The Sensor Management Prisoners Dilemma: A Deep Reinforcement Learning Approach

W. R. Faber

ABSTRACT

The number of sensor resources available to track Resident Space Objects has dramatically increased in recent years due to the increase of commercial sensor providers, investments in government sensor capabilities, and the need for data supporting Space Domain Awareness (SDA) and Space Traffic Management (STM). The current state-of-the-practice is for sensor providers to perform sensor collection campaigns in a decentralized fashion. Volumes of space go without surveillance while global STM data sets often contain unwanted redundancies. One solution to this problem would be to centralize the tasking, i.e., require that tasking is performed at a single centralized location. This is not a feasible approach since providers would then have to relinquish control of their network to a single end user, limiting their ability to meet other end user requirements. This work presents an alternative solution that allows providers to maximize their impact to collaborative STM. This responsible approach seeks global STM optimality within a Decentralized Sensor Management (DSM) paradigm. To achieve this approach the DSMP is framed as an iterative simultaneous game. The DSMP can be seen as an adaption the classical problem called, The Prisoners Dilemma. In this paper, the problem is formalized as a Partially Observable Markov Decision Process (POMDP). A Deep Reinforcement Learning (DRL) approach is developed to determine an optimal policy that allows the agents to avoid Nash Equilibria and improve overall reward. This method is applied to an optical network example and performance improvements are shown against current a self-interest policy.

1. INTRODUCTION

Maintaining custody and ensuring Safety of Flight (SoF) for tens of thousands of Resident Space Objects (RSOs) is a daunting task that requires a significant amount sensor resources. Sensor data providers maintain distributed networks of sensors located throughout the world and in space. Providers range in their ability to provide data based on a number of different reasons, such as, the number, type, and performance of sensors in their network. Combining observational data from multiple networks may provide improved coverage and tracking of RSOs. However, network management is performed in a decentralized fashion where each provider performs network optimization independently. Combined data products from decentralized networks are often sub-optimal when considering the group performance. This leads to unnecessary waste of available resources and unwanted redundancies in data. This paper addresses the Decentralized Sensor Management (DSM) problem and develops a new Deep Reinforcement Learning (DRL) approach that allows individual providers to manage their network in way that improves group performance.

The centralized Sensor Management Problem (SMP) deals with optimal allocation of sensor resources and can be framed as a Markov Decision Process (MDP) [7, 3]. Many approaches have been developed to find optimal decision making strategies for the SMP. Treating the problem as a stochastic optimal control problem on the information state one can utilize stochastic Dynamic Programing (DP) [10] or solve the DP problem in a receding horizon fashion [1, 13, 2, 12]. Other techniques have been used to avoid local optima when solving SMP to maximize overall information gain [9, 8]. Reinforcement Learing (RL) [14] and DRL have also shown success in SMP. Relevant examples being from Linares and Furfaro where they developed Actor-Critic methods for the space object tracking problem, as well as, Asynchronous Advantage Actor Critic (A3C) techniques to scale to large tasking problems such as those required for RSO cataloging [4, 5].

In transition from centralized to decentralized the problem shows similarities to a classical problem in Game Theory called the Prisoner's Dilemma [6]. Section 2 describes the problem in general and relationships are illustrated in section 3. This relationship provides a starting point for the formulation described in section 4 that frames the DSM problem as a Partially Observable Markov Decision Process (POMDP) and discusses the DRL approach. Lastly, section 5 provides an illustrative example that simulates a game between optical network providers. Performance improvements are shown against current a self-interest policy. The contributions of this paper are as follows:

- The formalization of the DSM RL problem as a POMDP
- A DRL sensor network management technique capable of determining an optimal policy to improve overall group performance
- A DSM technique that can accommodate for non-synchronous decision-making epochs
- The ability to predict uncooperative adversary sensor management behavioral patterns

2. PRISONER'S DILEMMA

The Prisoner's Dilemma (PD) is a hypothetical scenario used in Game Theory, Philosophy, and other fields to study decision-making differences between group optimality and individual optimality. It presents an example where two individuals acting in their own self-interest do not reach optimality. The scenario involves two prisoners being interrogated separately by the police. The prisoner's are provided two options, to either confess or to stay silent[1]. Each prisoner's resulting jail time is dependent on both their own decision and the other prisoner's. This frames a non-cooperative, two player, complete information, simultaneous game that is summarized using the Game Theoretical Normal-Form in Fig. 1 The normal-form shows outcomes for each of the different decision making combinations. For instance, if Prisoner 1 decides to stay silent and Prisoner 2 chooses to confess, then Prisoner 1 would receive five years while Prisoner 2 gets no jail time.

