COG-DICE: An Algorithm for Solving Continuous-Observation Dec-POMDPs

Madison Clark-Turner
Department of Computer Science
University of New Hampshire
Durham, NH 03824
mbc2004@cs.unh.edu

Christopher Amato
College of Computer and Information Science
Northeastern University
Boston, MA 02115
camato@ccs.neu.edu

Abstract

The decentralized partially observable Markov decision process (Dec-POMDP) is a powerful model for representing multi-agent problems with decentralized behavior. Unfortunately, current Dec-POMDP solution methods cannot solve problems with continuous observations, which are common in many real-world domains. To that end, we present a framework for representing and generating Dec-POMDP policies that explicitly include continuous observations. We apply our algorithm to a novel tagging problem and an extended version of a common benchmark, where it generates policies that meet or exceed the values of equivalent discretized domains without the need for finding an adequate discretization.

1 Introduction

As hardware costs have decreased, the prevalence of systems with multiple agents (e.g., routers, sensors, people, robots) has increased. These systems have the potential to be deployed in complex scenarios, but appropriate models and solution methods are needed to tackle the real-world issues of uncertainty and communication limitations. The decentralized partially observable Markov decision process (Dec-POMDP), which is an extension of the POMDP [Kaelbling *et al.*, 1998] to consider multiple decentralized agents, is a general model for considering these problems [Bernstein *et al.*, 2002; Oliehoek and Amato, 2016].

Despite the Dec-POMDP's potential to model many multi-agent problems, they have only been used to express small discrete problems. The many real-world problems that use continuous information (distance, velocity, and temperature among others) remain mostly unexpressed because no contemporary solvers can handle continuous observations although a few are capable of solving problems with continuous states [Amato *et al.*, 2017; Omidshafiei *et al.*, 2016; 2017]. It has been the role of the problem designer to separate continuous observations into a manageable number of discretizations. Accurately discretizing this information is hard even with expert knowledge of the problem. Naive discretizations that separate continuous information into equally sized sections could occlude pivotal aspects of the data that are only visible from the continuous perspective. In order for Dec-POMDPs to be truly useful, solution methods must solve problems with continuous values.

Therefore, this paper presents a solution representation for continuous observation problems using finite-state controllers and the continuous-observation Graph-based Direct Cross-Entropy (COG-DICE) algorithm for solving them. COG-DICE extends G-DICE [Omidshafiei *et al.*, 2016] to include continuous observations and improves its scalability when using discrete observations. Our method divides the observation space of an agent's policy into regions. This segmentation occurs separately on different parts of the solution (i.e., each node in the finite-state controller) in order to improve the solution's resulting quality. Our approach is evaluated on a novel tagging problem and a modified version of the recycling robots benchmark. In all cases, COG-DICE successfully generates solutions for problems with continuous observations that meet or exceed the results generated by current discrete solvers and hand-coded controllers.

2 Related Work

Though no continuous observation Dec-POMDP solvers currently exist, several continuous observation POMDP methods have been developed. In particular, methods have been developed that separate the observation space into a collection of distinct regions using the belief space of the POMDP to help identify the region boundaries [Hoey and Poupart, 2005; Brunskill *et al.*, 2008; Bai *et al.*, 2010]. These methods are similar in spirit to our approach, but the details are very different as we directly optimize the regions without the use of a belief state (which is not available to each agent in the decentralized case).

Many Dec-POMDP solvers have been developed [Oliehoek and Amato, 2016]. The Graph-Based Direct Cross-Entropy (G-DICE) algorithm is a state-of-the-art method that is scalable and can solve infinite-horizon problems with continuous state spaces [Omidshafiei *et al.*, 2016]. For these reasons, we build our continuous observation solver off of the existing G-DICE method.

3 Background

We first present a formal definition of Dec-POMDPs followed by an explanation of the G-DICE algorithm.

