Higher-Order Reasoning under Intent Uncertainty Reinforces the Hobbesian Trap

Otto Kuusela
University of Amsterdam
Amsterdam, The Netherlands
otto.kuusela@outlook.com

Debraj Roy
University of Amsterdam
Amsterdam, The Netherlands
d.roy@uva.nl

A SUPPLEMENTARY METHODS

In this section we provide supplementary information on the methods we used. In Section A.1 we describe more in depth the context which our model aims to capture, and in Section A.2 we review the definition of Interactive POMDPs (I-POMDPs).

A.1 Modelling Context of Interstellar Conflict

The goal is to model the behaviour of *civilisations* in the universe. By a civilisation we mean a group of one or more organisms that collectively control their own local region of the universe. This region could be a planet or a star system, for example. The civilisations of interest here develop technologies for observing and weapons for attacking other civilisations. Civilisations build observatories to listen to technosignatures – signs of technologically advanced civilisations – and use these to form their beliefs about the capabilities of other civilisations.

The model includes many such civilisations. We assume they are located together in a relatively small region of space (on the order of tens to hundreds of light-years) unaffected by the expansion of the universe. An example is a part of a galaxy such as the Milky Way. Here we make the simplifying assumption that civilisations do not move with respect to one another. This includes the assumption that they do not attempt to invade the home worlds of other civilisations. The multi-year delays between actions and their effects (imposed by the speed of light and long distances) are assumed to be negligible, for simplicity.

A fundamental challenge in modelling non-human civilisations is that we do not know how they behave. Here we should distinguish between *preferences* (what the civilisations wish would happen) and the *capability* to shape the world according to these preferences. An example of preferences is the willingness of the civilisations to use violence to protect themselves.

We assume that the most important goal of all civilisations is to ensure their survival. Such an assumption seems reasonable because of natural selection: organisms tend to value their own existence because the ones who did are the ones who survived. Other preferences are allowed to vary between civilisations. We further assume that the decision-making capabilities of civilisations are maximal. In other words, we assume civilisations behave *rationally* and therefore attempt to maximise their expected future rewards. These rewards reflect adherence to their preferences. The assumption of rationality seems reasonable, since we are concerned with civilisations that grow to planetary scales, controlling their own environments and developing advanced technologies. It is notable here that we only require the civilisation to act rationally as a whole; the individual organisms do not need to be rational. For example, during the Cold War the nuclear powers were approximately rational by thoroughly analysing possible courses of action [8].

To approximate this rational behaviour, we use the interactive partially observable Markov decision process (I-POMDP) framework. Each civilisation independently applies the framework to make decisions.

A.2 Interactive POMDPs

In an interactive POMDP (I-POMDP), the simple state space of MDPs and POMDPs is extended to include both the states and models of other agents [4]. Let $N = \{1, \dots, n\}$ be the set of agents and S the set of (environment) states. A finitely nested I-POMDP for agent i at level L is a tuple $(IS_{i,L}, A, O_i, T_i, Z_i, R_i, C_i)$ [4]. Here $A = A_1 \times \cdots \times A_n$ is a set of possible joint actions by all agents, where A_j is the set of possible actions of agent j. The transition function $T_i \colon S \times A \times S \to \mathbb{R}_{\geq 0}$ defines a probability distribution $T_i(s, a, \cdot)$ over the state of the environment at the next step given the current state s and joint action $a = (a_1, \dots, a_n)$. The observation function $Z_i : S \times A \times O_i \to \mathbb{R}_{\geq 0}$ defines a probability distribution $Z_i(s', a, \cdot)$ over the observation space O_i given the current state s' and previous joint action a. $R_i \colon S \times A \to \mathbb{R}$ is the reward function, which gives the expected reward for taking an action in a given state. Finally, the optimality criterion C_i specifies how agent i values rewards obtained at different time steps. An example is the optimality criterion introduced in Section 3.1.6 of the paper.

The most interesting part in this definition is the level L interactive state space

$$IS_{i,L} = S \times \left(\bigtimes_{k \in N \setminus \{i\}} \bigcup_{l=0}^{L-1} M_{k,l} \right) \tag{1}$$

of agent i. If $L = 0$, we define $IS_{i,0} = S$. Here S is the set of possible environment states and $M_{k,l}$ is the set of models for agent k at level l. Models contain a function mapping an agent's history of observations to a distribution over its actions A_k. While we will not characterise this set of models completely, we highlight an interesting subset. The set $\Theta_{k,l} \subset M_{k,l}$ contains the possible types, or *intentional models*, of agent k at level l. A *type* $\theta_{k,l} \in \Theta_{k,l}$ of agent k encapsulates all of the private information used by a rational agent k to make decisions. It consists of two parts: beliefs $b_{k,l}$ over the level l interactive states of agent k, and the *frame* $\hat{\theta}_k$ which contains the necessary components to define its model of the decision process. More formally, a frame is a tuple $(A, O_k, T_k, Z_k, R_k, C_k)$. All the components are defined like before. A type is then denoted $\theta_{k,l} = (b_{k,l}, \hat{\theta}_k)$. In addition to intentional models, another possible model is a simple no-information model where the agent chooses

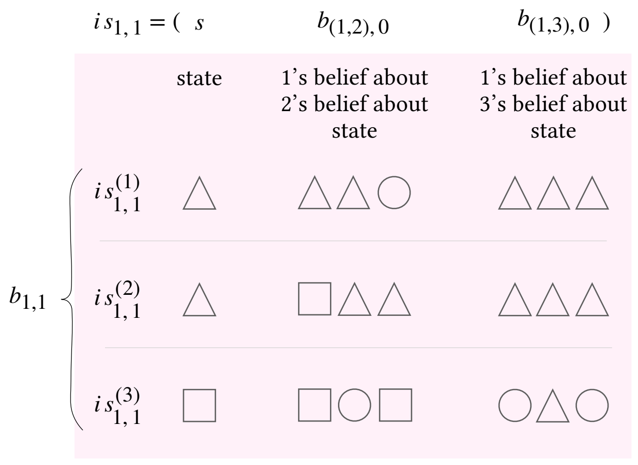
$is_{1,1} = ($ s $b_{(1,2),0}$ $b_{(1,3),0}$ $)$
state
1's belief about 2's belief about state
1's belief about 3's belief about state
$is^{(1)}_{1,1}$
$is^{(2)}_{1,1}$
$is^{(3)}_{1,1}$
$b_{1,1}$

Figure 1: An example of interactive states at level 1 in an I-POMDP using intentional models on all levels. Each row is a level 1 interactive state of agent 1. Such an interactive state consists of a state and a level 0 belief about every other agent's beliefs regarding the state. The level 0 beliefs are represented by an unweighted sample of three states. Together, the three level 1 interactive states form agent 1's level 1 belief $b_{1,1}$

actions randomly from its set A_k of possible actions. We will be using intentional models to model other agents; the exception is level 0, where we assume a no-information model.

