

---

# Gradient-based Optimization for Multi-resource Spatial Coverage

---

**Nitin Kamra**

Department of Computer Science  
University of Southern California  
nkamra@usc.edu

**Yan Liu**

Department of Computer Science  
University of Southern California  
yanliu.cs@usc.edu

## Abstract

Resource allocation for coverage of geographical spaces is a challenging problem in robotics, sensor networks and security domains. Recent gradient-based optimization solutions estimate utilities of actions by using neural networks to learn a differentiable approximation to spatial coverage objectives. In this work, we empirically show that spatial coverage objectives with multiple-resources are combinatorially hard to approximate for neural networks. Further, we propose a tractable framework to approximate a general class of spatial coverage objectives and their gradients using a combination of Newton-Leibniz theorem, spatial discretization and implicit boundary differentiation. We empirically demonstrate the efficacy of our proposed framework on multi-resource spatial coverage problems.

## 1 Introduction

Allocation of multiple resources for efficient spatial coverage is an important component of many practical systems, for e.g., robotic surveillance, mobile sensor networks and security game modeling. These systems generally involve assigning resources e.g. drones or sensors, each of which can monitor physical areas, to various points in a target domain such that a loss function associated with coverage of the domain is minimized [17]. Traditional methods used to solve multi-resource surveillance problems often rely on potential fields [6], discretization based approaches [12], voronoi tessellations [3] and particle swarm optimization [15, 18]. In case of multiple agents competing for placement, exact and approximate approaches have been proposed to maximize the agents' expected utilities in security domains [11, 1, 8, 7].

**Related Work:** Since spatial coverage problems feature continuous action spaces, a common technique used across many previous works is to discretize the area to be covered into grid cells and restrict the resource placement to discrete cells [12, 20, 5, 4] to find optimal allocation using integer linear programming. However, discretization quickly becomes intractable when the number of resources grows large. Recent works in spatial coverage domains have focused on incorporating advances from deep reinforcement learning to devise more general algorithms. For instance, Pham *et al.* [16] focus on the multi-UAV coverage using a model-free multi-agent RL method. StackGrad [1], OptGradFP [9], PSRO [13] are model-free fictitious play algorithms for allocating resources in the presence of multiple agents to cover continuous target spaces. To have better sample efficiency, more recent works take an actor-critic based approach [14], which additionally learns a differentiable approximation to the agents' utilities [10, 19] and calculates gradients of strategies w.r.t. the utilities. However this requires learning accurate reward functions which is combinatorially hard for multi-resource coverage.

**Contributions:** To address the above challenge, we present a framework to tractably approximate a general class of spatial coverage objectives and their gradients via spatial discretization without having to learn neural network based reward models. We only discretize the target domain to represent integrals and all set operations over it, but not the allocated positions of the resources. Hence we

mitigate the intractability caused by discretizing allocations of large number of resources, while also keeping the allocations amenable to gradient-based optimization. We demonstrate successful application of our framework to multi-resource spatial coverage problems.

## 2 Multi-resource spatial coverage problems

In this section, we formally introduce the multi-resource allocation problem along with an example application domain, which will be used for evaluation.

**Multi-resource spatial coverage:** Spatial coverage problems comprise of a target space  $Q \subset \mathbb{R}^d$  (generally  $d \in \{2, 3\}$ ) and a set of  $m$  resources. **Action:** An action  $u \in \mathbb{R}^{m \times \hat{d}}$  is the placement of all  $m$  resources in an appropriate coordinate system of dimension  $\hat{d}$ . **Coverage:** When placed, each resource covers (often probabilistically) some part of the target space  $Q$ . Let  $\text{cvg} : q \times u \rightarrow \mathbb{R}$  be a function denoting the coverage of a target point  $q \in Q$  due to action  $u$ . We do not assume a specific form for the coverage  $\text{cvg}$  and leave it to be defined flexibly, to allow many different coverage applications to be amenable to our framework. **Reward:** The scalar reward due to action  $u$  is defined as:  $r(u) = \int_Q \text{cvg}(q, u) \text{imp}(q) dq$ , where  $\text{imp}(q)$  denotes the importance of the target point  $q$ . The objective is to optimize the placement reward  $r$  w.r.t. action  $u$ .

