Learning-augmented search for multi-agent environments

Charles Higgins

May 28, 2021

1 General context

- Opponent modelling roughly aims to compute a model of other agents in order to optimize action selection.
- While a well developed field, methods have yet to extend to open environments.
- For an open environment, methods need to be swift, scalable and robust.
- This document looks at the swift component — in short, RL agents are swift to execute, however take a while to train and can be unpredictable when dynamics shift drastically from training. Model-based reasoning approaches (planning) tends to be safer and predictable, but slow to execute.
- This document briefly discusses the state of the art in combining search and learning, and identifies gaps in the literature.

2 Background

There are two main paradigms which combine learning and reasoning in games — 1) sample-based planning and 2) RL + Search.

2.1 (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.

2.1.1 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 to some arbitrary depth. 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.

2.1.2 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. The reason being is that 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 [12] 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, reflects the 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.

2.1.3 Belief-based planning with MCTS — Improving on POMCP

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, even with the relatively relaxed requirements as to the nature of the transition model (all that is required is a black-box simulator). Simply put, if the model 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.

Hayashi et al [6] 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 as a black-box simulator, with various parameters to be fitted. This fitting took place online, 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 [10, 3], in which agent types were parameters within the black-box simulator.

With a similar aim, Katt et al [8] 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, allowing for better plans to be made.

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 [5]. 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.

2.2 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 [11, 4, 9]. These researchers approach the problem of large state-spaces rather more from the side of learning rather than reasoning.

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

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 an value 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 [11], 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. [4] 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.

3 Shortcomings of related/current work

3.1 Shortcomings of RL + Search

The majority of work in the RL + Search space is focused on 2-player zero-sum games. This allows for (implicit) opponent modelling via the use of a policy function to guide search. This need not be explicit in this context: players' intentions are symmetric — the reward function for a player's opponent is the inverse of the players, 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, arguably more explicit and granular representations of a varying/heterogenous opponent types are necessary 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, but 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.

3.2 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. [6], 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 [1, 7]. 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.

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, 10, 2, 6][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.

5 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.

[1] This is not an exhaustive list.

- 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. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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.

[12] 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.