In this work we make two simplifying assumptions to the definition of an I-POMDP. First, we use a simplified definition of the interactive state space. Instead of (1) we use

$$IS_{i,L} = S \times \left(\bigtimes_{k \in N \setminus \{i\}} M_{k,L-1} \right). \tag{2}$$

In other words, instead of considering all possible levels of models of other agents, we only expect other agents to reason at a level one lower than us. This assumption follows that made in [11] and [2] and simplifies the work of the planning algorithm. The second assumption we make is related to the frames $\hat{\theta}_k$ of intentional models. In most of our experiments we assume that the frames of all civilisations are common knowledge to everyone. This means that civilisations know, for example, how others observe and receive rewards. What is more, we assume that all civilisations use the same transition function, denoted T. These assumptions allow us to identify a type $\theta_{k,l}$ with the beliefs $b_{k,l}$ over the level l interactive states of agent k. We occasionally use the notation $b_{(\cdot,k),l}$ (where $\cdot$ corresponds to a sequence of agents) to denote $b_{k,l}$ to emphasise who holds the belief about agent k's beliefs. For example, $b_{(1,2),1}$ denotes the level 1 beliefs of agent 2 as believed by agent 1. In later experiments we relax this assumption and allow civilisations to have uncertainty about others' frames.

Let us try to gain an intuitive understanding of the (simplified) interactive state space through an example. Here we are using intentional models on all levels. Let us assume that the state space is $S = \{\square, \triangle, \circ\}$. There are three agents, and we are reasoning from agent 1's perspective. This situation is illustrated in Figure 1. Beliefs are represented by unweighted samples of size three.

On the lowest level $IS_{i,0} = S$. In other words, level 0 interactive states are simply (environment) states; this corresponds to a POMDP as the agent does not have a model for the behaviour of other agents. In our example, level 0 interactive states are thus shapes and beliefs over level 0 interactive states are probability distributions over the possible shapes.

At level 1, $IS_{i,1} = S \times \left(\bigtimes_{k \in N \setminus \{i\}} \Theta_{k,0} \right)$. So a level 1 interactive state consists of an environment state and a model (type) for the behaviour of each other agent at level 0. Since we can omit the frames, a level 1 interactive state contains a state and a distribution over states (i.e. a belief over level 0 interactive states) for each other agent. Figure 1 illustrates level 1 interactive states. Each such interactive state consists of a state (a shape) and corresponding beliefs about agent 2 and 3's beliefs about the shape. Agent 1 thinks 2 has a reasonably accurate sensing capabilities, getting the shape right roughly 2/3 of the time. Agent 1 also thinks that while agent 3 can observe triangles and circles perfectly, it cannot detect squares and will in such a case believe the shape is either a circle or triangle with equal probability. The three level 1 interactive states together form agent 1's level 1 beliefs.

At level 2, each interactive state specifies a state and a belief over level 1 interactive states. In our example, this could be visualised as each row being a level 2 interactive state and each symbol in the two middle columns having corresponding level 0 beliefs for the other agents (agents 1 and 3 in the middle column, agents 1 and 2 in the rightmost column).

B PLANNING IN I-POMDPS

Our aim in this section is to introduce the theory of solutions to I-POMDPs and then discuss current algorithms designed for this purpose.

B.1 Solutions to I-POMDPs

Let us elaborate on what behaving optimally in an I-POMDP of agent i means. The goal is to find an optimal *policy* $\pi^* : \Delta(IS_{i,L}) \to \Delta(A_i)$ mapping beliefs over i's level L interactive states into a distribution over its actions. Since solving an I-POMDP at a nesting level $L > 0$ involves solving nested I-POMDPs modelling other agents' decision-making, we will discuss finding the optimal policy for a belief $b_{j,l}$ of agent $j \in N$ at level $l \leq L$.

The goal of agent j is to maximise the expected utility (1) (in the paper). To do this, it may consider the expected utility $U(b_{j,l}, a_j)$ of taking each action $a_j \in A_j$ in belief state $b_{j,l}$ and then continuing optimally. If it knows these expected utilities, it can simply choose the action with the highest expected utility; in other words, the set of optimal actions for agent j under this belief is $\text{Opt}(b_{j,l}) = \arg\max_{a_j \in A_j} U(b_{j,l}, a_j)$. The optimal policy $\pi^*(b_{j,l})$ assigns a positive probability exactly to the actions in this set. Additionally, we define the expected utility of the belief state $b_{j,l}$ as $U(b_{j,l}) = \max_{a_j \in A_j} U(b_{j,l}, a_j)$. The expected action utility of a_j

can be expressed as [2]

$$U(b_{j,l}, a_j) = \int_{IS_{j,l}} b_{j,l}(is_{j,l}) \mathbb{E}R_j(is_{j,l}, a_j) \, \mathrm{d}is_{j,l} + \gamma \int_{O_j} p(o_j \mid a_j, b_{j,l}) U(\mathrm{Forward}(b_{j,l}, a_j, o_j)) \, \mathrm{d}o_j. \quad (3)$$

To clarify the meaning of the terms in this expression, let us first note that the probability that action a_k is optimal for agent $k \neq j$ (according to agent j) is given by $\pi^*(b_{(j,k),l-1})(a_k) = 1/\left|\mathrm{Opt}(b_{(j,k),l-1})\right|$ if $a_k \in \mathrm{Opt}(b_{(j,k),l-1})$ and 0 otherwise. In other words, if there are multiple equally good actions, we assume other agents randomly choose between them. Then, the probability that the actions $a_{\sim j} \in A_{\sim j} = A_1 \times \cdots \times A_{j-1} \times A_{j+1} \times \cdots \times A_n$ are optimal for each other agent individually is given by

$$\mathbb{P}(a_{\sim j} \mid is_{j,l}) = \prod_{k \in N \setminus \{j\}} \pi^*(b_{(j,k),l-1})(a_k). \quad (4)$$

Now the expected immediate reward of taking action a_j in interactive state $is_{j,l}$ (in the first term of equation (3)) is given by

$$\mathbb{E}R_j(is_{j,l}, a_j) = \sum_{a_{\sim j} \in A_{\sim j}} \mathbb{P}(a_{\sim j} \mid is_{j,l}) R_j(s, a) \quad (5)$$

where we denote $a = (a_{\sim j}, a_j) = (a_1, \ldots, a_n)$.

In the second term of equation (3) the expression

$$p(o_j \mid a_j, b_{j,l}) = \int_{IS_{j,l}} b_{j,l}(is_{j,l}) \left(\sum_{a_{\sim j} \in A_{\sim j}} \mathbb{P}(a_{\sim j} \mid is_{j,l}) \int_S T(s, a, s') Z_j(s', a, o_j) \, \mathrm{d}s' \right) \mathrm{d}is_{j,l} \quad (6)$$

is the probability (density) of observing o_j given the taken action by agent j and the current belief. In addition, $\mathrm{Forward}(b_{j,l}, a_j, o_j)$ is the belief update operation. The belief update is at its essence an application of Bayes' theorem. The updated probability (density) for $is'_{j,l} = (s', (b'_{(j,k),l-1})_{k \in N \setminus \{j\}}) \in IS_{j,l}$ is given by

$$\mathrm{Forward}(b_{j,l}, a_j, o_j)(is'_{j,l}) \propto \int_{IS_{j,l}} b_{j,l}(is_{j,l}) \sum_{a_{\sim j} \in A_{\sim j}} \mathbb{P}(a_{\sim j} \mid is_{j,l}) T(s, a, s') Z_j(s', a, o_j) \times \left(\prod_{k \in N \setminus \{j\}} \int_{O_k} \delta_D(\mathrm{Forward}(b_{(j,k),l-1}, a_k, o_k) - b'_{(j,k),l-1}) Z_k(s', a, o_k) \, \mathrm{d}o_k \right) \mathrm{d}is_{j,l}. \quad (7)$$

The term in the parentheses makes sure that a positive probability density is assigned only to interactive states $is'_{j,l}$ where agent j's beliefs of other agents' beliefs are appropriately updated. Since in an I-POMDP can include nested I-POMDPs modelling other agents' decision making, this will trigger a belief update in these lower level models as well. The symbol δ_D stands for the Dirac delta.