		Prisoner 2	
		Silent	Confess
Prisoner 1	Silent	-1, -1	-5, 0
	Confess	0, -5	-3, -3

Fig. 1: Prisoner's Dilemma Normal-Form

In general, when both players confess they each get a moderately negative sentence. If one player stays silent and the other confesses, then the silent prisoner receives a severe negative sentence while the player that confesses receives no penalty. However, if both players stay silent, then they each receive a mild sentence. In this game, the *Pareto-Optimal*[2] occurs when both players choose to stay silent, resulting in mild sentences. This outcome is represented by the red circle in Fig. 2. However, when a player attempts to determine a dominant strategy all indications point to choosing the option to confess. This leads to a Nash Equilibrium, represented by the red circle in Fig. 3, where both players choose to confess and both receive a moderate sentence.

[1] The PD scenario is often explained differently depending on the field of study. Typical differences involve changes to the choices provided to the prisoners. For example, cooperate or defect and testify or lie are two common sets of choices that are used as alternatives to confess or stay silent. Regardless of the choices the core principles remain the same.

[2] *Pareto-Optimal* is the outcome at which no change can occur that is beneficial to one without being detrimental to others

		Prisoner 2	
		Silent	Confess
Prisoner 1	Silent	-1, -1	-5, 0
	Confess	0, -5	-3, -3

Fig. 2: *Pareto-Optimal* of the Prisoner's Dilemma

Consider the perspective from Prisoner 1. If Prisoner 2 decides to stay silent, then the best response would be to confess, which would result in zero total jail time for Prisoner 1. If Prisoner 2 decides to confess, then the best response would also be to confess, resulting in three years of jail time. From either Prisoner's perspective the dominant decision is to choose to confess. This leads to both prisoners being worse off.

		Prisoner 2	
		Silent	Confess
Prisoner 1	Silent	-1, -1	-5, 0
	Confess	0, -5	-3, -3

Fig. 3: Nash Equilibrium of the Prisoner's Dilemma

3. PRISONER'S DILEMMA IN DECENTRALIZED SENSOR SCHEDULING

Maintaining accurate state information for Resident Space Objects (RSO) often requires sensor resources that are distributed around the world. In some circumstances, it also requires multiple sensors in localized regions to meet accuracy requirements. Global sensor networks are expensive and require considerable infrastructure to construct and maintain. An alternative to building and maintaining a global network is to acquire data from multiple distributed data providers. With the increase in commercial and academic sensor resources the availability and variety of quality SSA data has skyrocketed, making the acquisition of data more cost effective. However, acquiring data from other entities often comes at the cost of sensor tasking control. In this case, the sensor scheduling problem becomes decentralized making it extremely difficult to achieve sensor tasking optimality. This leads to having a sub-optimal data set for RSO tracking. There are other examples where trade offs like this occur. For instance, in STM the goal is to coordinate space safety and sustainability within an international community of space actors. Many efforts aim to share data between nations to better support the STM mission, however, due to the decentralized nature of data collection there are often unwanted redundancies in data, as well as, regions of space without surveillance. Another example that causes a decentralized tasking issue is when sensors within a centralized network do not behave according to their tasking.

At a simplified level the decentralized sensor tasking problem can be framed similarly to the PD. Consider a scenario where two data providers are providing SSA data to a single end user. Both Provider 1 and Provider 2 have the choice to provide data on either object A, O_A, or object B, O_B. Both providers want to maximize the value of their data but they know that the end user is more concerned with object coverage. The value of tasking O_B is significantly more than the value of tasking O_A. However, if both providers task the same object the data becomes redundant and less valuable to the end user. If instead the providers task separate objects, then the tasks retain their value. The following example is depicted in Fig 4.

		Provider 2	
		O_A	O_B
Provider 1	O_A	2, 2	3, 6
	O_B	6, 3	4, 4

Fig. 4: The sensor management prisoner's dilemma in normal-form

In a similar fashion to the prisoners in section 2, the providers can determine a dominant strategy for selecting between O_A and O_B. In any case, the dominant choice is to task O_B. This results in a Nash Equilibrium since no player would change their decision given prior knowledge of the other player's choice[3]. The resulting payout from the Nash Equilibrium is show using a red circle in Fig. 5. This payout or outcome is less desirable to the end user because they

[3]This is under the assumption that each player is rational

are more interested in sensor coverage and would prefer the providers tasked separate objects.

		Provider 2	
		O_A	O_B
Provider 1	O_A	2, 2	3, 6
	O_B	6, 3	4, 4

Fig. 5: Nash Equilibrium of the sensor management Prisoner's Dilemma

The simplified depiction of the sensor management PD in Fig. 4 shows a snapshot in time with two-players and only two available actions. In reality, the decentralized sensor management problem is far more complex. It can more accurately be seen as a multi-player, multi-action, imperfect, incomplete, non-cooperative, simultaneous, iterated game. These complications are compared, summarized and justified in Fig. 6

Category	Standard PD	Sensor Management PD	Explanation
Number of Players	Two Player	n-Player	Dependent on what is modeled as a player. Players can be a single sensor or a network. In either case there can be many players.
Number of Actions	One Action	m-Action	Since players are modeled as single sensors or networks of sensors each sensor/network can see multiple objects at any given time
Complete/Incomplete	Complete	Incomplete	Players in real sensor management scenarios may not know timing of actions nor the full extent of the other players action space
Perfect/Imperfect	Perfect	Imperfect	Perfect knowledge of opponents actions is not available. Only imperfect observations of the actions
Iterative/Non-iterative	Non-iterative	Iterative	Real systems take actions sequentially over time and have intermittent latencies