3.1 Dec-POMDPs

A Dec-POMDP is defined by a tuple $\langle I, S, A, O, T, \Omega, R, \gamma \rangle$, with a finite number of potentially heterogenous agents I; the set of states S; the set of joint actions A, and each agent $i \in I$ has a set A^i of actions that it is capable of performing; the set of joint observations O, and each agent has a set O^i of observations that it can receive; the state transition function T, where $P(s'|s,a)$ indicates the probability of transitioning from state $s \in S$ to $s' \in S$ after performing joint action $a \in A$; the joint observation probability function Ω, where $P(o|s',a)$ indicates the probability of receiving joint observation $o \in O$ after performing joint action $a \in A$ and transitioning to state $s' \in S$; the reward function $R(s,a) \in \mathbb{R}$ that attributes a reward for performing joint action $a \in A$ when in the current state $s \in S$; and a discount factor, $\gamma \in (0,1]$, that is used to weight the benefits of receiving rewards sooner rather than later.

Execution of a Dec-POMDP occurs over a series of discrete time steps. Beginning in an initial state $s_0 \in S$ (or distribution over states), each of the I agents chooses an action from their set of available actions A^i. Based on these actions, a joint reward is generated from $R(s,a)$, a new state, s' is transitioned to according to T and a set of observations is generated from Ω, one per agent. The Dec-POMDP can model stochastic action executions and observations through T and Ω respectively.

Each agent i selects actions according to its (decentralized) policy, π_i, which is a mapping from the agent's own observation histories to its actions. Because the problem's state is not visible, it is beneficial for agents to remember the history of observations received and actions performed. A joint policy π is a set of individual policies, one for each of the I agents.

Beginning in state s, obtaining the value (V^π) of an infinite-horizon policy, π, can be calculated using the Bellman equation for computing this expected sum:

$$V^\pi(s) = R(s,a) + \gamma \sum_{s',o} P(s' \mid a, s) P(o \mid s', a) V^\pi(s'),$$

which represents the reward $R(s,a)$ (where a is defined by π) plus the expected discounted value of performing the joint policy in all of the subsequent potential states. Note that the agents do not observe s, but choose actions from the policy based on the observation histories received. A joint policy π^* that maximizes V^π for the Dec-POMDP given an initial joint state s_0 is termed the optimal joint policy: $\pi^* = argmax_\pi V^\pi(s_0)$.

3.2 Policies as Finite-State Controllers

Infinite-horizon joint policies can be represented by a collection of I finite-state controllers (Figure 1(a)), one for each agent [Bernstein *et al.*, 2009]. These controllers select actions based on the current node and transition based on the observation seen. Specifically, the finite-state controller we describe is a Moore controller, which for a single agent i, is composed of a set of m nodes Q^i. Each node selects an action from A^i according to a distribution $P(a^i \mid q^i)$ (where $a^i \in A^i$ and $q^i \in Q^i$). Nodes are connected using $|O^i|$ directed edges; each edge is labelled with a different observation from the set O^i. Edge transitions can occur between any two nodes in Q^i. When an observation is received, the policy follows the edge with the corresponding label toward a new node according to $P(q^{i\prime} \mid q^i, o^i)$ (where $o^i \in O^i$). The graph has no terminal state and can continue for an infinite number of steps, making it a preferable representation for infinite-horizon problems. For simplicity, only a discrete version of a controller is given in Figure 1(a).

The Bellman equation can be defined using Moore controllers from initial state s and initial controller nodes for all agents q:

$$V^\pi(q,s) = \sum_a \prod_i^n P(a^i \mid q^i) \Big[R(s,a) + \gamma \sum_{s'} P(s' \mid a, s) \sum_o O(o \mid s', a) \sum_{q^{i\prime}} \prod_i^n P(q^{i\prime} \mid q^i, o^i) V^\pi(q', s') \Big], \quad (1)$$

where π is defined as the set of controllers with action selections and node transitions defined probabilistically.