B.2 Algorithms for solving I-POMDPs

Many algorithms have been proposed for solving I-POMDPs. They can be divided into two categories: *offline* and *online* algorithms. Offline algorithms solve the full optimal policy π^* for all belief states (or a sampled subset) beforehand whereas online algorithms only solve the optimal action for the current belief at each time step.

Algorithms for solving I-POMDPs face three challenges [2]. The *curse of dimensionality* refers to the fact that the belief space in which the problem is solved is high-dimensional. In I-POMDPs this problem is even worse than in POMDPs, as the state space includes models of other agents which may be (I-) POMDPs themselves. This is sometimes referred to as the *curse of nested reasoning* [6]. Finally, as the time horizon considered gets longer, the space of possible policies grows exponentially (*curse of history*).

We will now introduce some of the most notable algorithms developed for solving I-POMDPs. As many of the algorithms are inspired by conceptually similar algorithms for POMDPs, we will also discuss some algorithms designed for them.

B.2.1 Value Iteration. Perhaps the most conceptually straightforward way to solve an I-POMDP is to use *value iteration.* The method is based on the fact that the expected utilities $U(\cdot) = \max_{a_j \in A_j} U(\cdot, a_j)$ of different beliefs are related to each other through equation (3). The idea is to construct a sequence of expected utility functions that converge to the optimal expected utility function. This optimal expected utility function corresponds to the optimal policy. The sequence $(U_t)_{t \in \mathbb{N}_{\geq 0}}$ starts with $U_0(b_{j,l}) = 0$ for all $b_{j,l} \in \Delta(IS_{j,l})$ and is then defined recursively by

$$U_t(b_{j,l}) = \max_{a_j \in A_j} \left(\int_{IS_{j,l}} b_{j,l}(is_{j,l}) \mathbb{E}R_j(is_{j,l}, a_j) \, \mathrm{d}is_{j,l} + \gamma \int_{O_j} p(o_j \mid a_j, b_{j,l}) U_{t-1}(\mathrm{Forward}(b_{j,l}, a_j, o_j)) \, \mathrm{d}o_j \right) \quad (8)$$

for all $t \in \mathbb{N}_{\geq 1}$. Here U_t can be thought of as the expected utility function for a finite horizon of length t [1]. In essence, the values U_{t-1} are used to compute the expected action utilities in equation (3) and U_{t+1} is set to the maximum expected action utility over the possible actions. The authors in [4] show that the sequence converges to a unique fixed point U^* corresponding to the optimal policy. The actual policy can be determined using the values of U^* in equation (3).

Naturally, computing the optimal policy in this way is not feasible because the belief space $\Delta(IS_{j,l})$ is continuous. However, the expected utility function $U_t(\cdot)$ is piecewise linear and convex for any $t \in \mathbb{N}_0$ [4]. This allows writing the value function as a linear combination

$$U_t(b_{j,l}) = \max_{\alpha_t \in \mathcal{V}_t} \sum_{is_{j,l} \in IS_{j,l}} b_{j,l}(is_{j,l}) \alpha_t(is_{j,l}). \quad (9)$$

Here $\mathcal{V}_t$ is the set of *alpha vectors* α_t. Each alpha vector corresponds to a *conditional plan* for a horizon of length t. A conditional plan specifies an action and a subsequent conditional plan for each possible observation received after taking the action [1]. The functions α_t map an interactive state to the expected utility of the corresponding conditional plan when starting from the interactive state.

Decomposing the expected utility function in this way means that the utility for any belief can be expressed using a finite set of alpha vectors. While this is helpful, the alpha vectors are still functions over an interactive state space that is continuous in general. In generalised point-based value iteration [3] the interactive state space is restricted to a finite set of environment states and a finite set of models (beliefs) for other agents. Another issue with the alpha vector decomposition is that in order to express the expected utility function exactly, the size of the set $\mathcal{V}_t$ of alpha vectors may be prohibitively large [12]. Inspired by point-based value iteration for POMDPs [10], [3] considers a fixed set of points in the belief space. This set consists of an initial finite set of beliefs and all beliefs that can be reached from these beliefs in a given number of time steps. The set $\mathcal{V}_t$ is limited such that each alpha vector is required to be optimal in at least one of the points in the fixed set of beliefs. This means that the size of the set of alpha vectors is upper bounded by the set of belief points considered. Point-based methods thus limit the curse of history by limiting the number of alpha vectors (and thus conditional plans) considered. Generalised point based value iteration still suffers from the curse of dimensionality. This is because the fixed set of beliefs considered grows exponentially with the number of time steps considered. In a way, the algorithm converts the curse of history into the curse of dimensionality.

B.2.2 *Interactive Particle Filter.*

The *interactive particle filter* (I-PF) [2] focuses on mitigating the curse of dimensionality by representing beliefs over interactive states with a finite set of *particles*. Each particle corresponds to a particular interactive state. Beliefs about others' beliefs are similarly represented by a set of particles within the particle. This is similar to Figure 1. A variant of the value iteration algorithm is employed: it builds a finite horizon look-ahead tree starting from a given belief to determine approximately optimal actions to take from the belief. We will not describe the value iteration algorithm here further.

A *particle filter* is used to maintain an up to date belief about the state. Since our algorithm also employs a particle filter, we will explain the intuition behind the technique. We will first explain it from the point of view of a POMDP and then discuss the application to I-POMDPs.

The set $\tilde{B}$ of particles approximates the current belief b. Specifically, given a corresponding vector w of (possibly uniform) weights, b is approximated by $\tilde{b}(s) = \sum_{\tilde{s}\in\tilde{B}} w(\tilde{s})\delta_D(s - \tilde{s})$, where δ_D is the Dirac delta. After taking an action a and receiving an observation o', the agent's belief needs to be updated in an attempt to estimate the new true state. This means that we seek a new set $\tilde{B}'$ and weights w' approximating the exact updated belief b'. A particle filter does this in two stages: *importance sampling* and *resampling*.

Importance sampling allows sampling from the unknown distribution b' by sampling from a proposal distribution and then weighting the samples appropriately. This is possible as long as the proposal distribution assigns a non-zero probability (density) to each state where the target probability (density) is positive. The sample from the proposal distribution is created by sampling a state $s(p') \sim T(s(p), a, \cdot)$ from the distribution defined by the transition function for each particle. Here $s(\cdot)$ denotes the state stored in the given particle. Each particle in the resulting set is weighted by the likelihood $Z(s(p'), a, o')$ of observing o' assuming the particle's state is the true state. The weights are then normalised to give a weighted approximation to the updated belief.

When applied multiple times in succession, the weights of the particles tend to have a high variance: most particles get a near-zero weight while most of the probability is on only a few particles [9]. A resampling step attempts to fix this by sampling (with replacement) from the set of particles using the generated weights. This means that the frequency of particles with high weight is increased while some particles with low weights might be completely cut out. The overall effect is to focus the set of particles on the most likely states. This sample – which now receives uniform weights – represents the updated belief $\tilde{b}'$.