Fig. 6: Summary of complications for real sensor management problems

4. FORMALIZATION OF DECENTRALIZED SENSOR MANAGEMENT

Decentralized sensor management deals with optimizing over a network of sensor providers where each provider tasks their resources independently. Fig. 7 illustrates a network of three providers each with a varying number of sensor resources. The problem can be viewed from two different perspectives, internal and external. The internal perspective treats the problem as a multi-player game where the game is played from a single player's viewpoint. The goal of the game would be to manage the player's sensor resources in a way that maximized the overall reward from all players. The external perspective instead treats the problem from an outsiders viewpoint that can not directly influence any of the providers' resource management strategies. Instead the outsider can manipulate a provider dependent reward function to drive towards maximizing the overall reward. Both internal and external perspectives can be framed as a Partially Observable Markov Decision Process (POMDP). This paper deals strictly with the internal perspective and leaves the external perspective as a topic for future research.

Fig. 7: Network of sensor networks where each pin represents a provider's sensor resource

4.1 Game Description and Formalization

Consider a decentralized tasking scenario with N different providers each with n_i sensor resources, $i = 1, ..., N$. There exists a finite set of objects $X = \{x_i, x_2, ..., x_m\}$ with cardinality m. At any given time, t, providers can use any number of sensors $n_i(t) \leq n_i$ to track a random subset of the objects $X_i(t) \subseteq X$. The expected cardinality of the random finite set $E[|X_{i,t}|] \leq k_i$ where $k_i < m$. The game is built on top of this framework and is played from a single provider's perspective. At each time step the provider selects an action according to a developed policy. The provider selects actions based only prior observations of the game's state, i.e., independent of explicit knowledge of other players policies. The reward structure favors provider selections that do not overlap with other players' actions. This game can be framed as a POMDP represented by the tuple,

$$< S, A, \mathscr{O}, P, R, Z, \gamma > . \tag{1}$$

The components of the tuple in Eq. 1 are:

- S is a finite set of states
- A is a finite set of actions
- $\mathscr{O}$ is a finite set of observations
- P is Markov transition probability function

- R is the reward function
- Z is the observation function
- γ is a discount factor

The current state of the game $S(t)$ is represented by a $k \times m$ pixel image. Each row is a vector $s(t) = [s_1, s_2, ..., s_m]$ where $s_i \in \{0,1\}$ corresponding to a boolean value for tasked and not tasked. The set of states, S, is equivalent to the permutations of the matrix. The value k is a hyper-parameter representing an integer value of time stpes. It is set to match the revisit rate requirement and provides the game with a moving history of previously tasked objects. In real scenarios, providers may never share their tasking that make up the values within the states, S. However, noisy observations of the tasks are available. The observation function Z maps the states into a finite set of observations, $\mathscr{O}$, which take form as images the same size as the states within S. This process is modeled as a Bernoulli Process with parameter P_{md}. The Bernoulli parameter is a hyper-parameter that can be tuned to model real missed detection probabilities. When P_{md} is set to one this POMPD becomes an MDP. Each action in the set of actions, A corresponds to tasking a single object in the set of objects X. The action based Markov transition function is, $P^a_{ss'} = P[S(t+1)|S(t) = s, A(t) = a]$. It is known that given the simultaneous nature of the game that $P^a_{ss'}$ is extremely difficult to model accurately. However, real sensor actions in DSM are bounded by natural processes. This allows for one to extract accurate models of $P^a_{ss'}$ using model-based RL approaches. Lastly, the reward structure is,

$$R^a_s = E[R_t|S(t) = s, A(t) = a] = \frac{1}{m}\sum_{i=1}^{m} \delta_i(S_{k,i}) \tag{2}$$

where $\delta_i(S_{k,i}) = 1$ if there exists a non-zero value in the $S_{k,i}$ column vector and $\delta_i(S_{k,i}) = 0$ otherwise. Now that the POMDP is established, one can define a policy that represents the distributions over actions given the states,

$$\pi(a|s) = P[A(t) = a|S(t) = s]. \tag{3}$$

Then, the state-value function and action-value functions can be written as follows,

$$v_\pi(s) = E_\pi[G_t|S(t) = s], \tag{4}$$

$$q_\pi(s,a) = E_\pi[G_t|S(t) = s, A(t) = a]. \tag{5}$$

The term G_t is the discounted return such that $G_t = \sum_{j=0}^{\infty} \gamma^k R_{t+k+1}$. The POMDP can be solved by determining either the optimal state-value function $v^*(s)$ or the optimal action-value function $q^*(s,a)$. For any MDP there exists one or more optimal policies π^* that achieve the optimal $v^*(s)$ and $q^*(s,a)$. Furthermore, $v^*(s)$ and $q^*(s,a)$ are recursively related by the Bellman optimality equations. The Bellman equations are non-linear and have no general closed form solution. Thus, many iterative Reinforcement Learning techniques have been developed as solutions, such as, Value Iteration, Policy Iteration, and Q-Learning.

