Multi-Agent Area Coverage Control Using Reinforcement Learning

Adekunle Adepegba, Suruz Miah, Davide Spinello
Summarized by Simon Hu

November 21, 2019

Outline

Problem Statement

- Given a group of N homogeneous agents moving in an environment $\Omega \subset \mathbb{R}^2$, find a configuration $p = (p_1, p_2, \ldots, p_N) \in \mathbb{R}^{2^N}$ of these agents that maximizes

$$\mathcal{H}(p,t) = \int_{\Omega} \max_{i=1,2,\ldots,N} f_i(\|q - p_i\|)\phi(q,t)\ \mathrm{d}q. \tag{1}$$

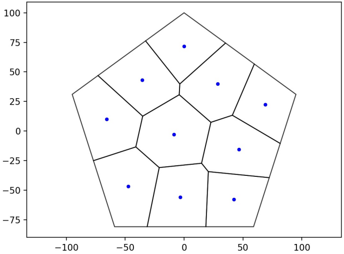
- $f_i : \Omega \to \mathbb{R}_{\geq 0}$ is Lesbegue measurable, non-increasing, and is a function of the distance between p_i and q. We will consider $f(x) = -x^2$.
- $\phi : \Omega \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is the *risk density*. It measures our belief in an event happening at point q at time t.
- Applications: harbor patrol, search and rescue operations, ocean data collection.

Voronoi Tesselations

- A collection $S = (S_1, S_2, \ldots, S_M)$ is a partition of Ω if each pair of S_i have disjoint interior and their union is Ω.
- The Voronoi partition is a particular partition given by $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_L)$ where

$$\mathcal{V}_i = \left\{ q \in \Omega \mid \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i \right\}.$$

- $c_{\mathcal{V}_i}$ and $m_{\mathcal{V}_i}$ are the centroid and mass of the i-th cell.



Voronoi Tesselations

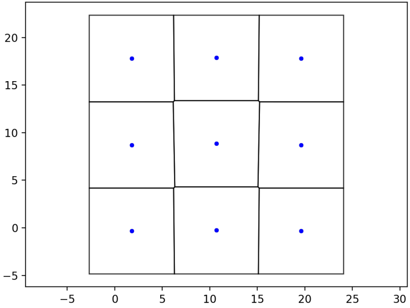
- If we partition Ω according to $\mathcal{V}$ we can rewrite the cost as

$$\mathcal{H}(p,t) = \sum_{i=1}^{N} \int_{\mathcal{V}_i} f_i(\|q - p_i\|)\phi(q,t)\ \mathrm{d}q.$$

- Each agent only needs information from its *neighbors* to compute its contribution to the global cost. Any algorithm taking advantage of this is *spatially distributed*.

Optimal Strategy

- (Cortés et al. 2004). The control strategy where each agent moves towards the centroid of its Voronoi cell locally solves the area coverage control problem.



Reinforcement Learning Approach

- Let $e_i = c_{\mathcal{V}_i} - p_i$. The authors consider the value function

$$V(e_i) = \sum_{\kappa=k}^{\infty} e_i^T Q e_i + u_i^T R u_i$$

  where $Q, R \in \mathbb{R}^{2\times 2}$ are positive definite.
- Perform the standard trick to write this as the HBJ

$$V^*(e_i) = \min_{u_i} \left[e_i^T Q e_i + u_i^T R u_i + V^{next}(e_i) \right].$$

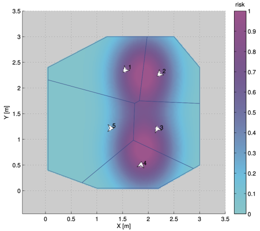
- Now we approximate the Value function $V(e_i)$ and policy u_i as neural networks.

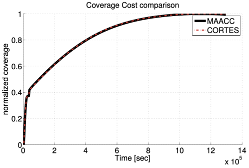
Soft Actor-Critic Method

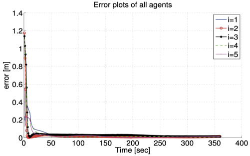
- The authors basically use the SAC DDPG algorithm we implemented in homework 4, with slight modifications.
  - They use least-squares loss instead of MSE.
  - Soft updates don't use the update rule

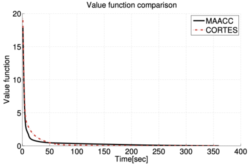
$$w_{target} \leftarrow \alpha w_{target} + (1 - \alpha) w_{source} \tag{2}$$

  but use the weights from the policy networks to update the weights from the Value function network.
- Since they use least-squares loss, they actually compute the analytical form, which involves taking inverses.
- We will use the update rule (2), use MSE loss, and actually implement the network.


risk
X [m]
Y [m]


Coverage Cost comparison
MAACC
CORTES
normalized coverage
Time [sec]
x 10^5


Error plots of all agents
i=1
i=2
i=3
i=4
i=5
error [m]
Time [sec]


Value function comparison
MAACC
CORTES
Value function
Time[sec]

Good, Bad, and the Ugly

- Good:
  - RL methods are robust to changes in the environment. In many applications your environment is dynamic.
  - PD Feedback Controllers are OK for handling changes but may not be good at learning the dynamics of the environment.
- Bad:
  - You can just implement a really finely tuned PD controller to solve this problem.
- Ugly:
  - Each agent keeps track of its OWN target and source, actor and critic networks. This is not very scalable!