In an I-POMDP, a joint action is needed to propagate a particle $p \in \tilde{b}_{j,l}$; therefore the beliefs $\tilde{b}_{(j,k),l-1}(p)$, $k \in N \setminus \{j\}$ (represented by sets of particles) over others' lower level interactive states are used to solve their optimal actions. Particle p is then propagated with the joint action like described above. In addition to updating the environment state, other agents' beliefs stored in the particle also need to be updated. This is done for each possible observation of each other agent[1]. These lower-level beliefs are again updated using an I-PF belief update. In other words, the belief update recursively traverses down the belief hierarchy in the particle until at level 0 a normal POMDP-based belief update is performed. After the beliefs of other agents are updated, the particle is weighted with the likelihood of both the agent itself observing its observation and each other agent observing their respective observations. The resulting set of particles, of size $|\tilde{b}_{j,l}| \prod_{k\in N\setminus\{j\}} |O_k|$, is then resampled down to the size $|\tilde{b}_{j,l}|$ of the original set of particles.

The number of particles needed to accurately represent a belief increases exponentially with the number of dimensions of the belief space [2]. This means that the particle filtering method cannot completely defeat the curse of dimensionality. However, it does help in focusing the computation of optimal actions to the relevant parts of the belief space. The I-PF does not address the curse of history.

B.2.3 *I-POMDP Lite.*

I-POMDP Lite [6] circumvents the three curses of planning in I-POMDPs by restructuring the problem. In an I-POMDP of agent j, other agents' decision making is typically modelled with nested I-POMDPs; each state has an associated I-POMDP model for every other agent. In I-POMDP Lite, the authors assume instead that other agents are modelled as *nested Markov Decision Processes* (nested MDPs). The I-POMDP Lite framework is therefore essentially a POMDP with the added information of others' policies, solved from the nested MDP.

Using a nested MDP as a model for other agents' behaviour corresponds to the assumption that each other agent $k \in N \setminus \{j\}$ has full observability of the current state and that k assumes that other agents can also perfectly observe the state. In addition to being easier to solve, the nested MDP assumption means that the state space does not need to be interactive and contain models for other agents. The reason is that the nested MDP policy is already a mapping from environment states to distributions over actions, so it can be used as a model for all states.

[1]The authors also describe a variant suitable for large or continuous observation spaces, where an observation is instead sampled for each particle.

The authors show that the performance of a policy computed with I-POMDP Lite significantly outperforms I-POMDP policies computed using I-PBVI (Section B.2.1). The performance difference is greater if the agents have better observation capabilities. This highlights an important downside to I-POMDP Lite: it is best suited to situations in which agents' observation capabilities are reasonably accurate. Assuming that other agents observe perfectly might not be a good model in some situations. For example, if observing other civilisations' strengths is difficult even for strong civilisations, their behaviour might be markedly different compared to if they had perfect information regarding how strong they are compared to others.

B.2.4 Policy Iteration. Thus far all the algorithms discussed use the expected utility function in one way or another to compute optimal actions. In policy iteration, the search happens instead in the space of policies [15]. The advantage of working with policies directly is that the policy might be optimal before the expected utility function itself converges. Intuitively this is because knowing which action is best is easier than knowing the exact expected utilities of taking different actions.

In Interactive Bounded Policy Iteration (I-BPI) [15], the candidate policies are expressed as *finite state controllers*. In this context, a finite state controller π consists of a set of nodes $\mathcal{N}$ each of which corresponds to a distribution over actions. Edges $\mathcal{E}$ between nodes are labelled with observations. The transition function $\mathcal{T}$ gives the probability of following each edge to a next node, given the action taken and observation received. Each node is associated with a vector of values for each possible interactive state. A value represents the expected utility of taking the action prescribed by the node and then continuing according to the finite state automaton. These vectors correspond to the alpha vectors in value iteration.

The agent's own policy is modelled as such a finite state controller. What is more, the policies of the other agents are also modelled as finite state controllers (provided that the reasoning level l is at least 1). This gives rise to a nested hierarchy of finite state controllers. Specifically (and continuing with the assumption that the frames of other agents at all levels are known and therefore they can be ignored), the algorithm maps the interactive state space $IS_{j,l} = S \times (\times_{k \in N \setminus \{j\}} \Delta(IS_{k,l-1}))$ into a new *compact interactive state space* $IS_{j,l} = S \times (\times_{k \in N \setminus \{j\}} \mathcal{F}_{k,l-1})$. Here each element in $\mathcal{F}_{k,l-1}$ consists of a node for each other agent in their respective finite state controllers. This transforms the interactive state space into a much smaller space, limited by the number of nodes in the lower-level finite state controllers.

Policy iteration in POMDPs consists of alternating between evaluating the policy and then improving it by generating new nodes corresponding to new vectors. A downside of policy iteration is that it causes the number of nodes to increase exponentially as iterations are performed. *Bounded* policy iteration provides a way to improve policies while keeping the number of nodes bounded. This is also the method I-BPI employs. In I-BPI policy iteration happens by iterating through the nested hierarchy of controllers bottom-up, evaluating and improving each policy at each level once before moving onto higher levels.

The authors found that I-BPI is able to solve problems with larger state spaces and deeper reasoning levels than I-PBVI. This is in part due to the reduction in the size of the interactive state space, aiding with the curse of dimensionality. However, a downside of the algorithm is that it may get stuck in a local optimum in the policy space instead of finding the overall optimal policy.

B.2.5 POMCP. The Partially Observable Monte Carlo Planning (POMCP) algorithm [13] is an online algorithm for solving POMDPs. While it is not an algorithm for solving I-POMDPs, we will nevertheless introduce its main ideas here. Many of them are also employed in our algorithm.

POMCP builds a search tree, where each node corresponds to a particular history of actions and observations. The tree starts with a single node (the *root node*) corresponding to an empty history. A node representing history h contains a value $V(h)$, a visitation count $N(h)$ and a set of particles $B(h)$. Each particle is a particular state s of the decision process. The set of particles is an unweighted representation of a belief.

The algorithm performs a look-ahead search from the initial belief. This is done through many Monte Carlo simulations, sampling successive sequences of states, actions, rewards and observations. The value $V(h)$ represents the average (discounted) utility of simulations starting from node with history h; the visitation count $N(h)$ counts the number of times a simulation has visited the node. A simulation starts by sampling a random particle from the root node. It then proceeds in two stages, which could be called the tree stage and the rollout stage.

In the tree stage, the actions are chosen informed by the nodes already in the tree. Specifically, the UCB1 (Upper Confidence Bound, version 1) formula from the UCT (Upper Confidence Bounds applied to Trees) algorithm is used to select the next action a as follows:

$$a = \arg\max_{a'} V(ha') + c\sqrt{\frac{\log N(h)}{N(ha')}} \tag{10}$$

The first term corresponds to a greedy selection of the action with the highest utility estimate. The second term is called the exploration bonus. It encourages choosing actions that are relatively unexplored. Note that an unexplored action ($N(ha') = 0$) is given "infinite" weight and is therefore always chosen first. The constant c controls the balance between exploitation of the actions believed to be the best and exploration of other actions. Tree search that uses the UCT algorithm is known as *Monte Carlo Tree Search* (MCTS).

After an action a is chosen and an observation o is received, the current state is propagated with the action using the transition function of the POMDP. The simulation moves to the node with history *hao*. This procedure is then repeated with the propagated state, using action selection from the new node.