4.2 Deep Reinforcement Learning Algorithm

In order to determine the optimal policy for the POMDP outlined in section 4.1, a DRL approach has been developed to determine π^* through collaborative learning that combines policy gradient RL with a network trained to predict opponent moves. The technique was adapted specifically for this research, but utilizes attributes used widely throughout the RL to overcome some of the cons of policy iteration. It should also be noted that typical DRL techniques are not offend recommended for game situations as described by the POMDP and the game description in Fig. 6. The main reasons being that most success in DRL has been shown on sequential, single action per turn games. Major aspects of solution have been evolved to account for the simultaneous, multi-player, multi-action per turn game that is DSM.

The solution begins by developing an agent consisting of two separate networks that perform collaboratively. The first network, called the provider, takes in the observed state image, $\mathscr{O}(t)$, and provides the policy, $\pi_\theta \approx \pi(s,a)$. The second network, called the predictor, takes in the opponents previously observed action image, $\mathscr{O}(\sqcup)_o$, and predicts the n-opponent provider predicted action, i.e., it predicts what the other STM providers are going to task. The provider is built using a Convolutional Neural Network (CNN) with parameters θ. These parameters define the policy π_θ.

The goal of the provider is to converge on an optimal policy π^*_θ. The predictor is also built using a CNN. The predictors parameters, ω, instead define a m-dimensional vector of independent probability values, p_ω, that present the probability of any action being selected at time t by opponent providers. The predictor can be run iteratively to determine a look ahead horizon of predicted opponent actions. This look ahead horizon is h steps long where h is a hyper-parameter[4]

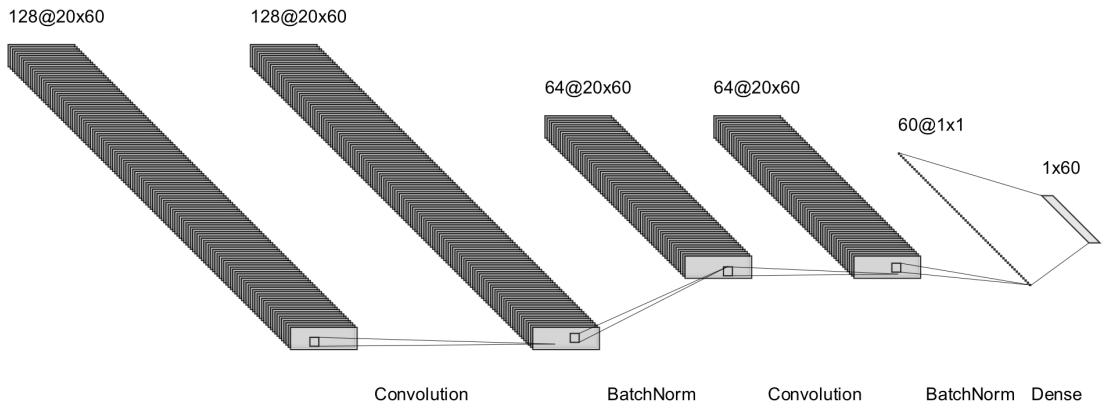

Fig. 8: Baseline Convolution Neural Network behind both Provider and Predictor Networks within DRL agent.

Fig. 8 illustrates the baseline CNN used to build both the Predictor and Provider networks within the DRL agent. Leaky Rectified Linear Unit (LeakyReLU) activation was used in between convolution layers and dense layers. Default values were used for stride and filter size. Sigmoid was used as an activation for the output layer. Max pooling layers were specifically avoided to retain as much information as possible about the relative distance between features within the image.

Parameters are updated by playing games between the collaborative Provider and Predictor Networks and random sets of n-opponents. This is done by building a simulation environment that executes a user defined number of iterated DSM games that simulate random numbers of opponent providers and behaviors. Opponent providers are modeled within the environment and behave according to deterministic behavioral models that mimic real systems.

The Provider, or policy, network is trained through Gradient Ascent in order to maximize,

$$J(\theta) = E_\pi[G(t)]. \tag{6}$$

In this fashion the parameter update equation is,

$$\theta := \theta + \alpha \nabla J(\theta). \tag{7}$$

In this solution, the $\Delta\theta = \alpha\nabla J(\theta)$ value is approximated by determining an approximate best response policy, $\hat{\pi}$ and minimizing loss[5] between $\hat{\pi}$ and π_θ.

$$\Delta\theta \propto \frac{\partial \pi_\theta}{\partial \theta}(\hat{\pi} - \pi_\theta) \tag{8}$$

The best response policy is determined in a similar fashion to the Monte Carlo Tree Search used in successful networks such as AlphaGo by DeepMind [11]. Actions are sampled in Monte Carlo and the return, G_t, is calculated by the simulation environment utilizing the provided state observations $\mathscr{O}(t)$, and the look ahead predictions from the predictor network. The G_t values are averaged over all sampled $A(t)$ to approximate $\hat{\pi}$.