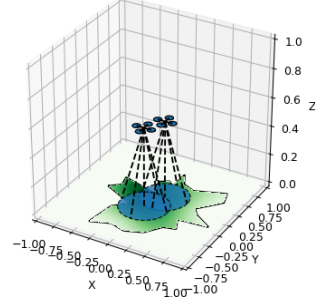


Figure 1: Areal surveillance with  $m = 2$  drones.

**Example 1 (Areal Surveillance).** An agent needs to allocate  $m$  areal drones with the  $i^{\text{th}}$  drone having three-dimensional coordinates  $u_i = (p_i, h_i) \in [-1, 1]^2 \times [0, 1]$  to surveil a two-dimensional forest  $Q \subset [-1, 1]^2$  of arbitrary shape and with a known but arbitrary tree density  $\rho(q)$ . Consequently,  $u \in \mathbb{R}^{m \times 3}$ . Each drone has a downward looking camera with a circular lens and with a half-angle  $\theta$  such that at position  $(p_i, h_i)$ , the drone  $i$  sees the set of points  $S_i = \{q \mid \|q - p_i\|_2 \leq h_i \tan \theta\}$ . A visualization of this problem with  $m = 2$  drones is shown for a sample forest in Figure 1. We assume a probabilistic model of coverage with a point  $q$  being covered by drone  $i$  with probability  $P_H(h_i) = e^{K(h_{\text{opt}} - h_i)} \left(\frac{h_i}{h_{\text{opt}}}\right)^{K h_{\text{opt}}}$  if  $q \in S_i$  and 0 otherwise. With multiple drones, the probability of a point  $q$  being covered can then be written as:  $\text{cvg}(q, u) = 1 - \prod_{i|q \in S_i} \bar{P}_H(h_i)$  where  $\bar{P}_H$  stands for  $1 - P_H$ . Hence, the reward function to be maximized is:  $r(u) = \int_Q \left(1 - \prod_{i|q \in S_i} \bar{P}_H(h_i)\right) \rho(q) dq$  with the tree density  $\rho(q)$  being the importance of target point  $q$ .

Note that drones provide best probabilistic coverage at a height  $h_{\text{opt}}$ . By increasing their height, a larger area can be covered at the cost of deterioration in coverage probability. Also coverage probability for regions with high tree density can be increased by placing multiple drones with overlapping views to oversee them; in which case, the drones can potentially stay at higher altitudes. Hence, this is a challenging domain with multiple trade-offs and complex combinatorial interactions between the resources. For experiments, we use the following constants:  $\theta = \frac{\pi}{6}$ ,  $h_{\text{opt}} = 0.2$ ,  $K = 4.0$ . However, note that these values only serve as practical representative values and the techniques introduced next apply to a broad class of coverage problems.

## 3 Methods

The key idea is to obtain a differentiable approximation to  $r(u)$  placement reward and then performing gradient ascent, thereby converging at a (locally) optimal value of  $u$ . We propose a tractable framework for this challenging task in the subsequent sub-sections.

### 3.1 Differentiable approximation for coverage objectives

Consider the objective:  $r(u) = \int_Q f(u, q) dq$  with  $u$  being the action and  $q$  being a point in the target domain  $Q$ . We assume that the action  $u$  has  $m$  components with  $u_i$  representing the location of  $i$ -th resource ( $i \in [m]$ ) and  $u_{\setminus i}$  representing the locations of all resources other than  $i$ . Note that the  $\text{imp}(q)$  function has been subsumed into  $f(u, q)$  in this formulation. We are interested in computing the gradient:  $\frac{\partial r}{\partial u_i}$ . However, this is a hard problem since: (a)  $r(u)$  involves integration over arbitrary (non-convex shaped) target domains which does not admit a closed-form expression in

terms of elementary functions and hence cannot be differentiated with autograd libraries like PyTorch and TensorFlow, and (b) most resources have a finite coverage area, outside of which the coverage drops to zero. This often makes the function  $f(u, q)$  discontinuous w.r.t.  $q$  given a fixed  $u$  especially at the coverage boundaries induced by the resources' coordinates, for e.g., drones have a circular probabilistic coverage area governed by their height and camera half-angle  $\theta$ , outside which the coverage probability suddenly drops to zero.