The simulation enters the rollout stage when a previously untried action is chosen. A rollout policy, typically a uniform distribution over the possible actions, is used to propagate the action forward until a predefined horizon is reached. When the rollout finishes, all the states generated during the simulation are added to their respective nodes. Exactly one new node is added to the search tree: this node corresponds to the newly visited history after taking the first untried action. The visitation counts of the visited nodes are increased by one. The value of each visited node is updated to reflect the updated average discounted utility received starting from that node.

In addition to structuring the search, the tree functions as a particle filter. The particles in each node represent an updated belief after taking an action and receiving an observation. The tree can thus also be called a *belief tree*. This is utilised in the algorithm after a real action a is taken and a real observation o is received. The tree is simply pruned such that the node hao, where h is the history of the current root node, becomes the new root. The particles in the new root node represent the updated belief.

The authors report that POMCP is able to scale to very large problems (with up to 10^{56} states). This scalability is thanks to MCTS being able to break the curses of dimensionality and history: sampling is efficiently focused on the most relevant parts of the search space. Another advantage of POMCP is that instead of having to know the exact form of the transition and observation functions, it is sufficient to be able to sample from these distributions. This is advantageous in systems that are hard to represent explicitly in the POMDP framework.

B.2.6 I-NTMCP. Interactive Nested Tree Monte Carlo Planning (I-NTMCP) [11] is an online planning algorithm for I-POMDPs. Like POMCP, it uses belief trees and MCTS to perform a forward search. Unlike POMCP, it maintains a nested hierarchy of trees. Our algorithm is heavily inspired by it.

Specifically, at the top of the hierarchy at level L is agent i's own search tree $\mathcal{T}_{i,L}$. Each node in this tree corresponds to a specific *agent history* of i – a sequence of actions and observations of i. The search tree has a root node which corresponds to the actual agent history observed by the agent. Nodes contain particles, each of which contains a state. Importantly, they also contain a *joint history*, specifying the agent histories of each other agent. These particles are also called history states. The set of particles represents a belief. In addition, nodes contain a visitation count and an average value of simulations that have started from the node.

At one level below are agent i's models of the other agents. These are search trees too. For example, $\mathcal{T}_{(i,k),L-1}$ is agent i's model of agent $k \in N \setminus \{i\}$. The nodes in this tree correspond to agent histories of k. Since i does not know the exact agent history of k, there may be multiple root nodes in this tree[2]. The tree $\mathcal{T}_{(i,k),L-1}$ uses the trees $\mathcal{T}_{(i,k,j),L-2}$, $k \in N \setminus \{k\}$, as models of other agents. This hierarchy continues until level 0.

Simulations happen bottom-up: first, M simulations are performed in each of the level 0 trees. Next, the level 1 trees are simulated. This continues until the highest level. A simulation consists of the same steps as POMCP. A particle is sampled from one of the root nodes of the tree; it is propagated step by step down the tree, until a previously untried action is used to propagate it; and finally a rollout is performed and the value is backed up to all of the nodes visited during the simulation. However, due to the multi-agent nature of the problem, there are important differences, which we will cover next.

Sampling a particle to start simulating from is straightforward in the highest level tree $\mathcal{T}_{i,L}$; this tree has only one root node. To focus the simulations in the lower-level trees on the histories (and therefore beliefs) deemed most likely by i, the following procedure is used. First, a particle is sampled from the root node of the top-level tree $\mathcal{T}_{i,L}$. This particle contains an agent history for each other agent. The stored agent history is used to determine a root node in a lower level tree. For example, if we are simulating $\mathcal{T}_{(i,k),L-1}$, we use the agent history of k stored in the sampled particle to uniquely determine a root node in $\mathcal{T}_{(i,k),L-1}$. We can then start the simulation by sampling a particle from this node. If we are simulating a tree further down the hierarchy, we simply repeat the same procedure from the root node identified in the level $L-1$ tree.

The forward step differs from POMCP fundamentally because in an I-POMDP, actions from all agents are needed to propagate a state. The action of the agent whose behaviour the tree models can be determined by using the tree policy (10). We then need an action from each other agent. Luckily, since the lower-level trees have already been planned in, we have a model for which actions the other agents will choose. Specifically, we use the joint history stored in the particle we want to propagate to find matching nodes in the trees on the level below. Let the agent history of k stored in the particle be h_k, and assume we are simulating a level $l > 0$ tree. The probability of choosing each action a_k is determined using the softargmax function, proportional to

$$\exp\left(\frac{N_{k,l-1}(h_k a_k)}{\sqrt{N_{k,l-1}(h_k)}}\right) \tag{11}$$

where $N_{k,l-1}$ denotes the visitation counts in the tree $\mathcal{T}_{k,l-1}$. If $l = 0$, we assume some default distribution over actions, such as a uniform distribution[3]. After the actions of the other agents are chosen, a next state can be sampled by propagating the state in the particle with the joint action and the I-POMDP transition function.

I-NTMCP scales to much larger problems than I-POMDP Lite (Section B.2.3), the algorithm the authors use as a baseline. This demonstrates that Monte Carlo Tree Search can effectively be applied to multi-agent problems.

B.2.7 LABECOP. So far all of the algorithms we have seen rely on the fact that either the state space or the observation space (or both) are finite and discrete[4]. Since our model has both a continuous state space and a continuous observation space, we need an efficient algorithm that can handle this requirement. The last algorithm we cover, *Lazy Belief Extraction for Continuous Observation POMDPs* [7] is, as the name suggests, such an algorithm for POMDPs. While the authors describe the algorithm in terms of sequences of state-action-reward-observation quadruples, we will instead present it in the context of a search tree for ease of understanding.

Each node in the search tree corresponds to a specific *action history*. In other words, the branches of the tree only correspond to particular actions; the tree does not branch on observations. The root node corresponds to the history of actions performed so far. Nodes contain particles, each of which corresponds to a state. In addition, the current belief is represented and updated separately from the search tree, for example with a particle filter.

[2]More accurately, these trees are therefore *forests*, i.e. collections of trees. This is the terminology we adopt for our algorithm.

[3]Calling the lowest level tree level 0 is the convention the authors use. Note, however, that strictly speaking the level 0 model (tree) corresponds to level 1 in the I-POMDP framework, since other agents' behaviour is modelled using a subintentional model (random actions). In our work we call level 1 what the authors here call level 0. The term level 0 is correct if the lowest level model is defined as a POMDP.

[4]The interactive particle filter can technically support both, but its performance makes it an unattractive option.

Planning consists of sampling sequences of states, actions, observations and rewards, starting from the current belief. These sequences are encoded by the particles: each particle knows the previous and next particle in the sequence. A particle contains a value: the discounted sum of rewards received during the sequence from that particle onward.

A central idea in the algorithm is that the set of particles in a node can be used to represent many different beliefs by weighting them appropriately. As sampling progresses down the tree, new beliefs are formed in each node corresponding to the currently sampled sequence. Let h be the current action history. Say we choose action a in the node corresponding to h, and receive an observation o. The sampling proceeds to the node corresponding to action history ha. We need to form a belief in this node based on the observation by weighting the particles in it. The weight $w_{ha}(p')$ of particle p' with state $s(p')$ is given by Bayes' theorem:

$$w_{ha}(p') \propto w_h(\text{ancestor}(p'))Z(s(p'), a, o) \tag{12}$$

where ancestor(p') is the particle before p' in its sequence and w_h are the weights in the node corresponding to h. The weights are then normalised. By weighting all the particles in the node in this way, a new approximate belief is created. This belief approximation is more accurate for actions that are taken more often, since there are more particles to weight.

