agent models are integrated into a MCTS via the simulation phase. In simple terms, they are implicitly included in the environment model — this is defined mathematically in Section . In both (Hayashi et al. 2020) and (Albrecht and Stone 2017) agent models are computed and viewed as parameters to be tuned in this environment, thus providing an interesting middle ground between a 'global' approach, in which an agent simply learns a single generative model of the environment from scratch (no opponent models), and a typical opponent modelling approach, which requires distinct agent models and environment models.

This method also has benefits as it translates the problem of opponent modelling into one of state-space search.

Shortcomings of related/current work

Shortcomings of RL + Search

The majority of work in the RL + Search space has focused on 2-player zero-sum games. This allows for (implicit) opponent modelling via the use of a policy function to guide search: players' intentions are symmetric — the reward function for a player's opponent is the inverse of that of the player, and as such self-play is a feasible and optimal method of training. However, when one moves to a general-sum game (with more than a single opponent), the intentions and characteristics of an opponent are harder to discern, and are likely to vary considerably. Therefore, more explicit and granular representations of a varying/heterogenous opponent types are needed in order to extend this architecture into a general sum, n-player environment.

RL + Search architectures have focused on well-defined games with perfect observations, and where observations are imperfect all agent actions are public: in effect, all unknowns are known-unknowns, rather than unknown-unknowns. In more fluid games, the adaptation of a partially observable environment cannot always be quite so well translated into public-belief states, and as such, methods like type-based reasoning might still be required despite their relative simplicity and requirement for prior-knowledge.

Shortcomings of sample-based planning

While belief-based planners reason well about agent types, they have failed to adequately address swift action-selection

despite having the capacity (and success) of computing a considerable amount of information about opponents. In Hayashi et al. (Hayashi et al. 2020), an opponent model is used solely in the roll-out (black-box simulator/transition simulation) as a predictive model, and not to guide play actively — action-selection (i.e. tree traversal), and rollout direction is typically guided by a generic upper-confidence tree algorithm. The inclusion of a policy network to guide the search tree in an partially observable environment has yet to be truly completed, despite type-based reasoning taking place and being included within the form of parameters for a particle reinvigoration engine. In colloquial terms, information is being computed but not as efficiently and effectively used as it could be.

Finally, there seems a distinct lack of work regarding actively seeking out information in action-based opponent modelling (N.B. outside of dialogue games). This is somewhat surprising as this area has been identified in several surveys as an open problem, and one worthy of consideration (Albrecht and Stone 2018; Hernandez-Leal et al. 2017). The inclusion of an information theoretic reward (by way of a value function) in an opponent modelling context would be a novel contribution within itself, however it too would be a simple and strong addition to an RL + Search based opponent modelling architecture.

In brief: RL+Search architectures have yet to be applied and tested in a multi-agent domain where they must reason about uncertain parameters of opponenet. Belief-based planners have proved to reason well about opponent parameters, however have yet to best leverage learning to shape action-selection despite having shown promise in developing techniques to compute this information.

Opponent specific policy guidance

Previous architectures which have fared well in ad-hoc coordination style environments have applied an agent-policy as opposed to an agent-model (Barrett and Stone 2015). This works well with type-based reasoning agents where one can be sure about an agent type, however relies on accurate agent models and a large amount of time to compute them.

Now, as discussed previously, the purely learned strategies are difficult to be globally accurate, and fail to update swiftly given a new opponent. However, if one has access to an agent type, it stands to reason that one could compute a policy and apply a policy should an agent of that type arrive.

In recent work, reinforcement learning has been used to restrict the search spaces to only promising moves. The major contribution of this piece of work is to apply opponent-specific policy networks to restrict the search space. Further, the extent of the restriction is dictated by an agent's confidence over the accuracy of an opponent model. Hence, if an opponent is complete unknown, the policy is not used at all, however, should the opponent be similar to an opponent, or similar to two different types of opponent, the search space should be pruned to allow for more searching in areas of the search space which are the most promising for agents of that sort.

The generation of these policies relies on offline training of cardinal opponents — i.e. opponents of a certain sort for a considerable period of time. Given that this policy is subsymbolic, we employ an augmented form of Deep Q learning (dueling Q networks) in order to learn a policy. these networks employ the Bellman equation to provide Q values for all possible actions when given a certain state — in this case, the observation an agent receives in each timestep.

Contribution

In each timestep, the generative model G used for both the particle filter and for the Monte Carlo Rollouts evaluates particles, and updates the belief-state. In terms of the environment formulation in Section , it updates the parameters of θ, i.e. the belief over opponent models. For simplicity, the opponent types are limited to four cardinal types — see appendix for explanation.