3.3 Continuous Observation Dec-POMDPs

The previous Bellman equations (e.g., Equation 1) assume the set of states and observations are discrete. A modified equation that includes continuous values could be represented by replacing the sums with integrals:

$$V^\pi(s) = R(s,a) + \gamma \int_{s',o} P(s'|a,s) P(o|s',a) V^\pi(s') ds' do.$$

Unfortunately, this Bellman equation is challenging to evaluate without structural assumptions. Instead of evaluating the policy exactly, we approximate the value using Monte Carlo sampling [Omidshafiei *et al.*, 2016]. Monte Carlo sampling consists of executing the joint policy x times from a variable initial state, s, and then averaging the values that each execution generated: $\hat{V}^\pi(s) = \frac{\sum_x r^\pi(s)}{x}$, where $r^\pi(s)$ is the (discounted) sum of rewards for one Monte Carlo simulation from state s until a horizon cutoff has been reached (where any future value is negligible). The averaged value provides an estimation of the joint policy's quality that improves as the number of simulations increases. Note that this method allows us to *evaluate* a policy that considers continuous states and observations from a given initial state (but not generate such a policy).

3.4 G-DICE

G-DICE is a sampling-based approach for solving Dec-POMDPs (Algorithm 1) [Omidshafiei *et al.*, 2016]. The algorithm generates joint policies (as finite-state controllers) by randomly sampling their actions selected and transitions at each node from sets of distributions. The joint policies are then evaluated and those with the highest values are used to bias the set of distributions in the subsequent iteration where the process is repeated.

The sampling distributions (the output, $\mathbb{O}$, and transition, $\mathbb{T}$, functions) provide the probabilities that specific node-action mappings and node-node transitions will occur. When

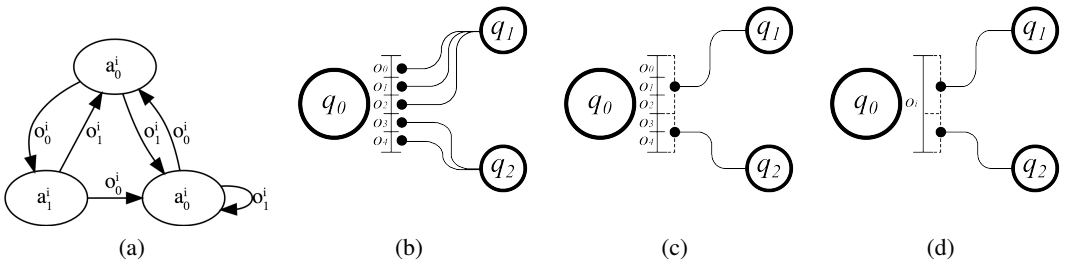

Figure 1: (a) An example Moore controller. (b) Node with discrete observation transitions. (c) Node with discrete region-dependent transitions. (d) Node with continuous region-dependent transitions (1 dimension).

G-DICE samples policies, it selects parameters according to these distributions. The output function for agent i ($\mathbb{O}^i$) outputs the probability that action a^i_j is chosen for node q^i_k in agent i's policy (where j and k are indices into A^i and Q^i respectively) using

$$\mathbb{O}^i_{j,k} = P(a^i_j | q^i_k). \tag{2}$$

The transition function for agent i ($\mathbb{T}^i$) similarly outputs the probability that node q^i_j will transition to node $q^{i\prime}_k$ having received observation o^i_l (where j and k are indices to Q^i, and l is an index into O^i) using

$$\mathbb{T}^i_{j,k,l} = P(q^{i\prime}_k | q^i_j, o^i_l). \tag{3}$$

G-DICE has the following input parameters: the number of nodes in each agent's policy, N_n, the number of iterations to perform, N_i, the number of samples to generate in each iteration, N_x, the maximum number of samples used to update the sampling distributions with N_k, and a learning rate, α.

The algorithm begins by initializing $\mathbb{O}$ and $\mathbb{T}$ as uniform distributions across all of the N_n nodes for each agent's policy (Algorithm 1, lines 1-3). The values V_b and V_w, the best and worst values observed by G-DICE so far, are also initialized and set to negative infinity (lines 4 and 5). The algorithm proceeds to enter the main loop, which it performs for N_i iterations. In each iteration, G-DICE generates N_x sample joint policies (line 8). The joint policies consist of I controllers, one for each agent, that are each composed of N_n nodes. Actions and transitions for each node are randomly sampled according to the distributions in $\mathbb{O}$ and $\mathbb{T}$. Once the joint policies have been generated, they are evaluated (line 10). In this paper we used Monte Carlo evaluation.

