Learning-augmented search for multi-agent environments

Charles Higgins

September 9, 2021

1 General context

It is a truth universally acknowledged that a machine-learning agent in possesion of suffient 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 or update their policies in response to stimulae is deemed non-stationary.

Despite prodigious advances in recent years in games posing challenges through enormous state-spaces, requiring strategic thinking and planning, mutli-agent systems pose a number of open questions, crucially around safe and trustworthy action selection. 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 reasobly act safely. In the pursuit of safe and trusted AI therefore methods to address this non-stationarity are required.

One assumption one can reasonably make is that the non-stationary element of the environment is due (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 asumption made by a group of methods which are loosely referred to as opponent modelling.

Opponent modelling, particularly within the area of game-playing AI, has had notable successes in closed environments, however open environments still remain an open question (pun intended) [1]. 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, obseravtions, and likely actions) and, equally swiflty, use this information to act safely, and ideally, optimally.

To meet this objective: a trustworthy method of action selection which is swift and robust, we propose a mutli-agent extension of policy-augented partially observable monte-carlo planning agent, with an opponent-weighted policy-augmented search component. This architecture 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 which proves to outperform both in isolation, and other state of the art baseline methods, given an uncertain environment.

The remainder of this paper is structured as follows: Section 2 introduces the background theory for opponent modelling, data-driven action-selection in game-playing AI and reasoning approaches. Section 3 formalises the system. Section introduces our contribution Section briefly describes the

simulation environment used to extract results and the heuristic opponent agents Section lists the upcoming experiments and expected results, Section concludes.

2 Background

This section provides an introduction to various basic concepts and intuitions required to understand the contex 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.

2.1 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 is not theoretically posisble. This however poses a problem as in reality, most environments exhibit a degree of non-stationarity. Indeed, a great 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 coridoor.

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.

Indeed even 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 sucessive state is probabilistic), the optimal policy should adjust, and will remain fixed. Mathematically speaking, this environment exhibits the markov-property, where firstly, there exists an optimal policy, and secondly, the optimal policy is inferrable from only the current state (i.e. memoryless).

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 envinronment is still stationary, however if that other agent learns, or updates it's policy, or enters and leaves, the environment is fundamentally changing.

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 undertand the actions of another agent in order to preempt, or truly understand the likely resulting state. This idea fundamentally underpins opponenet modelling — containing an element of non-stationarity and uncertainty within another agent model, and constructing and using these models in order to reduce uncertaintly and ultimately act optimally.

2.2 Methods of opponent modelling

There are two distinct parts of opponent modelling — the first involves constructing an opponent model for a specific opponent (or set of opponents) and 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 differnet environments and situations, some are better suited 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 unlikly to succed.

A common and far swifter form of opponent modelling is type-based reasoning. 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.

Both of the previous models could be subsymbolic (i.e. no real reasoning capability). In effect, one could view an agent model as a black-box of sorts, and simply apply it to certain situations. The highest level 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. This tends to be computationally draining and fails to be of practical use in scenarios with limited interaction protocls.

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

TODO: CITE A FEW PAPERS AND DESCRIBE THEM AND DESCRIBE AD-HOC COORDINATION SCENARIOS IN GENERAL. NOT MORE THAN A PARAGRAPH

2.3 Action-Selection

In general terms there are two broad approaches to action-selection for autonomous agents: a learning vs a reasoning approach. The former starts with limited knowledge about the environment, and through sucessive trials and learning from actions and rewards, decides to learn a policy. The latter typically requies more domain knowledge, and reasons about actions before they are made in order to meet some objective/reach a desired state.

2.4 Learning vs Reasoning (high level)

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. In general, data-driven approaches can perform well, given sufficent 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 impliment, and given a stationary environment, can learn an optimal policy to maximise rewards.

In contrast, the reasoning approach requires very few, if any, learnig iterations before performing well. Rather than learning from scratch, reasoning agents typically have a model of the environment (hence the nominer model-based) which they query, testing out the likely effect of each possible action, and subsequent action. 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 percieved 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 intial 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 dymanics. 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.