Having updated the values of θ, the Monte Carlo Search Tree is constructed. The major change from the normal comes in the rollout phase, or specifically, in tayloring the rollouts.

In each rollout, opponent actions are sampled at random from the set of available moves. In our suggestion, opponent moves are suggested by the opponent model sampled with a distribution computed by the particle filter. By doing this, the joint-action which ultimately dictates the resulting state is, assuming accurate classification of opponents, more accurate.

Secondly, during generic Monte Carlo rollouts the agent's actions are selected at random from the set of all agent's possible actions. In our modification, the agent's actions are sampled based on the probability of certain opponents, from the policies corresponding to those opponents. Therefore further pruning the samples to explore the search space which is likely to be the most promising.

We suggest that this is the first attempt in the field to have employed opponent-weighted policies to prune a search space for a fundamentally model-based agent.

Experimental Evaluation

Ad-hoc coordination: a challenge

A large proportion of opponent modelling research has been directed at adversarial games. This allows a number of assumptions to be made about opposing agents (hence the name 'opponent' modelling). The major assumption made is that of a purely adversarial opponent. From this assumption, one can assume that the reward function of an opposing agent is the inverse of their own. Given the assumption of an opponents reward function (or at a higher level, their

intentions), one can employ a number of techniques to optimise actions — for instance, pre-computing equilibria via iterative methods (e.g. fictitous play), self-play (training against oneself) and minimax pruning. These techniques have been used well previously in zero-sum games like Poker, Go and Chess. When one extends to environments which require cooperation as well as coompetetive objectives, these techniques require modification to be applicable.

The major effort behind this line of work is to extend opponent modelling techniques to more general problems, or more specifically, open environments. A growing field of research investigates ad-hoc coordination games. These are typically games which are sufficiently simple to be computed by an agent with limited computational power, but incorporate sufficient complexity as to be challenging. Rather more importantly, these games involve self-interested agents (i.e. no common world-model or shared objectives) and allow for competetive agents, however often require cooperation to acheive optimal results. They often involve more than a single other agent, and those other agents often can be deliberatly adversarial or cooperative in nature.

This environment shares a number of aspects with open domains: a number of heterogenous agents, exhibiting a variety of intents and abilities in a partially observable (uncertain) environment. Hence, this serves as a strong test-bed for an opponent modelling system designed for an open system and provides a number of state of the art systems to augment and benchmark against.

The level-based foraging domain A plausible environment (among a number of others) are variations of the Level-based foraging domain (LBFD) (used by (Albrecht and Ramamoorthy 2019; Barrett and Stone 2015; Hayashi et al. 2020; Papoudakis et al. 2020)[3]). The LBFD is represented by a two dimensional gridworld with a number of self-interested agents. The gridworld is populated by agents and objects. Each agent and object has a *level*. The game finishes after an arbitrary number of timesteps, or when all objects have been foraged. Agents have 6 discrete actions in each timestep :(north, south, east, west, stay still, load block). All agents more asynchronously, and select a single action per timestep. Agents receive a reward for foraging an object equal to the object's level. Agents cannot forage any object with a higher level than their own. In the case where an object's level is higher than an agent, agents can cooperate in order to forage the object together, providing that the sum of their levels' is greater than that of the object.

Baselines in ad-hoc coordination

1. Plastic Policy (Barrett and Stone 2015)

   Barrett et al. make a key contribution: they learn an optimal policy for acting with teammates in a simulated robo-soccer league via fitted q learning. They argue that in a complex domain it is necessary to move from a model-based approach to a policy-based approach due to prohibitively large state-spaces. At runtime they select which policy to use by a nearest neighbour classification based on action histories. This contains no feature of learning at runtime and assumes that the types are sufficiently expressive, and that the policies are optimal. Hence, when exposed to unknown agents, this *should*, in theory, perform worse than methods which have internal parameters to update.

2. Albrecht and Stone: reasoning about uncertain agent parameters(Albrecht and Stone 2017).

   Barrett and Stone addressed the problem of static types by including parameters within agent types, and perform an element of policy reconstruction within type-based reasoning. They suggest 3 methods of updating parameter values (approximate gradient ascent, approximate bayesian updating, and exact global updating) and provide a heuristic for selectively updating type's parameters. They then use a MCTS algorithm to evalute the best action.