Joint policies with value less than V_w are pruned, while those that are greater than V_b replace the current best joint policy and the value of V_b (lines 11-16). The top N_k joint policies with values greater than V_w are stored in ϕ_{list} along with the best joint policy observed thus far (line 17). The value of V_w is then set to the lowest value in ϕ_{list} as the new pruning metric for the subsequent iteration (line 18).

The joint policies in ϕ_{list} are then used to seed the new sampling distribution for the next iteration. The maximum likelihood estimate (MLE) of occurrences of action-to-node mappings and node-to-node transitions (which are the action and node frequencies in this case) are used to generate a new set of probabilities for the next iteration. These probabilities are smoothed using a learning rate (α) and combined with the previous iteration's probabilities (lines 19-23).

Algorithm 1: The G-DICE Algorithm

```
   input : N_n, N_i , N_x , N_k, α
1  for n = 1 to N_n do
2      Initialize the output function (𝕆);
3      Initialize the transition function(𝕋);
4  V_b ← −∞;
5  V_w ← −∞;
6  for i = 1 to N_i − 1 do
7      φ_list ← ∅;
8      for x = 1 to N_x do
9          π ← sample from 𝕆_i and 𝕋_i;
10         V_π ← Evaluate π;
11         if V_π ≥ V_w,i then
12             φ_list ← φ_list ∪ {π};
13             V_list ← V_list ∪ {V_π};
14         if V_π ≥ V_b then
15             φ_b ← π;
16             V_b ← V_π;
17     φ_b,list, V_b,list ← N_k policies in φ_list;
18     V_w,i+1 ← min(V_b,list);
19     for n = 1 to N_n do
20         𝕆_i+1 ← MLE of 𝕆_i using φ_b,list;
21         𝕆_i+1 ← α𝕆_i+1 + (1 − α)𝕆_i;
22         𝕋_i+1 ← MLE of 𝕋_i using φ_b,list;
23         𝕋_i+1 ← α𝕋_i+1 + (1 − α)𝕋_i;
24 return φ_b ;        // The best joint policy
```

4 Continuous Observations

G-DICE works well for problems with discrete observations, but it is incapable of generating policies for problems with continuous observations. With continuous observations, each node in the Moore controller would require an infinite number of transitions: an impossible task to represent let alone compute. Furthermore, discretization is also problematic. Every node has an edge for each of the agent's observations connecting the node to other nodes in the controller. If the number of observations increases, then so will the total number of transitions in the controller by N_n for each new observation. When

generating new samples, each transition is selected independently and therefore problems that possess fine discretizations are more complex and become intractable to solve. To deal with these issues, we develop a continuous observation controller representation and a continuous observation version of G-DICE, which exploits the knowledge that similar observations in a continuous space will transition to similar nodes to partition the observation space.

4.1 Continuous-Observation Controller

For the policy representation, we again use the Moore controller, but now include modifications that express transitions across regions of the observation space rather than with discrete observations (as shown in Figure 1(d)). We begin by introducing divisions (d), separators that denote boundaries in the observation space. As a parameter, COG-DICE allows the designer to select the number of partitions that the observation space is split into (N_t). This partitioning of the space occurs separately for each node. For example, with a one-dimensional continuous observation space over the interval $[0, 1]$ the divisions consist of a set of $N_t - 1$ points. These points partition the space using $[0, d_1), [d_1, d_2), \ldots, [d_{N_t-1}, 1]$. In problems with multi-dimensional observation data, divisions can be represented using hyperplanes or nonlinear separators.

Division functions ($\mathbb{D}$) are distributions that describe where the divisions will be placed (e.g., probabilities for the locations of the points in the one-dimensional space above). $\mathbb{D}$ is expressed differently in continuous and discrete-observation problems. In the discrete-observation problem, we will partition the observation space by ordering the observations and choosing discrete observations to denote the endpoints of the partitions, as shown in Figure 1(c). This partitioning is done separately for each agent and each node, so there exists divisions $d_l^{i,k}$ (for the l-th division for agent i on node k). The probability that a division is at a particular observation is expressed in the agent i's division function $\mathbb{D}^i$ as

$$\mathbb{D}^i_{j,k,l} = P(o^i_j | q^i_k, d^i_l), \tag{4}$$

which represents the probability that division l for agent i at node k occurs at observation j.