To actually sample episodes, we need to choose which actions to sample in each node we encounter. This is done using ideas from Monte Carlo Tree Search and UCB1. Specifically, given that we have already formed a belief b (as defined by the particles and the corresponding weights w) in a node, the action sampled is chosen as follows. Let $N_+(b)$ be the number of particles that get a positive weight under the belief. This approximates the total number of times actions have been sampled from the belief. Then let $W(b, a)$ be the sum of weights of particles that have a as the next action in their sequences. We can approximate the expected utility of taking action a from the belief and then continuing optimally as

$$\hat{U}(b, a) = \frac{1}{W(b, a)} \sum_{p:\, a(p)=a} w(p)V(p), \tag{13}$$

in other words, we calculate a weighted sum of the values of particles that have been with action a next ($a(p) = a$). Using these estimates, the next action is chosen with UCB1 according to

$$a = \arg\max_{a' \in A} \left(\hat{U}(b, a') + c\sqrt{\frac{\log N_+(b)}{N_+(b)W(b, a')}} \right). \tag{14}$$

The quantity in the numerator is the approximate number of times action a has been sampled from the belief. As before, c is the constant controlling the extent of exploration in action selection.

Sampling a sequence thus starts by sampling a state from the initial belief and then repeatedly choosing an action from a node, propagating the state with the action and creating a belief in the next node. After a previously untried action is sampled (the denominator of (14) is zero for some action), sampling stops, and a rollout is performed. The newly sampled particles are then added to each encountered node and their values are defined as the discounted sum of rewards received from that particle onward.

Table 1: The parameter values used in the experiments. Range refers to the values explored in experiment 1 and sensitivity analysis (Appendix C).

Parameter	Meaning	Value	Range
Parameters of Civilisations			
n	Number of agents	2	
r_D	Reward received by a destroyed civilisation	-1	
r_h	Reward for performing a hiding action	-0.01	
r_a	Reward for attacking another civilisation	$\{0, -0.1\}$	$[-0.2, 0.1]$
σ_{obs}	Standard deviation of observation noise	0.15	
L	Reasoning level of civilisations	$\{1, 2\}$	
γ	Discount factor	0.6	$[0.5, 0.7]$
v_m	Multiplier for visibility when performing a hiding action	0.5	
$\mathcal{I}_t$	Range of initial age	$[0, 50]$	
$\mathcal{I}_v$	Range of initial visibility	$\{1\}$	
$\mathcal{I}_{g_s}$	Range of possible growth speeds	$[0.3, 0.5]$	
$\mathcal{I}_{g_t}$	Range of possible takeoff ages	$[20, 40]$	
Parameters of the Solver			
n_{init}	Number of initial particles	1000	
n_{simul}	Number of simulations per forest per time step	10 000	
ε	Defines the discount horizon	0.1	
c_{explr}	MCTS exploration coefficient (see (4))	0.6	$[0.1, 1]$
c_{sft}	Softargmax coefficient (see (5))	0.1	$[0.01, 1]$
σ_{g_s}	Standard deviation of the noise added to the growth speed parameter	0.03	
σ_{g_t}	Scale of the noise added to the takeoff age parameter	3	

The performance of the algorithm is on par or even surpasses other algorithms on problems with continuous observation spaces. Many of the other algorithms tested had to use a discretised version of the observation space because they did not support continuous observation spaces. A downside of LABECOP is that the execution time increases superlinearly as more and more sequences are sampled, as later sequences have to weight more particles in each encountered node to form beliefs.

C PARAMETERS AND SENSITIVITY ANALYSIS

It is important to assess the extent to which the output of the model is sensitive to its input parameters. We have two groups of parameters: the parameters of the solving algorithm and the parameters of the model itself. Ideally, we would first find the parameters of the solving algorithm that are the most suitable for our model. After this, we would fix these and only assess the sensitivity of the model output to the model parameters. However, because model evaluations are quite costly, we opted to instead perform a global sensitivity analysis on a chosen subset of all of the parameters. For an explanation on how final solver parameters were chosen, see Appendix D.

We included two model parameters (the attack reward r_a and discount factor γ) and two solver parameters (the exploration coefficient c_{explr} and the softargmax coefficient c_{sft}) in the sensitivity analysis. Attack reward is an obvious choice for sensitivity analysis, since based on our experiments it greatly affects the outcomes. Discount factor is interesting to analyse since it determines the discount horizon, i.e. the amount civilisations look forward in time when they plan. The two solver parameters were chosen because among the parameters they seem the most influential: they determine how a forest is searched and how the results from that planning are used in planning in higher-level forests. The ranges explored are shown in Table 1. In addition, the analysis was performed at reasoning levels 1 and 2. We explore the sensitivity of two outputs: the proportions of attacks and hiding actions over a model run.

We calculate Sobol indices using SALib [5]. Sobol indices apportion the variation in the output variable of interest to the uncertainty in the model parameters [14]. *First order indices* indicate the proportion of the total variation of the output that is directly attributable to the parameter in question. *Total order indices* indicate how much of the variation is due to either the parameter itself or its interaction with other parameters. It is possible to also estimate *second order indices* which quantify the pairwise interactions, but we do not do that here since it is more computationally intensive. Since the indices are proportions, they should have values between 0 and 1. However, estimation errors can sometimes lead to negative estimates or estimates exceeding 1. Our Sobol sample consists of 64 samples for both reasoning levels, meaning 384 model evaluations per reasoning level.

The results of the sensitivity analysis are shown in Figure 2. For the proportion of attacks (top row), the results are as expected: most of the variation can be attributed to attack reward, while the influence of the other parameters is negligible. This is true at both reasoning levels.

For the frequency of hiding actions the results are less clear. The proportion of hiding actions remains relatively constant. Thus, there is not a lot of variance to explain and consequently the index estimates are quite uncertain. However, there is one parameter that stands out: at level 1 the discount factor γ accounts for between half and three quarters of all variance in the output. This warrants further investigation.

Figure 3 shows the proportion of hiding actions as a function of the discount factor. There does, indeed, seem to be a trend of increasing frequency of hiding when the discount factor is increased. As opposed to the result of the sensitivity analysis, the pattern seems to exist at both reasoning levels. The reason for why increasing the discount horizon increases hiding is unclear. One possible explanation is that looking further into the future gives the civilisation an earlier warning of a potential attack against it, which gives it more time to perform hiding actions. However, more investigation is needed.

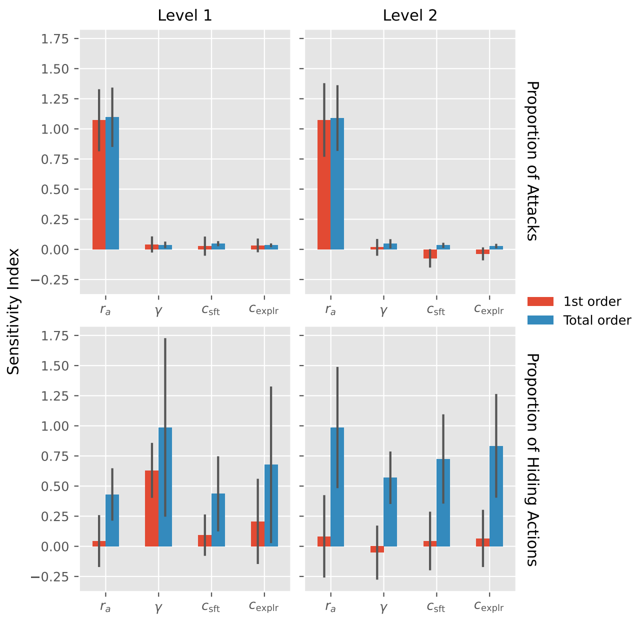

Figure 2: Sensitivity of proportion of actions that are attacks (top row) and hiding actions (bottom row). Sensitivity is shown for four continuous parameters (attack reward r_a, discount factor γ, softargmax coefficient c_{sft} and exploration coefficient c_{explr}) and for reasoning levels 1 and 2. Red bars show the first order sensitivity index and blue bars the total order sensitivity index. Black lines indicate 95% confidence intervals. Note that some indices exceed 1 or are negative. This is due to estimation errors caused by the limited number of samples we were able to perform.