After each time step the observed opponent actions, $\mathscr{O}_o$, are fit to the predictor network to update predictor parameters ω. This is done in a similar fashion to the Provider network parameters, except the loss is minimized between the

[4]Prediction errors build exponentially in time. If h is too high then errors will dominate prediction leading to false-belief in expected opponents actions. However, if h is too low then, since the provider is trained using this horizon, the predictor will move towards myopic policies

[5]Due to the nature of the problem their exists a distribution over π that is a kin to a multi-modal distribution. Utilization of Mean Square Error (MSE) loss is not advantageous. I have found loss convergence to be well behaved when using Binary Cross-Entropy (BCE) for both Provider and Predictor Networks

output prediction, $\overline{\mathscr{O}_\omega}$, and $\mathscr{O}_o$. Note the observed state, $\mathscr{O}(t) \equiv \mathscr{O}_o(t) \cup \mathscr{O}_p(t)$, where $\mathscr{O}_p$ is the observed state that corresponds to tasked objects by the provider agent and $\mathscr{O}_o$ is the last line of the image $\mathscr{O}_o(t+1)$.

$$\Delta\omega \propto \frac{\partial \overline{\mathscr{O}_\omega}}{\partial \omega}(\mathscr{O}_o - \overline{\mathscr{O}_\omega}) \tag{9}$$

This is process is continued until parameters θ and ω reach desired convergence. The flow for a single game is shown in Fig. 9.

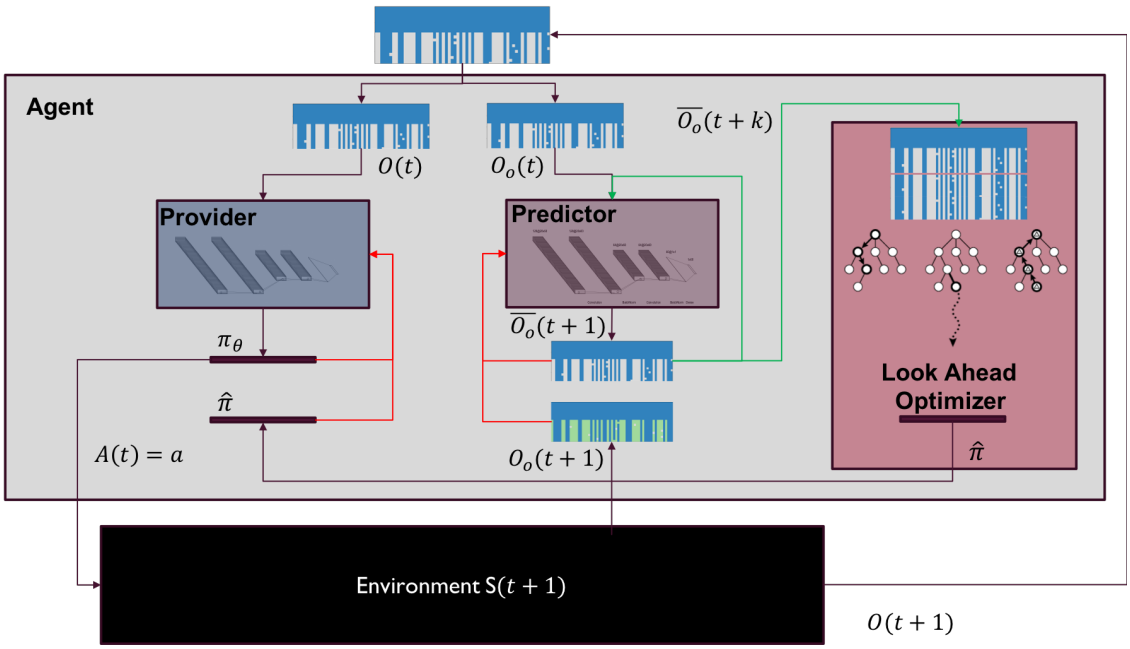

Fig. 9: Flow diagram of information through the agent during the learning process. Red lines signify parameter update processes while green lines are associated with the look ahead process.

5. SIMULATION RESULTS

This section discusses the results from utilizing the provider and predictor networks described in section 4.2 on the game outlined in Fig. 6 and formalized in section 4. Python and Keras were used to construct the simulation environment and all relevant aspects of the POMDP and DRL agents. Simulations were performed on a 2.5 GHz Intel Core i5 with 16GB ram. Due to the limited compute resources used in simulation, the real DSM problem was paired down by reducing the required revisit rate and reducing the cardinality of the taskable object set. The results are compared to the performance of a non-RL agent fitted with a self-interest based policy.