The continuous observation case lacks problem specified values on which to place the divisions and so divisions must be selected from the entire observation space (Figure 1(d)). Therefore, we use a probability distribution over this continuous space for representing where to place each division probabilistically. Many distributions could be chosen based on the domain. In the one-dimensional case over a bounded interval, the Beta distribution is a natural choice for each division, but other distributions (such as the Dirichlet) could also be used. The Beta distribution is parameterized by α and β, so again we can represent the probability that division l for agent i at node k occurs over the interval [0,1] as

$$\mathbb{D}^i_{k,l} = Beta(\alpha^i_{k,l}, \beta^i_{k,l}). \tag{5}$$

In problems with larger dimensionality, one-dimensional concepts can be extended by splitting the observation space along each axis separately, similar to axis-parallel splits, or distributions over nonlinear separators could be used (e.g., kernels).

In addition to selecting points that divide the observation space, we must redefine the node transition function ($\mathbb{T}$) to be the same for all observations within a partition. A partition (or region) of the observation space can be written as $\mathcal{R}_{d^i_m d^i_n}$, where d^i_m and d^i_n are two adjacent divisions for the same node that bound the observation space. A transition from node q^i to node $q^{i\prime}$ in agent i's controller can now be expressed over regions rather than over each observation as

$$\mathbb{T}^i_{j,k} = P(q^{i\prime}_j | q^i_k, \mathcal{R}_{d^i_m d^i_n}). \tag{6}$$

Execution of the joint policy is unchanged with the exception of node transitions. Traditional Moore controllers follow transitions labelled by the discrete observation that was received after performing the most recent action. In the continuous-observation controller, the transition is defined by determining the region of the observation and following the transition for that region.

4.2 COG-DICE

The key extensions in COG-DICE are shown in Algorithm 2. In summary the differences include: the new controller representation with continuous observations, the evaluation of this representation, sampling from the continuous division distribution, and updating the continuous division distribution. While these operations have the same general form as the original G-DICE algorithm, they are nontrivial to design and more complex to carry out due to being continuous rather than discrete.

One notable difference is the addition of input parameter N_t, the number of regions the observation space should be separated into at each node. In the initialization step of the algorithm, the action selection function, $\mathbb{O}$, is unchanged between the two algorithms, however, in COG-DICE the updated node transition function $\mathbb{T}$ and division function $\mathbb{D}$ are set to uniform distributions (Algorithm 2, lines 3 and 4). That is, we are using the updated controller representation and searching not only over the action selection and node transition parameters, but also the division parameters.

In COG-DICE's main loop we generate a set of joint policies and update the sampling distributions. The process of generating joint policies occurs by selecting actions for each node according to $\mathbb{O}$ (line 8), divisions of the observation space for each node according to $\mathbb{D}$ (line 11), and transitions from each node to other nodes based on regions of the observation space according to $\mathbb{T}$ (line 12). For example, assuming our problem has continuous one-dimensional observations, we could use the Beta distribution in $\mathbb{D}$, as described in Section 4.1, to express the probability for each division at each node of each agent. As a result, N_x controllers are sampled for each agent.

The joint policies are evaluated, again using the new method described in 4.1, and the best N_k joint policies are placed in ϕ_{list} which is used to update the sampling distributions for the next iteration. $\mathbb{O}$, $\mathbb{T}$, and $\mathbb{D}$ are all updated by taking the MLE of the values present in ϕ_{list}. Intuitively, the MLE for the action selection and node transition distributions can be calculated based on the counts of these parameters from the best N_k joint policies (lines 14-17), while the distributions over divisions in the continuous case represent the

