Team exploration and task allocation on graphs using spectral decomposition and Deep Reinforcement Learning

Klinsmann Agyei
Faculty of Control and Robotics
ITMO University
Saint Petersburg, Russia
klinsmannjurgenagyei@gmail.com

***Abstract*—Robots were originally built to improve the quality of lives of humans by automating tasks that are dangerous, repetitive or tedious for humans [6]. These tasks may include assembling of parts of complex machines like cars, sorting of products in a warehouse, assisting in household chores and so on [4]. While the use of multiple robots has shown promise in improving efficiency and achieving overall objectives, challenges such as coordination and communication arise [12]. Reinforcement learning is considered a better option than classical control algorithms, but it also faces challenges such as task allocation, planning and execution, and training time and power [2]. This paper introduces a novel algorithm that integrates spectral graph decomposition with deep reinforcement learning. The algorithm demonstrates improved performance by eliminating certain termination conditions, such as agents colliding with each other, agents occupying the same position, agents exploring the same position by different agents and many others.**

I. Introduction

Reinforcement Learning has in the past shown some promising prospect in enhancing the future of robot learning with past data in a form of observation and rewards. The goal is to design systems that use richly structured perception, perform planning and control that adequately adapt to environmental changes [3]. In reinforcement learning, robots or agents traverse their environment using three key representations: state (s), reward (r), and action (a) [16] with the set of all actions and states represented as $\mathcal{A}$ and $\mathcal{S}$ respectively. The agent assesses its environment and determines its current state s_t at timestamp t, and then takes an action a_t and repeats the same steps till its goal is achieved as shown in equation 1. The agent makes decisions on the actions to be taken through a policy π, commonly referred to as the agent's brain. This policy can be either deterministic, as illustrated in equation 1, where all actions are predetermined with the same action for each state, or stochastic, where agents select different actions for the same state, with each action associated with a probability, as illustrated in equation 2.

$$\pi(s) \in \mathcal{A}(s) \ \forall \ s \in \mathcal{S} \tag{1}$$

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s) \ \forall \ s \in \mathcal{S} \ and \ a \ \in \mathcal{A}(s) \tag{2}$$

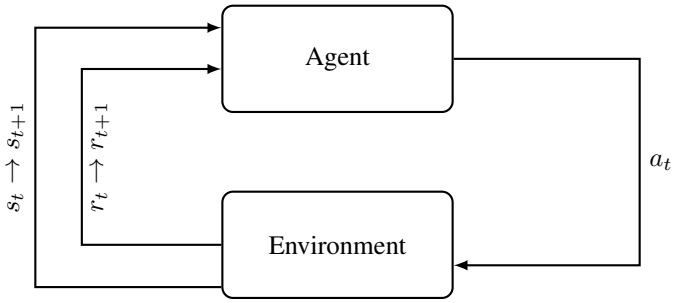

Fig. 1. Illustration of Reinforcement Learning with an Agent Interacting with Its Environment.

Training an agent to find the best solution to problems can be done in two main ways, policy-based methods trains the policy function directly by learning the policy by mapping each state to action [14]. The value-based method is designed to learn and estimate the value of a state-action pair. In other words, it effectively maps each state to its expected value. The value of a state is the expected discounted return that the agent can achieve if it initiates in that particular state and subsequently follows our policy as illustrated in equation I .

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 + R_{t+2} \ldots | S_t = s] \tag{3}$$

A. Multi-Robot System

Multi-Robot Systems are one of the core subfields of robotics, with a wide range of applications. These applications span from construction management to Multi-Robot Factories, where multiple robotic agents collaborate and coordinate to achieve a common goal. They are also employed in Cyber-Physical Systems, where each subsystem of the cyber-physical system acts as an agent, contributing to its overall functionality. The versatility of Multi-Agent Systems is evident across various domains [9]. Multi-Agent Systems consist of autonomous entities, commonly known as agents, that collaborate with each other and their environments to achieve a common or individual goal [5]. The system utilizes data from interactions among agents and the environment to make future decisions or take actions that subsequently solve the assigned task. A Multi-Agent System is more flexible and efficient because tasks are divided among various agents. It is also more reliable, as the task can still be accomplished even in the event of a failure on the part of an agent.