**Theorem 1.** *Let the objective function be  $r(u) = \int_Q f(u, q) dq$ . Denoting the set of points covered by the  $i$ -th resource as  $S_i$ , the interior of a set with  $\text{in}(\cdot)$  and the boundary with  $\delta(\cdot)$ , the gradient of  $r(u)$  w.r.t. the  $i$ -th resource's location  $u_i$  is given by:*

$$\frac{\partial r(u)}{\partial u_i} = \int_{\text{in}(Q \cap S_i)} \frac{\partial f(u, q)}{\partial u_i} dq + \int_{Q \cap \delta S_i} (f(u, q) - f(u_{\setminus i}, q)) \frac{\partial q_{Q \cap \delta S_i}}{\partial u_i} n_{q_{Q \cap \delta S_i}}^T dq \quad (1)$$

*Proof.* We defer the full proof to Section A.1 in the appendix and only mention the key steps involved here. To handle discontinuity across boundaries, we first split the integral into two parts - over the  $i$ -th resource's coverage area  $S_i$  and outside it. We further split the resulting integrals into their interiors and boundaries and perform differentiation using the Newton-Leibniz theorem. By applying geometric and topological relationships between the interior and the boundary, we simplify the resulting expressions to arrive at eq 1.  $\square$

The first term in eq 1 corresponds to the change in  $f$  inside the coverage area of resource  $i$  due to a small change in  $u_i$ , while the second term elegantly factors-in the effects of movement or shape change of the coverage area boundary due to changes in  $u_i$  (e.g. when a drone moves or elevates in height). While we show the general result here, the term  $\frac{\partial q_{Q \cap \delta S_i}}{\partial u_i} n_{q_{Q \cap \delta S_i}}^T$  can be simplified further using implicit differentiation of the boundary of  $S_i$ , which depends on the particular game under consideration. We show the simplification on our example domain in section A.2 in the appendix.

### 3.2 Discretization based approximation

While we now have a general form for  $r(u)$  and  $\frac{\partial r}{\partial u}$ , both forms comprise of non closed-form integrals over the target domain  $Q$  or its subsets. While evaluating  $r$  and  $\frac{\partial r}{\partial u}$  in practice, we adopt a discretization based approach to approximate the integrals. Given a target domain  $Q \subset \mathbb{R}^d$  with  $d \in \{2, 3\}$ , we discretize the full  $\mathbb{R}^d$  space into  $B_1, \dots, B_d$  bins respectively in each of the  $d$  dimensions. **Approximating spatial maps:** All spatial maps i.e. functions over the target domain  $Q$  (e.g.  $f(u, q)$ ), are internally represented as *real tensors* of dimension  $d$  with size:  $(B_1, \dots, B_d)$ . **Approximating sets:** All geometric shapes (or sets of points) including  $S_i$  for all resources (e.g., the circular coverage areas of drones) and the target domain  $Q$  itself (e.g., the irregular shaped forest) are converted to *binary tensors* each of dimension  $d + 1$  with size:  $(B_1, \dots, B_d, 3)$ . The final dimension of length 3 denotes interior, boundary and exterior of the geometric shape respectively, i.e. a binary tensor  $T$  has  $T_{b_1, \dots, b_d, 0} = 1$  if the bin at index  $(b_1, \dots, b_d)$  is inside the geometric shape,  $T_{b_1, \dots, b_d, 1} = 1$  if the bin is on the boundary of the geometric shape and  $T_{b_1, \dots, b_d, 2} = 1$  if the bin is outside the geometric shape. **Approximating operators:** The *binary tensors* associated with the  $\text{in}(\cdot)$  and the  $\delta(\cdot)$  operators are performed by our efficient divide-and-conquer shape discretizer, which is presented in section A.4 due to space constraints. The other set operations are approximated as follows: (a) set intersections are performed by element-wise *binary tensor* products, (b) integrals of spatial maps over geometric sets are approximated by multiplying (i.e. masking) the *real tensor* corresponding to the spatial map with the *binary tensor* corresponding to the geometric set followed by an across-dimension sum over the appropriate set of axes.

While our discretized bins growing exponentially with dimension  $d$  of the target domain may appear to be a limitation, our method still scales well for most real-world coverage problems since they reside on two or three-dimensional target domains. Further, unlike previous methods we only discretize the target domain but do not simultaneously restrict the action  $u$  to discrete bins. Hence, we do not run into intractability induced by discretizing high-dimensional actions due to multiple resources  $u$  is amenable to gradient-based optimization. Our proposed framework acts as an autograd module for  $r(u)$ , differentiable w.r.t. input  $u$ , and provides both the forward and the backward calls (i.e. evaluation and gradients).