3. Hayashi et al — Particle filtering with DRNNs (Hayashi et al. 2020).

   Hayashi et al. employ type-based reasoning with parameterisation. They encorporate the opponent types and parameters within types into the black-box environment simulator for the POMCP. They then select optimal actions via a MCTS planner at run-time. Their contribution, a deep-recurrent neural network architecture for particle reinvigoration, shows to perform well at updating despite an initially poorly configured model. At runtime, they use an unspecified heuristic for rollout policy.

Both Hayashi et al and Albrecht and Stone focus on inferring a correctly specified world model, and query this world model with a MCTS style algorithm. However, unlikey Barrett et al, they do not update their actions given this information, and rely solely on the MCTS to adapt, given the inclusion of the opponent model in the rollouts. Given that the existence of types imples an element of prior knowledge, it seems only reasonable to update actions, even if it is only to direct the rollouts, given this information, as opposed to an entire prior-learned policy.

Experiments

- Baseline — plastic policy
- Baseline — Monte Carlo Agent
- Baseline — Monte Carlo Particle Filter
- Show average performance against randomly selected opponents over 10000 rounds.
- Show
- Monte Carlo Policy — show robustness to reduced sample/rollout size.
- Show robustness compared to learning agent (in terms of learning)

[3]This is not an exhaustive list.

Evaluation/Thoughts

Future work & Conclusions

- Monte Carlo — supervised learning for quick learning and policy generation (Similar to monte-carlo A3C supervision...)
- Value theoretic function — learn more about opponents to better classify faster
- Strategy Generation — Monte Carlo agent for strategy imposing...

Closing comments

References

Albrecht, S. V., and Ramamoorthy, S. 2016. Exploiting causality for selective belief filtering in dynamic Bayesian networks. *Journal of Artificial Intelligence Research* 55:1135–1178.

Albrecht, S. V., and Ramamoorthy, S. 2019. Comparative Evaluation of Multiagent Learning Algorithms in a Diverse Set of Ad Hoc Team Problems.

Albrecht, S. V., and Stone, P. 2017. Reasoning about hypothetical agent behaviours and their parameters. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 1, 547–555. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS).

Albrecht, S. V., and Stone, P. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence* 258:66–95.

Barrett, S., and Stone, P. 2015. Cooperating with unknown teammates in complex domains: A robot soccer case study of ad hoc teamwork. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, 2010–2016. AI Access Foundation.

Barrett, S.; Rosenfeld, A.; Kraus, S.; and Stone, P. 2017. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence* 242:132–171.

Barrett, S.; Stone, P.; and Kraus, S. 2011. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*, 567–574.

Brown, N., and Sandholm, T. 2017. Safe and nested subgame solving for imperfect-information games.

Brown, N.; Bakhtin, A.; Lerer, A.; and Gong, Q. 2020. Combining deep reinforcement learning and search for imperfect-information games. In *34th Conference on Neural Information Processing Systems (NeurIPS)*.

Doucet, A., and Johansen, A. M. A Tutorial on Particle Filtering and Smoothing: Fifteen years later. Technical report.

Hayashi, A.; Ruiken, D.; Hasegawa, T.; and Goerick, C. 2020. Reasoning about uncertain parameters and agent behaviors through encoded experiences and belief planning. *Artificial Intelligence* 280:103228.

Hernandez-Leal, P.; Kaisers, M.; Baarslag, T.; and de Cote, E. 2017. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity. *CoRR* abs/1707.0.

Katt, S.; Oliehoek, F. A.; and Amato, C. 2017. Learning in POMDPs with monte carlo tree search. In *International Conference on Machine Learning*.

Leibo, J. Z.; Zambaldi, V.; Lanctot, M.; Marecki, J.; and Graepel, T. 2017. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 1, 464–473.

Lerer, A.; Hu, H.; Foerster, J.; and Brown, N. 2019. Improving policies via search in cooperative partially observable games.

Mealing, R., and Shapiro, J. 2017. Opponent Modeling by Expectation-Maximization and Sequence Prediction in Simplified Poker. *IEEE Trans. Comput. Intellig. and AI in Games* 9(1):11–24.

Panella, A., and Gmytrasiewicz, P. 2017. Interactive POMDPs with finite-state models of other agents. *Autonomous Agents and Multi-Agent Systems* 31(4):861–904.

Papoudakis, G.; Christianos, F.; Schäfer, L.; and Albrecht, S. V. 2020. Comparative Evaluation of Multi-Agent Deep Reinforcement Learning Algorithms.

Ross, S.; Pineau, J.; Chaib-Draa, B.; and Kreitmann, P. 2011. Bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research* 12:1729–1770.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*.

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Opponent Types description

Policy learning hyperparameters

Proof of convergence for MCTS and PF