1) Graphs

In networked and multi agents systems, agents and their relationship are simulated using graphs [21]. Graphs shows how information are shared between these agents [13]. In the context of a networked and multi-agent system, the transfer of information relies on two primary components within a graph simulation: vertices, which symbolize agents, and edges, which represent connections through which information is transferred [20]. A graph is constructed with a finite number of vertices and edges, where the set of vertices is denoted by

$$V = \{v_1, v_2, ..., v_n\}$$

The edges are the connections between two vertices, as illustrated in 2, and denoted by

$$E = \{v_1v_2, v_2v_3....v_iv_j\}$$

A graph is denoted by

$$\mathcal{G} = (V, E)$$

These two primary components can be modelled under directed or undirected graphs [18].

Directed Graph A directed graph is a type of graph in which the edges have a specific direction, indicating the path of information flow. In the context of networked and multi-agent systems, the information flow between agents follows a specified path determined by the directed graph [17]. A directed graph, $\mathcal{G} = (V, E)$ is a type of graph that consists of a nonempty finite set of nodes, denoted by, $V = \{1, ..., v_n\}$ and a set of ordered pairs of nodes, denoted by $E \subseteq V \times V$, that represent the direction of the edges. A directed graph is deemed "strongly connected" when every edge follows a specified path, or information flow between nodes adheres to a specified route. A node is designated as a root if it possesses a directed path to the other nodes without having a parent itself [13]. A rooted directed tree is a directed graph wherein, with the exception of one node, all other nodes should have precisely one parent node.

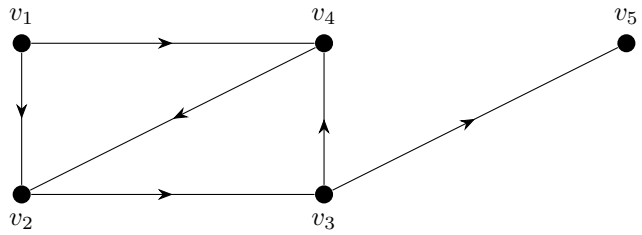

Fig. 2. Directed Graph

Undirected Graph An undirected graph has no direction. Thus, there is no specific direction to the flow of information between agents (vertices) hence the edges possessing no direction [10]. An undirected graph may be regarded as a specialized form of a directed graph, wherein unordered pairs of nodes are permissible [8]. The edge $(i, j) \in E$ is denoted as an interaction between agents i and j, allowing them to exchange information. Consequently, in the directed graph, both edges (i, j) and (j, i) correspond to a single edge (i, j) in the undirected graph [15].

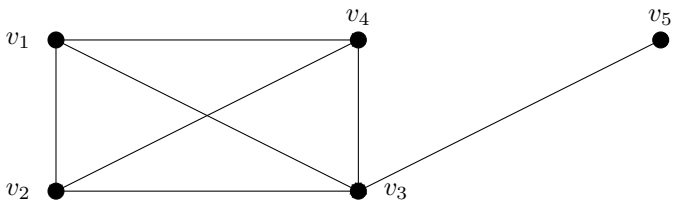

Fig. 3. Undirected Graph

2) Matrices

One of the most commonly employed methods for representing a graph is through the use of matrices [13].

a) Adjacence Matrix

Let $\mathcal{G}$ be an undirected graph with n vertices. The adjacency matrix A is given by:

$$A(\mathcal{G}) = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

where $a_{ij} = 1$ if there is an edge between vertices i and j, and $a_{ij} = 0$ otherwise. It shows the adjacency relationship in the

graph [1]. Returning to Figure 3, the corresponding adjacency matrix is

$$A(\mathcal{G}) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

II. Proposed Method

A. Problem Formulation

Consider a problem involving multiple robots for search and rescue and task allocation. In this scenario, a group of robots enters an unknown 2D rectangular environment with an area denoted as $\mathcal{A} \in \mathbb{R}^2$. The environment is bounded by a fence $\mathcal{F} = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)\}$. The main objective is for the robots to efficiently detect the entire environment in order to search for victims and rescue them as quickly as possible. Each robot has the freedom to move within the environment.

A novel graph-based spectral clustering algorithm is proposed, coupled with deep reinforcement learning algorithms, to address the problem. The approach is divided into two parts: firstly, the environment topography is represented as a graph $\mathcal{G} = (V, E)$ as illustrated in Figure 5 where each node represents a distinct location, region or object in the environment and the edges represent the connection or relationship between the nodes, and a spectral decomposition algorithm is proposed to partition among the robots or agents. In the second part, the environment is explored, and agents discover victims using deep reinforcement learning techniques.