The simulation consists of a set of objects, X, with cardinality, $m = 60$ in space. Subsets of X are visable by $m = 9$ sensor resources operated by m independent opponent sensor providers. The behavioral function of the opponent networks within the simulated environment can be modified to mimic any real system. In this simulation, the opponent providers have deterministic tendencies such that the environment simulates them tasking the same sets of objects at each step for any given game. This is done to simulate a common behavior of persistent stare ground-based optical sensors. Each of the nine sensors can task more than one object at any given time step. In this simulation, the RL agent controls a single sensor capable of tasking one object per time step. The agent controlled sensor is modeled as steerable ground-based optical system. The set of actions available to the agent, A, consists of 60 unique tasks, one for each object in X. The agent is trained to learn a policy that maximizes the global coverage of all the providers and attempts to meet revisit rate requirement such that all 60 objects are observed within $h = 20$ time step horizons. Each horizon consisting of a 10-minute interval resulting in a 200-minute minimum revisit rate window. For proof of concept p_{md} and k are both set to 1.

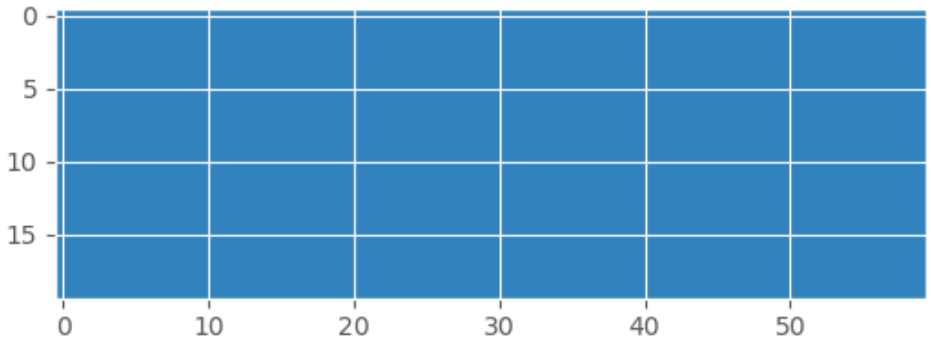

Fig. 10: Initial game state image $S(t)$ where $t = t_0$. the y-axis corresponds to time while the x-axis spans A

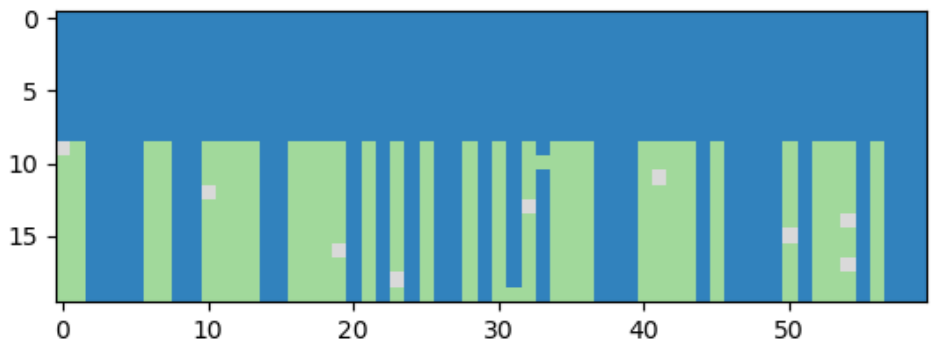

Fig. 11: Game state image $S(t)$ where $t = 10$ using Self-Interest based policy

Fig. 10 shows the image at the start of a game, $t = t_0$. This game image corresponds to a reward, $R(t) = 0$. If Fig. 10 was a plot time-steps would be the y-axis with the $0-th$ pixel location being the oldest. The tasked objects would span the x-axis. As the game is simulated and actions are taken, the image begins to populate with tasks from both the agent and the opponent provider networks. Fig. 11 shows a game simulated using a self-interest based policy at time step $t = 12$. All green vertical bars correspond to the persistent opponent sensor tasking. The tasking provided by the self-interest based agent can be recognized as gray pixels or the pixels outside vertical bars. Fig. 12 shows an image of a game state after time has progressed passed the horizon h. In both images, Fig. 11 and Fig. 12, the self-interest based policy does not optimize for overall sensor coverage. This results in duplicates and lost objects. Duplicates occur when the policy favors actions that are already being taken by opponent providers. They can be seen as the gray pixels in Fig. 11 and Fig. 12. Lost objects are those that have not been tasked within the minimum revisit rate window h. These lost objects can be seen as blue vertical bars in the images.