2.5 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 [13, 5, 10]. The major difficulties of these domains (Chess, Go, even Poker) is that of a truly enourmous state-space. Research which has acheived noteriety has combined local state-space search with deep neural networks to approximate complex functions.

While each of the mentioned papers 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 intractible. To address the issue of an enourmous 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 seach 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 netowrks) 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/unfavourable). 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 [13], 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. [5] 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.

Crucially, this research approaches the problem of large state-spaces rather more from the side of learning as opposed to reasoning. In contrast, a body of work has tackled partially observable environments with an explicit focus on reasoning: planning.

2.6 (Sample-based) Planning

Planning is, in essence, a search problem — an agent queries an internal model which captures the dynamics of the environment to find a sequence of actions to achieve some goal state or maximise a reward. In complex domains, the search-space swiftly becomes so large as to be computationally intractable, hence the recent focus on model-free forms of learning to solve similar problems. However, model-free approaches tend to be unpredictable. As they have no model to reason about the environment, they can exhibit incorrect and even dangerous behaviour in unforeseen scenarios — typically there is no reasoning process, but simply a mapping of a state to an action. Given this unpredictability, the application of purely model-free approaches to some domains carries risk and is therefore undesirable. Extending planning to complex domains remains an open question and a valuable and viable direction of enquiry. At present, the most promising area appears to be sample-based planning, which requires exploring only some of the search space online (i.e. at runtime).

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

Monte Carlo Tree Search is a sample-based search algorithm which requires a simulation model of the environment to evaluate possible actions and future states in order to converge to an optimal policy online (i.e. re-evaluating from each state). Each node represents a possible state, and transitions (edges) are possible (probabilistic) actions. At a given state, an agent traverses a computed tree, selecting actions with a heuristic until reaching a leaf node. At the leaf node, it then randomly selects actions and receives a potential successor state from the transition model. This is referred to as a rollout, and continues to some arbitrary depth of tree depending on available resources. It then evaluates the state and the rewards achieved in the leaf node, and propagates the rewards backwards up the tree. This cycle (traversal, simulation (rollout) and backpropagation) then continues for a determined period, until a new action is selected, and a new state observed.

This sampling-based approach is typically paired with a UCB1 algorithm (upper confidence bounds) which balances exploitable actions with under-explored alternatives, which has shown to converge (eventually) to an optimal policy [12].

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 the previous Section: Section 2.5)

2.6.2 Monte Carlo Search in a partially observable environment — adopting MCTS 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 casuses 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. 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 [14] 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 effect, Monte Carlo methods are used in a bi-directional manner: forwards to evaluate action selection; backwards to update belief over states.

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 likelhihoods) 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. 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.

2.6.3 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 [9] 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 incorportated 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 [7] augmented a POMCP planner with a deep-recurrent neural network as a mechanism for particle reinvigoration (suggesting 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 [11, 4], in which agent types were parameters within the black-box simulator.

2.6.4 Improving MCTS — information theoretic rewards

Finally, Fischer and Tas augmented a MCTS style of planning with information theoretic rewards in a continuous domain. This means in practice increasing an internal reward for reaching states where the agent received more information about the environment. The basic idea behind this is that often as an agent's understanding of a domain increases, the convergence to optimal action selection is swifter. While this was tested in a single-agent domain, the concept of augmenting a sampling-based search procedure with information-based rewards showed promise [6]. Despite the domain being marginally different from that of opponent modelling, the concept of explicitly rewarding an agent for looking for information has parallels in opponent exploration. Ultimately, the concept of deliberately manipulating a game state to learn more about an opponent is worthy of investigation, and seems at present somewhat under-explored.

2.6.5 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 deapth, 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 actions 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. In both [7] and [1] 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 distict agent models and environment models.

This method also has benefits as it translates the problem of opponent modelling into one of state-space search, allowing for the incorporation of policy/value netowrks and/or sampling-based planners to address complexities of opponent modelling, which often result in large (probabilistic) state-spaces.

3 Environment Abstraction: TODO: UPDATE AND DEVELOP THIS WEEK

3.1 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,

$Z = Pr(o_{t+1}|s_{t+1}, a_t)$ is the stochastic observation function;

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

$\gamma \in [0, 1]$ is the discount factor;

b_0 is the belief about the initial state.