Algorithm 2: The COG-DICE Algorithm

input : N_n, N_i, N_x, N_k, N_t, α
1 **for** $n = 1$ *to* N_n **do**
2 Initialize the output function ($\mathbb{O}$);
3 Initialize the transition function($\mathbb{T}$);
4 Initialize the division function($\mathbb{D}$);
⋮
5 **for** $i = 1$ *to* $N_i - 1$ **do**
6 **for** $x = 1$ *to* N_x **do**
7 **for** $n = 1$ *to* N_n **do**
8 $\pi_s \leftarrow$ sample from $\mathbb{O}_i$ for each agent;
9 **for** $t = 1$ *to* N_t **do**
10 **if** $t < N_t - 1$ **then**
11 $\pi_s \leftarrow$ sample from $\mathbb{D}_i$ for each agent;
12 $\pi_s \leftarrow$ sample from $\mathbb{T}_i$ for each agent;
⋮
13 **for** $n = 1$ *to* N_n **do**
14 $\mathbb{O}_{i+1} \leftarrow$ MLE of $\mathbb{O}_i$ using $\phi_{b,list}$;
15 $\mathbb{O}_{i+1} \leftarrow \alpha\mathbb{O}_{i+1} + (1-\alpha)\mathbb{O}_i$;
16 $\mathbb{T}_{i+1} \leftarrow$ MLE of $\mathbb{T}_i$ using $\phi_{b,list}$;
17 $\mathbb{T}_{i+1} \leftarrow \alpha\mathbb{T}_{i+1} + (1-\alpha)\mathbb{T}_i$;
18 $\mathbb{D}_{i+1} \leftarrow$ MLE of $\mathbb{D}_i$ using $\phi_{b,list}$;
19 $\mathbb{D}_{i+1} \leftarrow \alpha\mathbb{D}_{i+1} + (1-\alpha)\mathbb{D}_i$;
20 **return** ϕ_b ; `// The best joint policy`

most likely distribution given the best sampled division points (lines 18 and 19). The MLE of a Beta distribution [Natrella, 2010] can be computed by obtaining the sample geometric mean G_x of the set of normalized data values and the sample geometric mean's mirror image $G_{(1-x)}$, with:

$$G_x = \prod_{i=1}^{N} (x_i)^{1/N} \tag{7}$$

$$G_{(1-x)} = \prod_{i=1}^{N} (1 - x_i)^{1/N} \tag{8}$$

These values can then be used to approximate α and β using the logarithmic approximation of the digamma function with:

$$\alpha = \frac{1}{2} + \frac{G_x}{2(1 - G_x - G_{(1-x)})} \tag{9}$$

$$\beta = \frac{1}{2} + \frac{G_{1-x}}{2(1 - G_x - G_{(1-x)})} \tag{10}$$

The algorithm continues updating the sampling distributions based on the highest-valued policies and sampling new policies until the desired number of iterations. While probabilistic convergence to the optimal solution in the limit given the controller size and partition number can be proven in the discrete case (by extending earlier proofs [Omidshafiei *et al.*, 2016; Oliehoek *et al.*, 2008]), a similar proof for the continuous case is future work.

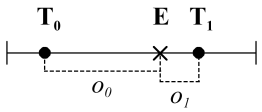

Figure 2: The tagging problem. The evader (E) is represented as an X and the taggers (T) are represented as black dots. Dashed lines indicate observations.

5 Experiments

The COG-DICE implementation was tested on two different domains: a tagging simulator and a recycling robots problem.

5.1 Tagging Simulation

As seen in Figure 2, in the one-dimensional tagging problem, multiple taggers work to tag a single evader in as short a time as possible. The state space consists of the locations of the taggers and evader and the observations are the distance and direction of the tagger to the evader. These are represented using one-dimensional continuous values (denoted by [0,10] and (-10,10) respectively). Negative observations indicate the distance that the evader is to the left of the tagger and positive observations indicate the distance to the right. Tagging agents can perform stochastic move actions (left or right one space) or a tagging action. Move actions and observations are noisy. The tagging action returns a positive reward if it is executed in a fixed proximity to the evader and a negative reward otherwise. The evader follows its own fixed policy to try and escape the taggers.

In an attempt to compare our method to a state-of-the-art optimal Dec-POMDP solver, we solved a discrete version of the tagging problem with GMAA*-ICE [Spaan and Oliehoek, 2008]. This problem domain had 1000 states and 361 possible joint observations. GMAA*-ICE was able to generate policies to a horizon of two after an hour of execution, but was unable to solve the horizon three problem even after multiple days of execution. We expect similar results with other optimal Dec-POMDP methods.