## 4 Experiments

In our experiments, we differentially approximate rewards using the following variants: (a) feed-forward neural networks [*nn*], (b) graph neural networks [*gnn*], and (c) our differentiable coverage approximation [*diff*] for different values of  $m \in \{1, 2, 4\}$  over 5 different forest instances differing in shape and tree density. For the *nn* and *gnn* baselines, we trained neural networks, one per forest and per value of  $m$ , to predict the reward. The neural networks take as input the action  $u$  and output a prediction for the reward  $\hat{r}$ . Please see section A.3 in appendix for network architectures and hyperparameters. We use  $d = 2$  dimensional forests and discretize them into  $B_1 = B_2 = 200$  bins per dimension for a total of  $40K$  bins.

The maximum true reward  $r$  achieved by the three methods in all cases averaged over all the forest instances is summarized in Table 1. It is clear that *diff* always achieves the maximum true reward. While the difference from *nn* and *gnn* is less pronounced for  $m = 1$ , as the number of resources increases beyond 1, the approximation quality of *nn* and *gnn* deteriorates and the difference becomes very significant. This is also reflected in the plots of true reward achieved vs training iterations shown in Figure 2. Since *diff* is an unbiased approximator of the true reward<sup>1</sup>, the true reward continues to increase till convergence for *diff*. For *nn* and *gnn*, the true reward increases initially but eventually goes down as the action  $u$  begins to overfit the biased and potentially inaccurate approximations made by *nn* and *gnn*<sup>2</sup>. Figure 3 shows the final locations computed for a randomly chosen forest and with  $m = 2$  for all three methods.

Table 1: Maximum reward averaged across forest instances achieved for Areal Surveillance domain.

	$m = 1$	$m = 2$	$m = 4$
<i>diff</i> (ours)	<b>9366.03 ± 657.18</b>	<b>16091.09 ± 932.77</b>	<b>25117.98 ± 1554.34</b>
<i>nn</i>	9293.26 ± 646.37	14649.32 ± 1206.60	18962.87 ± 2018.54
<i>gnn</i>	9294.47 ± 664.28	14604.11 ± 1189.48	19353.93 ± 2701.81

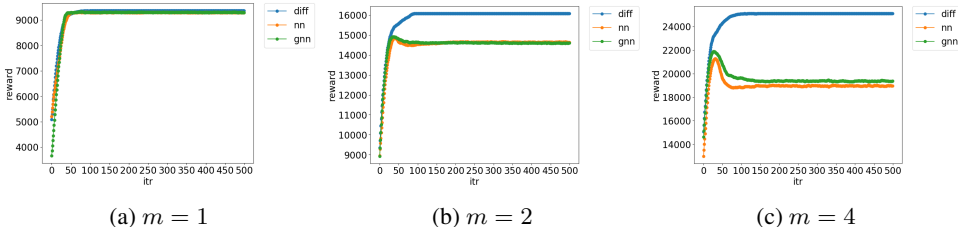


Figure 2: Plots of true reward achieved over DeepFP iterations by *diff*, *nn* and *gnn*.

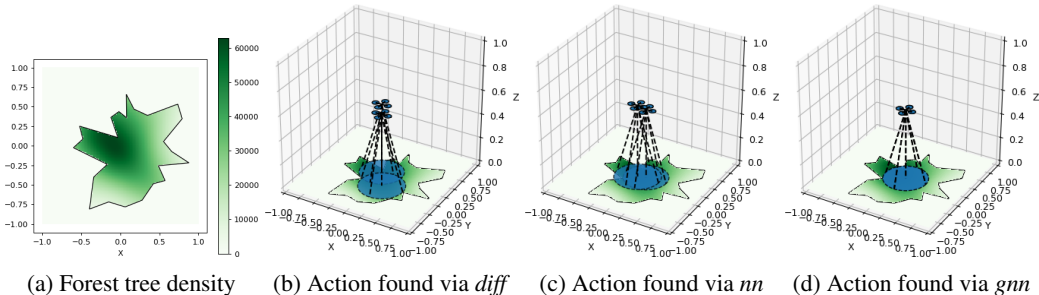


Figure 3: Visualizing final actions for a randomly chosen forest with  $m = 2$ .

## 5 Conclusion

In this work, we show that spatial coverage objectives with multiple-resources are combinatorially hard to approximate with neural networks. We propose to directly approximate a large class of multi-resource spatial coverage objectives and their gradients tractably without learning neural network based reward models. By employing our spatial discretization based approximation framework, we show improved performance in multi-resource spatial coverage problems.