At any time t, an agent takes an action (a_t), receives an observation (o_t) and reward (r_t). 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 is to minimise or eliminate the mismatch between the agent's belief/encoded model of the environment and the dynamics of the real environment (i.e. the ground truth). This model of the environment is expressed (in this framework) by the observation (imperfect observability) and transition functions.

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 the model dictates the quality of the state-transition and observation functions, and therefore the quality of the overall policy derived.

One can view Θ as an encapsulated opponent 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 observations and action pairs ($h_t = [a_0, o_0, a_1, o_1 ... a_{t-1}, o_{t-1}]$) taken to reach the state s_t.

3.2 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.

The definition of Θ, and methods to accurately estimate Θ, where theta refers to an number of opponent models in an open environment is ultimately the goal of the PhD.

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

3.3 Individual agent models

Each $\theta_i \in \Theta$ is an agent present in the environment.

3.3.1 Type-based reasoning

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.

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

3.3.2 Policy reconstruction

Given prolonged or repeated interactions, it stands to reason that some types, if ill-fitting, could be updated by way of hyper-parameters. For instance, take a basic Q-learning agent, a key hyper-parameter might refer to the learning rate, which would in turn affect the fitting of the relevant agent model. Hence, in an approach similar to Albrecht and Stone [1], one can encapsulate parameters *within* opponent types. Each opponent is represented as a tuple:

$$\theta = \langle \phi^*, \pi, \rho \rangle \tag{6}$$

Where ϕ represents the model type, π represents a list containing the relevant parameter values, and rho is a dictionary in which each entry relates a parameter value index to the possible parameter range. Each entry will either contain a list containing discrete values, or a range indicating the scope of continuous variables.

4 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.

4.1 The level-based foraging domain

A plausible environment (among a number of others) are variations of the Level-based foraging domain (LBFD) (used by [3, 4, 7, 11][1]). 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.

4.2 Baselines in ad-hoc coordination

1. Plastic Policy [4]

   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[1].

   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 [7].

   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

[1] This is not an exhaustive list.

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.

5 Shortcomings of related/current work

5.1 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.

5.2 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. [7], 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 [2, 8]. 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.

6 Proposal

The proposal is to build upon a basic POMCP architecture with three major additions:

- Type-based reasoning to include prior knowledge of agent types. A particle-filtering mechanism can be used to reason about agent parameters in a dynamic and partially observable environment.
- An agent-type specific policy network which guides the search samples. In effect re-using the belief over agent types computed within the particle-filtering component.
- An information theoretic value-function in early stages of interaction to swiftly identify another agent. This could be implemented via an updated value-function, rewarding actions and states which inspire maximal type-divergence (states where agent types act notably differently).

The major novel contributions of this work would be:

- Extending RL + Search to a general sum, n-player environment.
- Extending RL + Search to an explicit opponent modelling environment.
- Extending belief-based planers with reinforcement learning.
- Extending a RL+Search architectures with an information theoretic value function.

References

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

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

[3] Stefano V Albrecht and Subramanian Ramamoorthy. Comparative Evaluation of Multiagent Learning Algorithms in a Diverse Set of Ad Hoc Team Problems, 2019.

[4] S Barrett and P Stone. 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, pages 2010–2016. AI Access Foundation, jun 2015.

[5] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[6] Johannes Fischer and Ömer Sahin Tas. Information Particle Filter Tree: An Online Algorithm for POMDPs with Belief-Based Rewards on Continuous Domains. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 3177–3187, 2020.

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

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

[9] Sammie Katt, Frans A Oliehoek, and Christopher Amato. Learning in POMDPs with monte carlo tree search. In *International Conference on Machine Learning*, 2017.

[10] Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games, 2019.

[11] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative Evaluation of Multi-Agent Deep Reinforcement Learning Algorithms, 2020.

[12] Stéphane Ross, Joelle Pineau, Brahim Chaib-Draa, and Pierre Kreitmann. Bayesian approach for learning and planning in partially observable markov decision processes. *Journal of Machine Learning Research*, 12:1729–1770, 2011.

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

[14] David Silver and Joel Veness. 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*, 2010.