COG-DICE was applied to two variants of the one-dimensional tagging simulation: one with discrete observations and one with continuous observations. Both variants used continuous states and two tagging agents. For all tagging problems we set the input parameters as: N_n = 3, N_i = 500, N_x = 2000, N_k = 50, N_t = 3, α = 0.1. We restricted the controller size to see how a concise policy performed on this problem and chose other parameters that seemed reasonable. We also tested other similar parameter values and the results did not change significantly. We also compare with a hand-coded controller, which moves the taggers toward the evader and attempts to tag if it is within an appropriate range.

We first compare our discrete-observation results for various discretization granularities. Recall that for this case, COG-DICE will place the divisions only at the given discretizations. As seen in Table 1, when the tagging problem used discrete observations, the value of the joint policies generated by COG-DICE increased as the size of the discretizations decreased. The observation space was discretized into regions of size 0.5, 1.0, 2.0, and 4.0. Observations were rounded to the nearest discretization. As expected, the level

Discretization	Hand-Coded	COG-DICE
continuous	-	101.8
0.5	86.32	89.9
1.0	68.89	65.09
2.0	-22.11	54.09
4.0	-154.44	13.49

Table 1: Values of joint policies in the tagging problem

of discretization exhibited strong influence over the structure and value of the solutions that were generated. If the divisions were located too far outside of the tagging range (e.g. discretizations of 2.0 and 4.0), then the taggers would have to perform additional move actions in order to ensure that the evader was within the tagging proximity. Using coarser discretizations reduced the time until convergence, but resulted in poorer solutions.

We also see that COG-DICE can outperform the hand-coded solution as seen in Table 1. As the discretization becomes finer both methods improve, but because COG-DICE is optimizing the solution based on the given discretization, it can often perform better. When finer discretizations were employed, the COG-DICE policies resembled the hand-coded controllers in both structure and value but took increasingly longer to converge due to the greater number of potential division points to investigate. Convergence was assumed when the average deviation in policy values for the previous 20 iterations fell below 0.01. The time taken until convergence doubled between discretizations of 4.0 (70050s) and 0.5 (140700s) indicating the computational strain of the problem. Interestingly, the convergence of discretization of 1.0 (79670s) occurred prior to the convergence of the a discretization of 2.0 (99330s) but this was due to the finer discretization having randomly sampled a joint policy closer to the optimal solution before than the more coarse discretization. Note that implementation improvements and parameter turning could greatly reduce running time and high-quality solutions were found long before convergence.

In the continuous-observation results we were able to generate a joint policy using COG-DICE that exceeded the values of any discretization. The algorithm successfully identified division points in the observation space that maximized the potential to tag correctly. The continuous-observation version of COG-DICE also outperformed all discretizations of the hand-coded controller, substantially improving solution quality by considering the full range of observations.

5.2 Recycling Robots

In the decentralized recycling robot problem, two agents plan when to collect small and large cans or recharge their batteries to maximize overall efficiency. Gathering small cans depletes the battery a small amount and returns a small reward, but can be done by each robot independently. Moving the big can creates a large reward, but requires collaboration and diminishes the battery more aggressively. Exhausting the battery is undesirable and produces a negative reward.

Current implementations of the recycling robots problem use discrete values ("high" and "low") for observations of the remaining battery charge [Amato *et al.*, 2012]. We present a continuous state and observation representation using the percentage of the battery remaining. In the original problem, the probability of the battery state transitioning to a lower state decreased from the "high" to "low" states. For the continuous problem, we used noisy linear functions to represent the decrease in battery expenditure across the state space.

As seen in Table 2, the highest known value for the discrete infinite-horizon recycling robots problem is 31.93 [MacDermed and Isbell, 2013]. We were able to generate a similar value (31.27) on the same problem using G-DICE in 1605s. We used the same parameters for G-DICE as we did in the tagging simulation problem with exception to N_k where we used 25. By decreasing the number of retained samples we encourage the policy to converge more quickly.

In both joint policies of the discrete problem, the agents would only ever collect small cans or recharge. Theoretically the agents should be able to perform the large can action and small can actions several times before needing to recharge. Unfortunately, because the battery state information is coarse, agents are unable to make guarantees about the other agent's state and therefore cannot coordinate cooperative actions.