<sup>1</sup>The only bias in *diff* is the discretization bin sizes, which can be made arbitrarily small in principle.

<sup>2</sup>Please see section A.3 in the appendix for a detailed analysis of this phenomenon.

## Broader Impact

Our work is broadly applicable for multi-resource coverage problems often arising in robotic surveillance, mobile sensor networks and security domains. We have provided an unbiased and differentiable approximation for coverage objectives which can be used for gradient-based learning. Our methods allow computing appropriate placement locations for resources such as drones, sensors or guards to defend crucial physical resources from unethical exploitation.

## Acknowledgments and Disclosure of Funding

This research was supported in part by NSF Research Grant IIS-1254206, MURI grant W911NF-11-1-0332 and USC Viterbi Graduate PhD fellowship.

## References

- [1] Kareem Amin, Satinder Singh, and Michael P Wellman. Gradient methods for stackelberg security games. In *UAI*, pages 2–11, 2016.
- [2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [3] Alireza Dirafzoon, Mohammad Bagher Menhaj, and Ahmad Afshar. Decentralized coverage control for multi-agent systems with nonlinear dynamics. *IEICE TRANSACTIONS on Information and Systems*, 94(1):3–10, 2011.
- [4] Jiarui Gan, Bo An, Yevgeniy Vorobeychik, and Brian Gauch. Security games on a plane. In *AAAI*, pages 530–536, 2017.
- [5] William Haskell, Debarun Kar, Fei Fang, Milind Tambe, Sam Cheung, and Elizabeth Denicola. Robust protection of fisheries with compass. In *IAAI*, 2014.
- [6] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer, 2002.
- [7] Taoan Huang, Weiran Shen, David Zeng, Tianyu Gu, Rohit Singh, and Fei Fang. Green security game with community engagement. *arXiv preprint arXiv:2002.09126*, 2020.
- [8] Matthew P. Johnson, Fei Fang, and Milind Tambe. Patrol strategies to maximize pristine forest area. In *AAAI*, 2012.
- [9] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI*, 2018.
- [10] Nitin Kamra, Umang Gupta, Kai Wang, Fei Fang, Yan Liu, and Milind Tambe. Deepfp for finding nash equilibrium in continuous action spaces. In *Decision and Game Theory for Security (GameSec)*, pages 238–258. Springer International Publishing, 2019.
- [11] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.
- [12] Chan Sze Kong, New Ai Peng, and Ioannis Rekleitis. Distributed coverage with multi-robot system. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2423–2429. IEEE, 2006.
- [13] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.
- [14] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

- [15] Ali Nasri Nazif, Alireza Davoodi, and Philippe Pasquier. Multi-agent area coverage using a single query roadmap: A swarm intelligence approach. In *Advances in practical multi-agent systems*, pages 95–112. Springer, 2010.
- [16] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Aria Nefian. Cooperative and distributed reinforcement learning of drones for field coverage. *arXiv preprint arXiv:1803.07250*, 2018.
- [17] Alessandro Renzaglia, Lefteris Doitsidis, Agostino Martinelli, and Elias B Kosmatopoulos. Multi-robot three-dimensional coverage of unknown areas. *The International Journal of Robotics Research*, 31(6):738–752, 2012.
- [18] Martin Saska, Jan Chudoba, Libor Přeučil, Justin Thomas, Giuseppe Loianno, Adam Třešňák, Vojtěch Vonásek, and Vijay Kumar. Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 584–595. IEEE, 2014.
- [19] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019.
- [20] Rong Yang, Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *AAMAS*, 2014.

## A Appendix

### A.1 Proof for the the coverage gradient theorem

In this section, we provide the detailed proof of Theorem 1.

*Proof.* While function  $f$  can be potentially discontinuous in  $q$  across resources' coverage boundaries,  $r(u)$  integrates over  $q \in Q$  thereby removing the discontinuities. Hence, instead of directly taking the derivative w.r.t. a particular resource's location  $u_i$  inside the integral sign, we split the integral into two parts - over the  $i$ -th resource's coverage area  $S_i$  and outside it:

$$r(u) = \int_{Q \cap S_i} f(u, q) dq + \int_{Q \setminus S_i} f(u, q) dq \quad (2)$$