The Provider DRL agent outlined in section 4.2 learned through experience by playing 5000 simulated games against random opponents. Fig. 13 shows the accuracy and loss throughout the 5000 games of both internal networks, Provider and Predictor. The resulting agent was then used on the same game shown earlier with the self-interest based agent in Fig. 11 and Fig. 12. The image in Fig. 14 was taken at the same time as in Fig. 11. Even early in the game

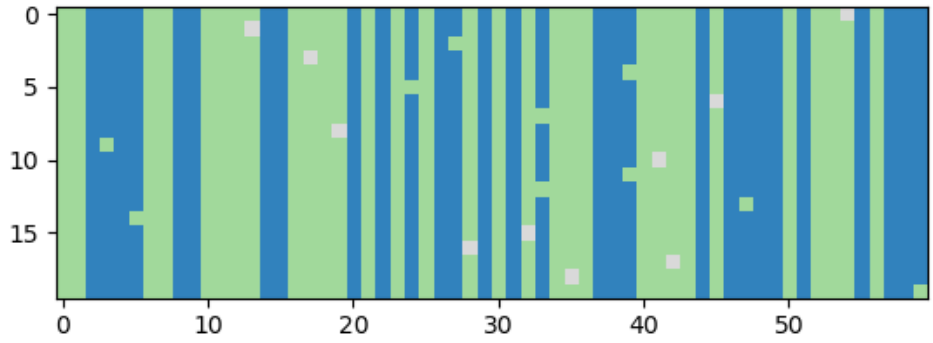

Fig. 12: Game State image $S(t)$ where $t = 35$ using Self-Interest based policy

Fig. 13: Game play loss and accuracy for both the provider and predictor networks

improvements can be seen when using the DRL agent. For instance, there is only one color, gray, instead of green and gray as in Fig. 11. This is a visual queue that shows that there are no duplicate tasks at any given time. The image in Fig. 15 was taken at the same time as in Fig. 12. There continues to be no duplicate tasks when using the DRL agent and there are fewer lost objects. The images from the game played with the proposed DRL agent correspond to higher overall returns and, in turn, better overall network coverage.

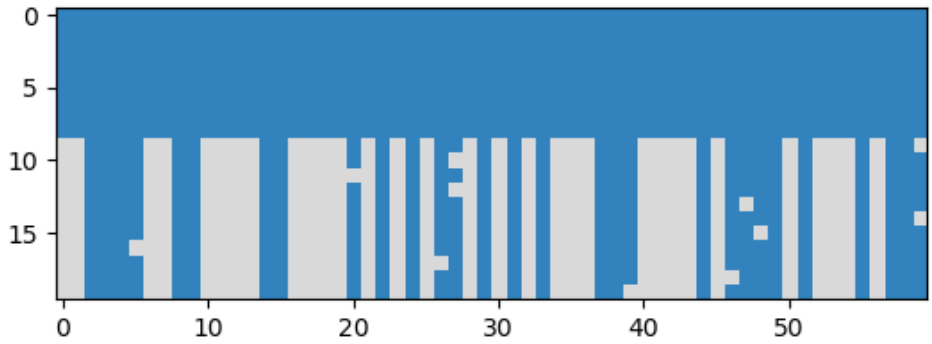

Fig. 14: Game state image $S(t)$ where $t = 10$ using Provider network policy

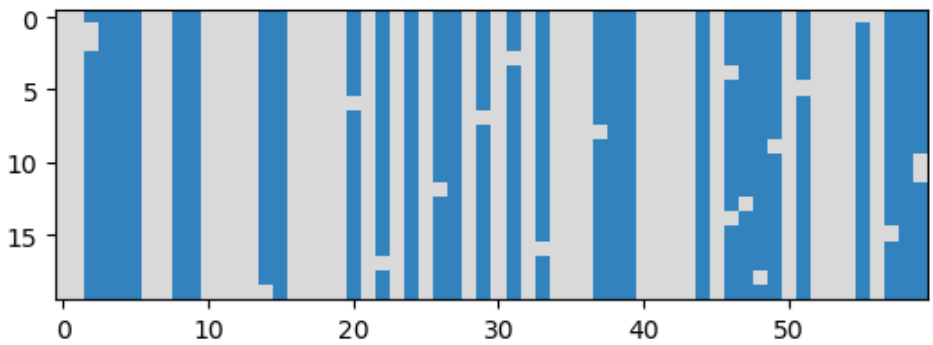

Fig. 15: Game state image$S(t)$ where $t = 35$ using Provider network policy

To further illustrate of the superior performance by Provider agent, 50 games were simulated and used to compare between the DRL Provider agent and the self-interest based agent. Each game was simulated for 40 time steps. The results are shown in Fig. 16. On average the Provider agent's return was 33% higher then the self-interest based agent. Also, the Provider agent scored higher than the self-interest based agent on all 50 games.

6. CONCLUSION

In summary, this paper provides initial results showing the viability of a new DRL algorithm on a variation of the Decentralized Sensor Management (DSM) problem. The solution consists of developing a Deep Reinforcement Learning (DRL) agent, that represents a single data provider, to determine an optimal policy through trial and error. The optimal policy achieves the optimal state-value and action-value function of the POMPD developed in 4.1. The problem formulation was developed by drawing the relationship between, then building upon the Prisoner's Dilemma. The approach was demonstrated using a decentralized Resident Space Object (RSO) tracking problem and performance was compared that of an agent with a built in self-interest based policy. The agent's internal networks provide the agent with capability to select actions that increase the groups overall object coverage. The internal Predictor network can also predict opponent providers future tasks with up to 90% accuracy when the opponent behaves rationally. Future work will scale the solution using more computational resources to meet needs of larger RSO tracking problems. It