Problem Setting	Value
Highest Known Value	31.93
Discrete Observations (G-DICE)	31.27
Continuous Observations (COG-DICE)	34.43

Table 2: Values of recycling robot joint policies

COG-DICE was able to generate a joint policy with a value that exceeded the discretized problem value (34.43) in 21500s (although, again, high-quality policies were found long before this time) by making use of coordinated actions. COG-DICE used the same parameters as G-DICE and N_t was set to 2 so that the boundaries between different battery states would be easily identifiable. The joint policy, through the use of continuous observations, was able to make guarantees about how battery states in both robots would diminish over time and coordinate cooperative actions and recharges so that the policies could be more efficiently optimized.

6 Conclusion

This paper presented, for the first time, an algorithm that generates joint policies for Dec-POMDPs with continuous observations. We presented both a discrete-observation version of the algorithm, which is applicable in domains with a large number of discrete observations, and a continuous-observation version. This method is broadly applicable as many real-world domains have large or continuous observation spaces. COG-DICE has been successful in generating joint policies for both a novel and a preexisting problem and has highlighted the negative impacts that inappropriate discretization can have on joint policy structure and value. For future work, we are interested in extending this work to high-dimensional observation spaces by exploring other (nonlinear) divisions and optimizing the algorithm parameters by either integrating these optimizations into the algorithm or possibly building on previous work on Bayesian non-parametrics [Liu *et al.*, 2015].

Acknowledgments

We thank our collaborators at Lincoln Lab for helpful discussions about this work, particularly Dan Griffith and Emily Anesta. Partial funding was provided by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001.

References

[Amato *et al.*, 2012] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2012.

[Amato *et al.*, 2017] Christopher Amato, George D. Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P. How, and Leslie P. Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 2017.

[Bai *et al.*, 2010] Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A Ngo. Monte Carlo value iteration for continuous-state POMDPs. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, pages 13–15, 2010.

[Bernstein *et al.*, 2002] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[Bernstein *et al.*, 2009] Daniel S Bernstein, Christopher Amato, Eric A Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34(1):89, 2009.

[Brunskill *et al.*, 2008] Emma Brunskill, Leslie Pack Kaelbling, Tomas Lozano-Perez, and Nicholas Roy. Continuous-state POMDPs with hybrid dynamics. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 2008.

[Hoey and Poupart, 2005] Jesse Hoey and Pascal Poupart. Solving POMDPs with continuous or large discrete observation spaces. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1332–1338, 2005.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:1–45, 1998.

[Liu *et al.*, 2015] Miao Liu, Christopher Amato, Xuejun Liao, Lawrence Carin, and Jonathan P How. Stick-breaking policy learning in Dec-POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2015.

[MacDermed and Isbell, 2013] Liam C MacDermed and Charles Isbell. Point based value iteration with optimal belief compression for Dec-POMDPs. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 100–108, 2013.

[Natrella, 2010] Mary Natrella. NIST/SEMATECH e-handbook of statistical methods. 2010.

[Oliehoek and Amato, 2016] Frans A Oliehoek and Christopher Amato. *Concise Introduction to Decentralized POMDPs*. Springer, 2016.

[Oliehoek *et al.*, 2008] Frans A Oliehoek, Julian FP Kooij, Nikos Vlassis, et al. The cross-entropy method for policy search in decentralized POMDPs. *Informatica*, 32(4):341–357, 2008.

[Omidshafiei *et al.*, 2016] Shayegan Omidshafiei, Ali-akbar Agha-mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. Graph-based cross entropy method for solving multi-robot decentralized POMDPs. In *Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, pages 5395–5402. IEEE, 2016.

[Omidshafiei *et al.*, 2017] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. Decentralized control of multi-robot partially observable markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, pages 231–259, 2017.

[Spaan and Oliehoek, 2008] Matthijs T. J. Spaan and Frans A. Oliehoek. The MultiAgent Decision Process toolbox: software for decision-theoretic planning in multiagent systems. In *Multi-agent Sequential Decision Making in Uncertain Domains*, 2008. Workshop at AAMAS.