Splitting the integral at the boundary of the discontinuity allows us to explicitly capture the effect of a small change in  $u_i$  on this boundary. Denoting the interior of a set with  $\text{in}(\cdot)$  and the boundary with  $\delta(\cdot)$ , the derivative w.r.t.  $u_i$  can be expressed using the Newton-Leibniz formula as:

$$\begin{aligned} \frac{\partial r(u)}{\partial u_i} &= \int_{\text{in}(Q \cap S_i)} \frac{\partial f(u, q)}{\partial u_i} dq + \int_{\delta(Q \cap S_i)} f(u, q) \frac{\partial q_{\delta(Q \cap S_i)}}{\partial u_i} n_{q_{\delta(Q \cap S_i)}}^T dq \\ &+ \int_{\text{in}(Q \setminus S_i)} \frac{\partial f(u \setminus i, q)}{\partial u_i} dq + \int_{\delta(Q \setminus S_i)} f(u \setminus i, q) \frac{\partial q_{\delta(Q \setminus S_i)}}{\partial u_i} n_{q_{\delta(Q \setminus S_i)}}^T dq, \end{aligned} \quad (3)$$

where  $\frac{\partial q_{\delta(Q \cap S_i)}}{\partial u_i}$  denotes the boundary velocity for  $\delta(Q \cap S_i)$  and  $n_{q_{\delta(Q \cap S_i)}}$  denotes the unit-vector normal to a point  $q$  on the boundary  $\delta(Q \cap S_i)$  (similarly for  $\delta(Q \setminus S_i)$ ). Since  $f(u \setminus i, q)$  does not depend on  $u_i$ , we can set  $\frac{\partial f(u \setminus i, q)}{\partial u_i} = 0$ . Next observe that the boundaries can be further decomposed as:  $\delta(Q \cap S_i) = (\delta Q \cap S_i) \cup (Q \cap \delta S_i)$  and similarly  $\delta(Q \setminus S_i) = (\delta Q \setminus S_i) \cup (Q \cap \delta S_i)$ . However since  $u_i$  does not change the boundary of the target domain  $\delta Q$ , we have:

$$\frac{\partial q_{\delta Q \cap S_i}}{\partial u_i} = 0, \quad \forall q \in \delta Q \cap S_i \quad (4)$$

$$\frac{\partial q_{\delta Q \setminus S_i}}{\partial u_i} = 0, \quad \forall q \in \delta Q \setminus S_i \quad (5)$$

Further on the boundary of  $S_i$ , the following unit-vectors normal to the boundary are oppositely aligned:

$$n_{q_{\delta(Q \setminus S_i)}} = -n_{q_{\delta(Q \cap S_i)}} \quad \forall q \in Q \cap \delta S_i. \quad (6)$$

Substituting the above results, we can simplify the gradient expression in eq 3 to:

$$\frac{\partial r(u)}{\partial u_i} = \int_{\text{in}(Q \cap S_i)} \frac{\partial f(u, q)}{\partial u_i} dq + \int_{Q \cap \delta S_i} (f(u, q) - f(u \setminus i, q)) \frac{\partial q_{Q \cap \delta S_i}}{\partial u_i} n_{q_{Q \cap \delta S_i}}^T dq \quad (7)$$

□

The first term in eq 1 corresponds to the change in  $f$  inside the coverage area of resource  $i$  due to a small change in  $u_i$ , while the second term elegantly factors-in the effects of movement or shape change of the coverage area boundary due to changes in  $u_i$  (e.g. when a drone moves or elevates in height). While we show the general result here, the term  $\frac{\partial q_{Q \cap \delta S_i}}{\partial u_i} n_{q_{Q \cap \delta S_i}}^T$  can be simplified further using implicit differentiation of the boundary of  $S_i$ , which depends on the particular game under consideration. We show the simplification on our example domain in section A.2.

### A.2 Implicit boundary differentiation for gradient simplification

As mentioned in section 3.1, the term  $\frac{\partial q_{Q \cap \delta S_i}}{\partial u_i} n_{q_{Q \cap \delta S_i}}^T$  from eq 1 can be simplified further using implicit differentiation of the boundary of  $S_i$ . In our example domains, the coverage boundaries

induced by all resources (drones) are circular. With the location of  $i$ -th drone as  $u_i = \{p_i, h_i\}$ , the boundary is given as:

$$\delta S_i = \{q \mid \|q - p_i\|_2 = h_i \tan \theta\}$$

Any point  $q \in Q \cap \delta S_i$  satisfies:

$$\|q - p_i\|_2 = h_i \tan \theta$$

Differentiating this boundary implicitly w.r.t.  $p_i$  and w.r.t.  $h_i$  gives:

$$\begin{aligned} \left( \frac{\partial q}{\partial p_i}^T - I_2 \right) \frac{q - p_i}{\|q - p_i\|_2} &= 0, \text{ and} \\ \frac{\partial q}{\partial h_i}^T \frac{q - p_i}{\|q - p_i\|_2} &= \tan \theta. \end{aligned}$$