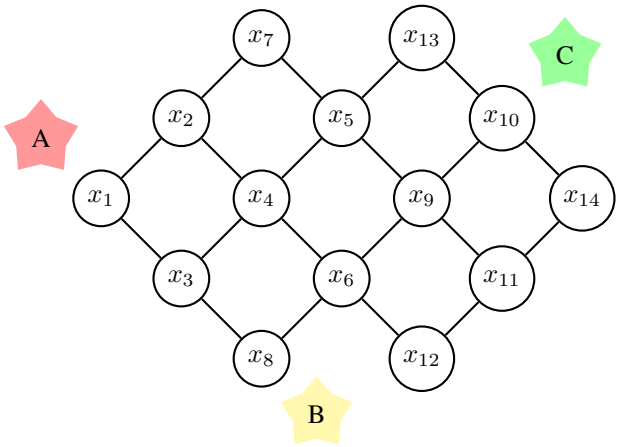

Fig. 4. A graph topology representation of an environment with three robots A, B and C

B. Spectral Decomposition of Environment

The Spectral Decomposition algorithm utilizes the relationship between eigenvalues and eigenvectors of matrices associated with the graph [11], such as the adjacency matrix or the Laplacian matrix to partition a graph in k mutually disjoint, connected parts [19]. This paper propose three steps to perform spectral decomposition of a graph, The eigenvalues, $\lambda(A(\mathcal{G}))$ and eigenvectors, $v(A(\mathcal{G}))$ of the adjacency matrix or the Laplacian matrix of the graph is computed. These matrices capture the connectivity and structure of the graph. Second, the eigenvalues are arranged in descending order, eigenvectors corresponding to the largest k eigenvalues are selected, where k is the number of agents or robots. These eigenvectors are called the Fiedler vectors, and they reveal the hidden clusters or communities in the graph Third, each vertex of the graph is assigned to a cluster based on the sign or value of the corresponding entry in the Fiedler vector. For example, if the Fiedler vector is [0.1, -0.1, 0.1, -0.1], then we can assign the vertices with positive entries to one cluster, and the vertices with negative entries to another cluster. This way, we obtain a partition of the graph that minimizes the number of edges between the clusters

Algorithm 1 Spectral Decomposition of the Environment

Input: Graph G represented by its adjacency matrix
Output: Clusters representing the partition of the graph
Step 1: Compute the Adjacency matrix
$L \leftarrow$ ComputeAdjacencyMatrix(G)
Step 2: Compute the eigenvalues and eigenvectors of the Laplacian matrix
$(\lambda, V) \leftarrow$ ComputeEigenvaluesAndEigenvectors(L)
Step 3: Sort the eigenvalues and corresponding eigenvectors
SortEigenvaluesAndEigenvectors(λ, V)
Step 4: Choose the k smallest eigenvectors corresponding to the k smallest eigenvalues
$k \leftarrow$ DetermineNumberOfClusters()
$V_k \leftarrow$ SelectKSmallestEigenvectors(λ, V, k)
Step 5: Cluster the data points
Clusters $\leftarrow$ ClusterDataPoints(V_k)
Return Clusters =0

Remark 1 (Algorithm 1 explained)
Consider a graph $\mathcal{G}$ with adjacency matrix A($\mathcal{G}$). Use numerical techniques to compute the eigenvalues and eigenvectors of the adjacency matrix. Sort Eigenvalues and Eigenvectors (Step 3): Sort the eigenvalues in descending order and rearrange the corresponding eigenvectors accordingly. Choose Eigenvectors (Step 4): Choose the k largest eigenvectors corresponding to the k largest eigenvalues. The number of eigenvectors to choose (k) is often determined based on heuristics or by corresponding it with the number of robots. Step 5): Apply clustering algorithms like k-means clustering to the selected eigenvectors to partition the graph into clusters.

To illustrate, consider the environment graph depicted in Figure 5. After sorting the eigenvalues and eigenvectors in descending order of magnitude, we observe that the three largest eigenvalues are 2, 2, and 1.618. Subsequently, the corresponding eigenvectors are
$v_1 = [-0.250, 0.000, -0.250, 0.000, -0.250, 0.000, -0.250, 0.000, -0.250, 0.000, -0.250, 0.000, -0.250, 0.000]$

$v_2 = [0.000, 0.225, 0.000, 0.225, 0.000, 0.225, 0.000, 0.225, 0.000, -0.225, 0.000, -0.225, 0.000, 0.225]$

$v_3 = [0.000, 0.225, 0.000, -0.225, 0.000, 0.225, 0.000, -0.225, 0.000, -0.225, 0.000, 0.225, 0.000, 0.225]$