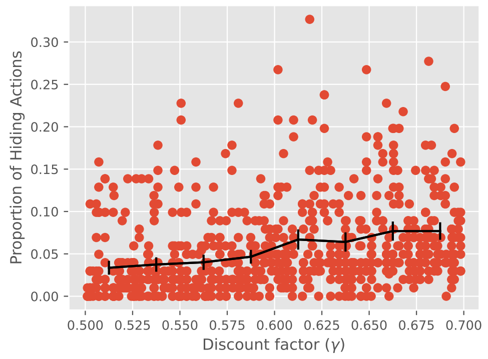

Figure 3: The proportion of hiding actions as a function of the discount factor γ. All data from the sensitivity analysis, both levels 1 and 2, are shown. Black line shows an average with 95% confidence intervals.

D PERFORMANCE AND PARAMETERS OF THE SOLVING ALGORITHM

In this section, we explain the rationale used when choosing parameter values for the solver algorithm. Since our algorithm is new, we will also present some of the data pertaining to its operation.

Ideally, we would first search the parameter space of the solver to find values that are optimal for our model. Optimal parameters of the solver could be found by testing the algorithm with different combinations of parameters and measuring the average reward (over time and over multiple repetitions) of an agent using it against various fixed opponents. Because of computational constraints, we opt instead to choose the values more heuristically. We will discuss these choices as we encounter relevant data.

First, we will show how i) the number of generated particles and ii) the fraction of all possible nodes explored varies across the depth of the search trees. We will compare two cases: when the number of samples per forest n_{simul} is varied, and when the length of the discount horizon (as defined by γ and $\varepsilon = 0.1$) is varied. We do this to get an understanding of suitable values for n_{simul} and γ. All the results in this section are for a model with two agents. Note that while γ is technically the amount of discounting civilisations do for future rewards and is thus not a solver parameter, here we slightly abuse it as such. The result of this is shown in Figure 4 (top two rows). In the left column we vary the number of simulations performed per forest. Doing 10 000 is at the limit of our computational budget so we want to investigate whether an additional 5 000 would make a significant difference. Since the difference between the results at 10 000 and 15 000 simulations is not very large and the number of particles per node is significantly higher at 10 000 than at 1 000, we chose the former. At ten thousand simulations all nodes of the forest are explored down to a depth of 4; naturally some nodes are explored more than others.

Our goal with choosing the discount factor γ was to make sure that each node has sufficient particles to accurately represent a belief. This is important as our belief space is continuous and has $4n$ dimensions, where n is the number of agents. We chose $\gamma = 0.6$ corresponding to a maximal search depth of 4, since the number of particles in the most explored node is roughly a thousand at minimum throughout the forest and on average the nodes have between 10 and 100 particles at depth 4.

An important aspect of the algorithm is determining the actions the other agents choose based on the lower level forests. We call this *querying* the forests. A query is not always successful. It is possible that the belief of the other agent has diverged. This means that all particles are deemed impossible based on the observations we have generated for the other agent as we simulate a higher level forest. It is also possible that not all of the actions have been expanded, or that the agent is altogether missing the relevant node in their forest. In all of these cases we assume the other agent chooses a random action, lacking a better alternative. However, it is important to quantify the extent to which this happens.

Figure 4 (bottom row) shows the success rates during the first planning step in a level 2 forest. We can see that at 10 000 simulations and with a search depth of 4 ($\gamma = 0.6$) the success rate is approximately 90% (the leftmost bar of lower right graph), which is reasonably good. At greater search horizon lengths the proportion of successful queries decreases, as the lower forest is less likely to have explored a specific node deeper in the forest.

The success rate changes over time. Figure 5 shows this for different combinations of the exploration coefficient c_{explr} and soft-argmax coefficient c_{sft}. These success rates are based on the sensitivity analysis data and thus represent a relatively broad sweep over the parameter space. They are shown over the entire duration of the simulation, which is 100 time steps. The figure shows that having a lower c_{sft} is advantageous for the success rate. This can be explained by the fact that if c_{sft} is very high, queries result in a broader range of possible actions of the other agent. Thus the following queries to the lower forest may target nodes that are not very well explored.

A sufficiently high exploration coefficient is also important. If the exploration coefficient is too low, even a single unlucky simulation in a forest might mean that the corresponding first action is never expanded again. On the other hand, the exploration coefficient must not be too high: otherwise the benefits of the UCB1 algorithm are lost and the search becomes a full-width expansion of the tree. Based on the success rates it seems that an exploration coefficient of over 0.55 is better than a smaller coefficient: the blue line remains above the red for a part of the hundred step interval, and otherwise they seem roughly equal. Based on these considerations and further trial-and-error tuning, we settled on $c_{\mathrm{sft}} = 0.1$ and $c_{\mathrm{explr}} = 0.6$.

REFERENCES

[1] Darius Braziunas. 2003. *POMDP solution methods: a survey.* Technical Report. Department of Computer Science, University of Toronto.

[2] Prashant Doshi and Piotr J. Gmytrasiewicz. 2009. Monte Carlo Sampling Methods for Approximating Interactive POMDPs. *J. Artif. Int. Res.* 34, 1 (March 2009), 297–337.

[3] Prashant Doshi and Dennis Perez. 2008. Generalized Point Based Value Iteration for Interactive POMDPs. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1* (Chicago, Illinois) *(AAAI'08)*. AAAI Press, 63–68.

[4] Piotr J. Gmytrasiewicz and Prashant Doshi. 2005. A Framework for Sequential Planning in Multi-Agent Settings. *J. Artif. Int. Res.* 24, 1 (July 2005), 49–79.

[5] Jon Herman and Will Usher. 2017. SALib: An open-source Python library for Sensitivity Analysis. *The Journal of Open Source Software* 2, 9 (Jan. 2017). https://doi.org/10.21105/joss.00097

[6] Trong Nghia Hoang and Kian Hsiang Low. 2013. Interactive POMDP Lite: Towards Practical Planning to Predict and Exploit Intentions for Interacting with Self-Interested Agents. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (Beijing, China) *(IJCAI '13)*. AAAI Press, 2298–2305.

[7] Marcus Hoerger and Hanna Kurniawati. 2021. An On-Line POMDP Solver for Continuous Observation Spaces. In *2021 IEEE International Conference on Robotics and Automation (ICRA)* (Xi'an, China). IEEE Press, 7643–7649. https://doi.org/10.1109/ICRA48506.2021.9560943

[8] Karim Jebari and Niklas Olsson-Yaouzis. 2018. A Game of Stars: Active SETI, radical translation and the Hobbesian trap. *Futures* 101 (2018), 46–54. https://doi.org/10.1016/j.futures.2018.06.007

[9] Tiancheng Li, Miodrag Bolic, and Petar M. Djuric. 2015. Resampling Methods for Particle Filtering: Classification, implementation, and strategies. *IEEE Signal Processing Magazine* 32, 3 (May 2015), 70–86. https://doi.org/10.1109/MSP.2014.2330626

[10] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. 2003. Point-Based Value Iteration: An Anytime Algorithm for POMDPs. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (Acapulco, Mexico) *(IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1025–1030.