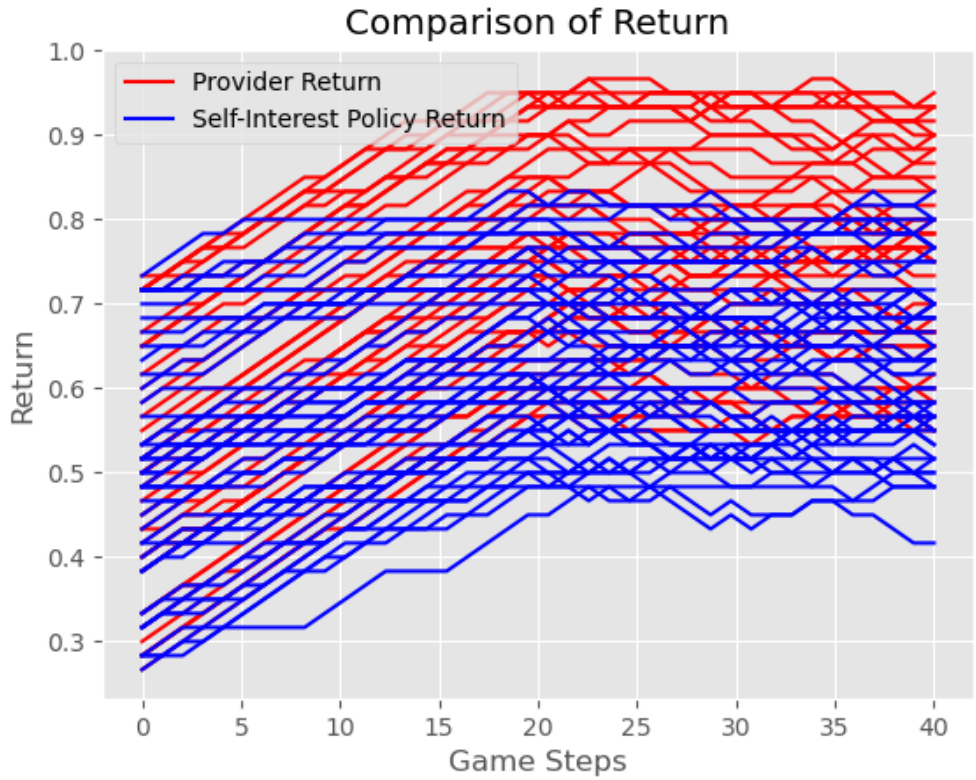

Fig. 16: Provider agent (red) performance compared to self-interest (blue) approach in 50 game simulation. y-axis shows return where 1 means perfect coverage while 0 is no coverage.

will also look to develop a solution to the problem from an external DSM perspective as described in section 4. Lastly, reasoning will be applied over the Predictor network to infer opponent sensor capabilities.

7. REFERENCES

[1] S. Chakravorty and R. Erwin. Information space receding horizon control. In *IEEE Int. Symp. on Approx. Dyn. Prog. and Reinf. Learn.*, 2011.

[2] S. Chakravorty and R. Erwin. Information space receding horizon control. *Proceedings of the 2011 International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011.

[3] D. L. Isaacson and R. W. Madson. *Markov Chains: Theory and Applications*. Wiley, New York, NY, 1976.

[4] R. Linares and R. Furfaro. Dynamic sensor tasking for space situational awareness via reinforcement learning. 2016. Advanced Maui Optical and Space Surveillance Technologies Conference,, Maui, HI,.

[5] R. Linares and R. Furfaro. An autonomous sensor tasking approach for large scale space object cataloging. 2017. Advanced Maui Optical and Space Surveillance Technologies Conference,, Maui, HI,.

[6] William Poundstone. *Prisoner's Dilemma*. 1st Anchor Books ed. Anchor, 1993.

[7] M. Puterman. *Markov Decision Process*. Wiley-Interscience, Hoboken, NJ, 1994.

[8] D. Raihan, W.R. Faber, S. Chakravorty, and I.I Hussein. Aas/. In *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference*, Snowbird, Utah, 2018.

[9] D. Raihan, W.R. Faber, S. Chakravorty, and I.I Hussein. Sensor scheduling under action dependent decision making epochs. In *Proceedings of the 22nd International Conference on Information Fusion*, Ottawa, Canada, 2019. IEEE.

[10] J. Si, A. G. Barto, W. B. Powell, and D. C. Wunsch II (Eds). *Hanbook of Learning and Approximate Dynamic Programming*. IEEE Press, Piscataway, NJ, 2004.

[11] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484489, 2016.

[12] Z. Sunberg, S. Chakravorty, and R. Erwin. Information space receding horizon control for multi-agent systems. In *American Control Conference*, 2011.

[13] Z. Sunberg, S. Chakravorty, and R. Erwin. Multi agent information space receding horizon control. *Proceedings of the 2012 American Control Conference*, July 2012.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.