Each vertex was assigned to a cluster based on the sign of the corresponding entries in the eigenvectors. In this case, if the entries are positive, negative, and positive, then the vertex belongs to cluster 1. If the entries are negative, positive, and negative, then the vertex belongs to cluster 2. If the entries are negative, negative, and positive, then the vertex belongs to cluster 3. This way, partition was done as illustrated in 5

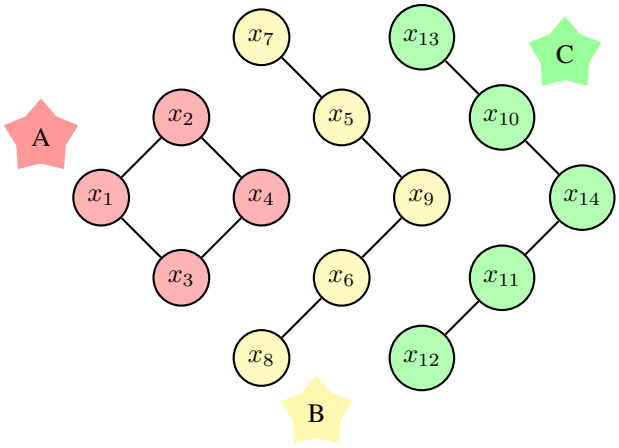

Fig. 5. Clustered graph with k=3 representing three agents, A, B and C

C. Reinforcement Learning

Reinforcement Learning (RL) is an effective tool for training agents to traverse through a graph achieves to achieve better performance to traditional graph traversal algorithms [7]. Proximal Policy Optimization algorithm is used because it strikes a balance between important factors like ease of implementation, ease of tuning, sample complexity,sample efficiency and trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small. The algorithm was implemented using the actor-critic model which uses two deep neural networks, one taking the action(actor) and the other handles the rewards(critic). Consider a multi agent actor-critic model on graph $\mathcal{G}$. In each step or node traversal, there is an update to an existing policy to seek improvement on certain parameters. It ensures that the update is not too large, that is the old policy is not too different from the new policy, it does so by essentially "clipping" the update region to a very narrow range as illustrated in 4 where θ is the policy parameter, $\hat{\mathbb{E}}_t$ is the empirical expectation over time, r_t is the importance sampling ratio, $\hat{A}_t$ is the estimated advantage at time t and ϵ is a hyperparameter. Advantage function is the difference between the future discounted sum of rewards on a certain state and action, and the value function of that policy. Importance Sampling ratio, or the ratio of the probability under the new and old policies respectively, is used for update is a hyperparameter denotes the limit of the range within which the update is allowed

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{4}$$

where,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Algorithm 2 Proximal Policy Optimization (PPO)

Input: Policy network π_θ, Value network V_ϕ
Parameters: Number of epochs N, Number of steps per epoch T, Batch size B, Learning rate α, Discount factor γ, Clipping parameter ϵ
Initialize θ, ϕ randomly
for $n = 1$ **to** N **do**
  for $t = 1$ **to** T **do**
    Generate trajectory $\tau = \{(s_i, a_i, r_i)\}_{i=1}^{T}$ using π_θ
    Compute advantages $A(s_i, a_i) = \sum_{t'=t}^{T} (\gamma^{t'-t} r_{t'} - V_\phi(s_{t'}))$
    Compute target value $V_t^{target} = \sum_{t'=t}^{T} (\gamma^{t'-t} r_{t'})$
  end for
  Update value network parameters ϕ by minimizing the loss: $\frac{1}{B} \sum_{b=1}^{B} (V_\phi(s_b) - V_t^{target})^2$
  for $t = 1$ **to** T **do**
    Compute surrogate loss:
    $L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\hat{A}_t ht\right)\right.\right.$
Update policy network parameters θ by minimizing the surrogate loss: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{CLIP}(\theta)$
  end for
end for=0

Remark 2 (Algorithm 2 explained)
The algorithm takes policy network π_θ and the value network V_ϕ, along with various parameters such as the number of epochs, steps per epoch, batch size, learning rate, discount factor, and clipping parameter as input. The policy and value network parameters (θ and ϕ) are initialized randomly.The algorithm iterates over a fixed number of epochs. Within each epoch, it generates trajectories using the policy network and computes advantages and target values based on the observed rewards and estimated values.The value network parameters

(ϕ) are updated by minimizing the mean squared error between the predicted values and the target values.The policy network parameters (θ) are updated by minimizing a surrogate loss function, which is clipped to ensure that the updated policy remains close to the old policy. This helps stabilize training and prevent large policy updates

III. Numerical Results

In this section, we evaluate the exploration and task allocation performance of the overall graph and the clustered or spectrally decomposed graphs based on the time required to explore the entire graph or environment. This comparative analysis offers insights into scalability and performance. The experiment was conducted on a system equipped with an Intel(R) Core(TM) i5-7200U CPU running at 2.50GHz with 8GB of RAM.