[11] Jonathon Schwartz, Ruijia Zhou, and Hanna Kurniawati. 2022. Online Planning for Interactive-POMDPs using Nested Monte Carlo Tree Search. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 8770–8777. https://doi.org/10.1109/IROS47612.2022.9981713

[12] Guy Shani, Joelle Pineau, and Roberta Kaplow. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems* 27 (2013), 1–51. https://doi.org/10.1007/s10458-012-9200-2

[13] David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems*, J. Lafferty,

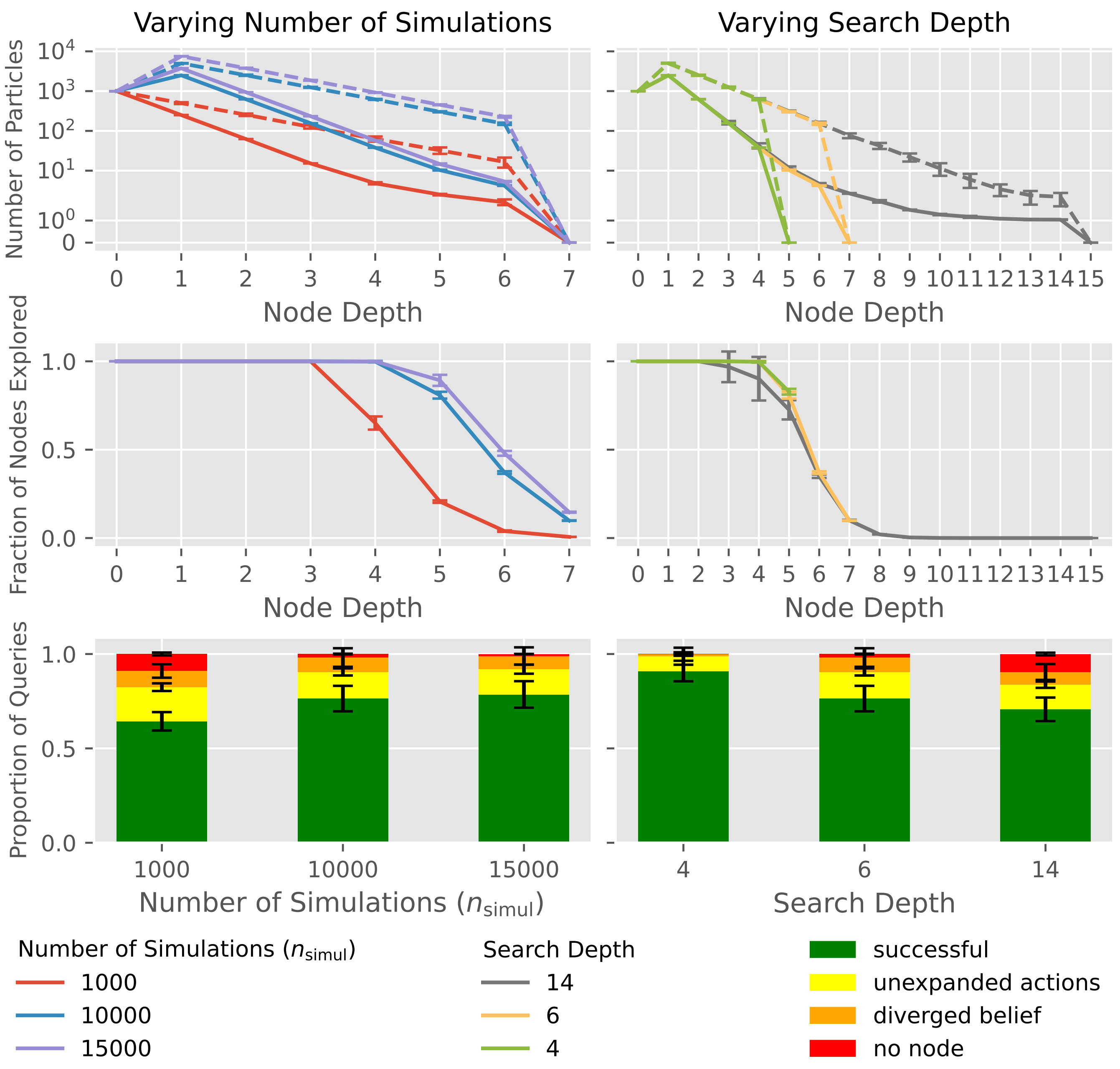

Figure 4: Metrics of a forest after a single planning step. In the left column the number of simulations n_{simul} is varied. In the right column the search depth is varied. The depths 14, 6 and 4 correspond to discount factors 0.85, 0.7 and 0.6. First row shows the average (solid line) and maximum (dashed line) number of particles in the nodes at the given depth, shown on the horizontal axis. Middle row shows the fraction of nodes out of all possible nodes that exist in the forest. The bottom row shows the success rate of determining the other agent's action using a lower level forest after it has been planned in once. The parameters used here are the same as Table 1 except for the ones varied. The discount factor γ is 0.7 instead of 0.6 to show particles at greater depths. The uncertainties shown are confidence intervals, calculated over five repetitions.

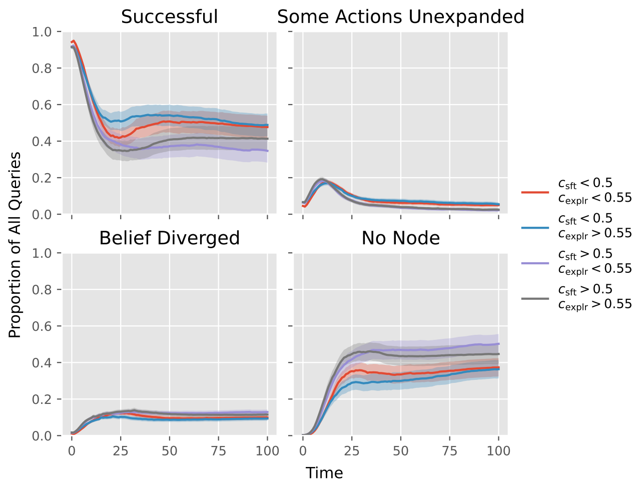

Figure 5: The success rates of queries to the level 1 forest of the other agent when planning at reasoning level 2. A query refers to determining the action of the other agent by calculating the optimal action in the forest that represents their policy. A query can be successful (top left), some of the actions might be untried (top right), the belief might have diverged (lower left) or the node might not have been visited in the forest at all (lower right). Shaded regions show a 95% confidence interval.

C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Eds.), Vol. 23. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf

[14] Ralph C. Smith. 2013. *Uncertainty Quantification: Theory, Implementation, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, PA. https://doi.org/10.1137/1.9781611973228

[15] Ekhlas Sonu and Prashant Doshi. 2015. Scalable solutions of interactive POMDPs using generalized and bounded policy iteration. *Autonomous Agents and Multi-Agent Systems* 29 (2015), 455–494. https://doi.org/10.1007/s10458-014-9261-5