Noting that the outward normal  $n_q$  at any point  $q \in Q \cap \delta S_i$  is given by  $\frac{q - p_i}{\|q - p_i\|_2}$ , we now have:

$$\begin{aligned} \frac{\partial q}{\partial u_i}^T n_q &= \left\{ \left( \frac{\partial q}{\partial p_i}^T n_q \right)^T, \frac{\partial q}{\partial h_i}^T n_q \right\} \\ &= \left\{ \left( \frac{q - p_i}{\|q - p_i\|_2} \right)^T, \tan \theta \right\} \end{aligned}$$

### A.3 Using neural networks to approximate multi-resource coverage objectives

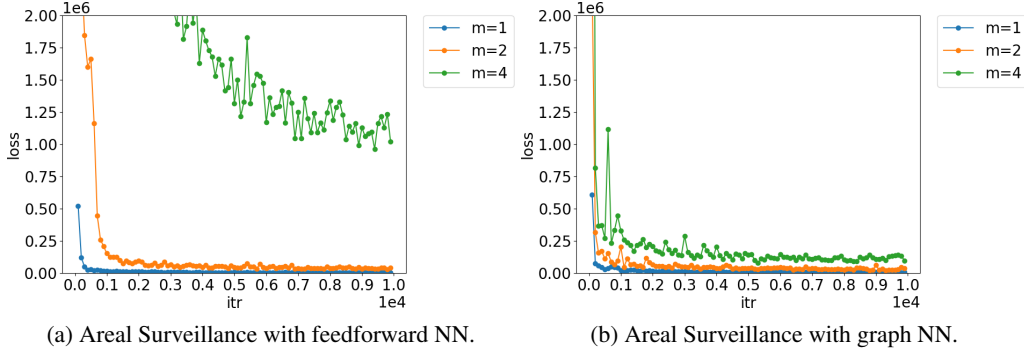


Figure 4: MSE for reward prediction with feedforward and graph neural networks.

We explore the efficacy of using neural networks to learn approximate differentiable models of coverage objectives. We trained neural networks, one per forest and per value of  $m$  to predict the reward for the forest domain. The neural networks take as input the action  $u$  and output a prediction for the reward  $\hat{r}$ . We trained all networks for 10,000 iterations with the Adam optimizer having learning rate 0.01 and a batch size of 64.

Figure 4 shows the corresponding curves of mean square error (MSE) loss vs. training iterations when the neural networks are strictly feedforward and also when we use graph neural networks [2], which treat each resource as a node in a graph and are much better at generalizing in combinatorial interaction settings. The network architectures used are shown in Table 2. We observe that while graph neural networks are better than feedforward networks in approximating the coverage rewards, both of them still suffer from high absolute values of MSE even after 10000 training iterations. The MSE is much higher when a larger number of resources are used e.g.,  $m = 4$  case (green lines) incur higher errors than  $m = 1$  case (blue lines) under all settings. This demonstrates that multi-resource coverage objectives are combinatorially hard to approximate even with graph neural networks, especially more so as the number of resources and consequently combinatorial interaction between them increases.



Table 2: Network architectures for reward models

Net type	Structure
<i>nn</i>	$\mathbb{R}^{m \times 3} \xrightarrow{fc, relu} \mathbb{R}^{128} \xrightarrow{fc, relu} \mathbb{R}^{512} \xrightarrow{fc, relu} \mathbb{R}^{128} \xrightarrow{fc, relu} \mathbb{R}^1$
<i>gnn</i>	$\mathbb{R}^{m \times 3}, -, - \xrightarrow{\substack{node\_enc \\ 3 \rightarrow 32}} \mathbb{R}^{32}, -, - \xrightarrow{\substack{edge\_net \\ 64 \rightarrow 16}} \mathbb{R}^{32}, \mathbb{R}^{16}, -$ $\xrightarrow{\substack{node\_net \\ 48 \rightarrow 32}} \mathbb{R}^{32}, \mathbb{R}^{16}, - \xrightarrow{\substack{glob\_net \\ 48 \rightarrow 16}} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16}$ $\xrightarrow{\substack{edge\_net \\ 96 \rightarrow 16}} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16} \xrightarrow{\substack{node\_net \\ 64 \rightarrow 32}} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16}$ $\xrightarrow{\substack{glob\_net \\ 64 \rightarrow 1}} \mathbb{R}^1$