To construct the graphs, a random graph generator was employed to create environment graphs with varying numbers of nodes and edges. The search time for both types of graphs were recorded (see Table I). Each experiment trial was repeated five times to ensure statistical significance.

TABLE I
COMPARISON OF OG WITH DRL AND SG WITH DRL

Number of nodes	OG with DRL	SG with DRL
	Multi-Search time	Multi-Search time
10	0.132 ± 0.018	0.087 ± 0.011
20	1.925 ± 0.045	0.436 ± 0.066
30	2.718 ± 0.072	0.785 ± 0.121
40	3.511 ± 0.099	1.134 ± 0.176
50	4.304 ± 0.126	1.483 ± 0.231

IV. Conclusion

In this paper, we present a novel graph decomposition algorithm combined with a deep reinforcement learning model for multi-agent exploration and task allocation. The algorithm decomposes the environment into a number of clusters equal to the number of agents, facilitating exploration, followed by utilizing deep reinforcement learning to navigate the decomposed graph. This approach has demonstrated greater efficiency compared to general exploration and task allocation using deep reinforcement learning. In future research, we intend to consider the individual strengths of each agent and devise strategies for reallocating tasks among agents who complete their exploration ahead of others.

References

[1] Qasim Ali and Sergio Montenegro. Role of graphs for multi-agent systems and generalization of euler's formula. In *2016 IEEE 8th International Conference on Intelligent Systems (IS)*, pages 198–204, 2016.

[2] R Babuška, L Busoniu, and B De Schutter. Reinforcement learning for multi-agent systems. Technical report, Technical Report 06-041, Delft Center for Systems and Control, Delft ..., 2006.

[3] George A Bekey. Springer handbook of robotics (b. siciliano and o. khatib; 2008)[book review]. *IEEE Robotics & Automation Magazine*, 15(3):110–110, 2008.

[4] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444, 2022.

[5] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.

[6] Martin Hägele, Klas Nilsson, J Norberto Pires, and Rainer Bischoff. Industrial robotics. *Springer handbook of robotics*, pages 1385–1422, 2016.

[7] Patrick Hart and Alois Knoll. Graph neural networks and reinforcement learning for behavior generation in semantic environments. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1589–1594. IEEE, 2020.

[8] Wei Jiang, Yiyang Chen, and Themistoklis Charalambous. Consensus of general linear multi-agent systems with heterogeneous input and communication delays. *IEEE Control Systems Letters*, 5(3):851–856, 2020.

[9] Vicente Julian and Vicente Botti. Special issue on multi-agent systems. *Applied Sciences*, 13(2), 2023.

[10] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[11] James B Kennedy, Pavel Kurasov, Corentin Léna, and Delio Mugnolo. A theory of spectral partitions of metric graphs. *Calculus of Variations and Partial Differential Equations*, 60:1–63, 2021.

[12] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin, and Birgitta Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

[13] Mehran Mesbahi and Magnus Egerstedt. *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010.

[14] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[15] Mariá C.V. Nascimento and André C.P.L.F. de Carvalho. Spectral methods for graph clustering – a survey. *European Journal of Operational Research*, 211(2):221–231, 2011.

[16] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 50(9):3826–3839, 2020.

[17] Meng; Mesbahi Mehran; Egerstedt Magnus Rahmani, Amirreza; Ji. Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization 2009-jan vol. 48 iss. 1*, 48, jan 2009.

[18] Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[19] Liu Yong, Guo Gencheng, and Qi Jingjing. An algorithm of system decomposition based on laplace spectral graph partitioning technology. In *2008 International Conference on Computer Science and Software Engineering*, volume 2, pages 85–89. IEEE, 2008.

[20] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[21] Pu Zhang, Huifeng Xue, Shan Gao, and Jialong Zhang. Distributed adaptive consensus tracking control for multi-agent system with communication constraints. *IEEE Transactions on Parallel and Distributed Systems*, 32(6):1293–1306, 2020.