#### A.4 Divide and conquer based shape discretizer

The python pseudo-code for the discretizer is shown below and makes use of a recursive geometric map-filling method which uses divide and conquer to efficiently compute the interior, exterior and boundary of any geometric shape stored in the *Shapely* geometric library format. Note that a minimal functional pseudo-code using *Numpy* has been presented here to facilitate understanding. Our actual code is more complex and allows working with PyTorch tensors on both CPU and GPU while also supporting batches of geometric objects. We also have other specialized versions (not shown here) which work faster for circular geometries.

```

import numpy as np
from shapely.geometry import Polygon, Point

def get_g_map(geom, lims, deltas):
    ''' Computes the geometric maps from geometry.
    Args:
        geom: Shapely geometry object
        lims: Tuple (x_min, x_max, y_min, y_max) for generated
              geometric map
        deltas: Discretization bin size; tuple (delX, delY)

    Returns:
        g_map: numpy.ndarray of shape (nbinsX, nbinsY, 3)
              containing (interior, boundary, exterior) indicator of
              geometry in the third dimension.
    '''
    x_min, x_max, y_min, y_max = lims
    delX, delY = deltas
    nbinsX = round((x_max - x_min) / delX)
    nbinsY = round((y_max - y_min) / delY)

    g_map = np.zeros((nbinsX, nbinsY, 3)) # (int, bound, ext)
    fill(geom, g_map, 0, nbinsX, 0, nbinsY, lims, deltas)
    return g_map

def fill(geom, g_map, i1, i2, j1, j2, lims, deltas):
    ''' Fills g_map of shape (nbinsX, nbinsY, 3) with 1s at
        appropriate locations to indicate interior, exterior and
        boundary of the shape geom. This method makes recursive
        calls to itself and fills up the g_map tensor in-place.
    '''

```

```

Args:
    geom: A shapely.geometry object, e.g. Polygon
    g_map: A numpy.ndarray of shape (nbinsX, nbinsY, 3)
    i1: left x-coord of recursive rectangle to check against
    i2: right x-coord of recursive rectangle to check against
    j1: bottom y-coord of recursive rectangle to check against
    j2: top y-coord of recursive rectangle to check against
    lims: Tuple (x_min, x_max, y_min, y_max) for generated
          geometric map
    ,,,
    deltas: Discretization bin size; tuple (delX, delY)

x_min, x_max, y_min, y_max = lims
delX, delY = deltas

box = Polygon([(x_min + i1*delX, y_min + j1*delY), \
               (x_min + i2*delX, y_min + j1*delY), \
               (x_min + i2*delX, y_min + j2*delY), \
               (x_min + i1*delX, y_min + j2*delY)])

if box.disjoint(geom):
    g_map[i1:i2, j1:j2, 2] = 1.0
elif box.within(geom):
    g_map[i1:i2, j1:j2, 0] = 1.0
else: # box.intersects(geom)
    if (i2 - i1 <= 1) and (j2 - j1 <= 1):
        g_map[i1:i2, j1:j2, 1] = 1
    elif (i2 - i1 <= 1) and (j2 - j1 > 1):
        j_mid = (j1 + j2) // 2
        fill(geom, g_map, i1, i2, j1, j_mid, lims, deltas)
        fill(geom, g_map, i1, i2, j_mid, j2, lims, deltas)
    elif (i2 - i1 > 1) and (j2 - j1 <= 1):
        i_mid = (i1 + i2) // 2
        fill(geom, g_map, i1, i_mid, j1, j2, lims, deltas)
        fill(geom, g_map, i_mid, i2, j1, j2, lims, deltas)
    else: # (i2 - i1 > 1) and (j2 - j1 > 1):
        i_mid = (i1 + i2) // 2
        j_mid = (j1 + j2) // 2
        fill(geom, g_map, i1, i_mid, j1, j_mid, lims, deltas)
        fill(geom, g_map, i_mid, i2, j1, j_mid, lims, deltas)
        fill(geom, g_map, i1, i_mid, j_mid, j2, lims, deltas)
        fill(geom, g_map, i_mid, i2, j_mid, j2, lims, deltas)

```