Challenges and Opportunities in Deep Reinforcement Learning With Graph Neural Networks: A Comprehensive Review of Algorithms and Applications

Sai Munikoti, Member, IEEE, Deepesh Agarwal, Laya Das, Mahantesh Halappanavar, Senior Member, IEEE, and Balasubramaniam Natarajan, Senior Member, IEEE

Abstract—Deep reinforcement learning (DRL) has empowered a variety of artificial intelligence fields, including pattern recognition, robotics, recommendation systems, and gaming. Similarly, graph neural networks (GNNs) have also demonstrated their superior performance in supervised learning for graph-structured data. In recent times, the fusion of GNN with DRL for graph-structured environments has attracted a lot of attention. This article provides a comprehensive review of these hybrid works. These works can be classified into two categories: 1) algorithmic contributions, where DRL and GNN complement each other with an objective of addressing each other’s shortcomings and 2) application-specific contributions that leverage a combined GNN-DRL formulation to address problems specific to different applications. This fusion effectively addresses various complex problems in engineering and life sciences. Based on the review, we further analyze the applicability and benefits of fusing these two domains, especially in terms of increasing generalizability and reducing computational complexity. Finally, the key challenges in integrating DRL and GNN, and potential future research directions are highlighted, which will be of interest to the broader machine learning community.

Index Terms—Deep learning, deep reinforcement learning (DRL), graph neural network (GNN), hybrid DRL–GNN, survey.

NOMENCLATURE

- $x$: State of a Markov decision process.
- $A$: Action space of Markov decision process.
- $r$: Reward in a Markov decision process.
- $S$: Solution set.
- $\Pi$: Policy.
- $G$: Graph.
- $\Upsilon$: Node set.
- $E$: Edge set.
- $u$: Node.
- $A$: Adjacency matrix.
- $\epsilon$: Exploration rate.
- $T$: Time.
- $p$: Transition probability.
- $\Omega$: Observation probability.
- $a$: Action.
- $E$: Expectation.
- $\gamma$: Discount factor.
- $\theta$: Learning parameters.
- $J$: Cost function.
- $h$: Embedding vector.
- $MDP$: Markov decision process.
- $DRL$: Deep reinforcement learning.
- $GNN$: Graph neural network.
- $DNN$: Deep neural network.
- $RL$: Reinforcement learning.
- $DQN$: Deep Q-learning.
- $GAT$: Graph attention network.
- $RGCN$: Relational graph convolutional network.
- $RDRL$: Relational deep reinforcement learning.
- $MTDRL$: Multitask deep reinforcement learning.
- $CTDE$: Centralized training and decentralized execution.
- $POMDP$: Partially observed Markov decision process.
- $PPO$: Proximal policy optimization.
- $TRPO$: Trust region policy optimization.
- $DDPG$: Deep deterministic policy gradient.
- $GCN$: Graph convolutional network.
- $NAS$: Neural architecture search.
- $NIPA$: Node injection poisoning attack.
- $MADRL$: Multiagent deep reinforcement learning.
- $CO$: Combinatorial optimization.
- $S2V$: Structure to vector.
- $VRP$: Vehicle routing problem.
- $A2C$: Advantage actor–critic.
- $KG$: Knowledge graph.
Table of acronyms:

| Acronym | Description                  |
|---------|------------------------------|
| KGC     | Knowledge graph completion.  |
| GAN     | Generative adversarial network. |
| XAI     | Explainable artificial intelligence. |
| GPU     | Graphic processing unit.     |
| PGNN    | Position-aware graph neural network. |
| IaGNN   | Identity-aware graph neural network. |
| NLP     | Natural language processing. |
| CV      | Computer vision.             |
| RDDL    | Relational dynamic influence diagram language. |
| MVC     | Maximum vertex cover.        |
| TSP     | Traveling salesman problem.  |
| GRU     | Gated recurrent unit.        |
| JSSP    | Job shop scheduling problem. |
| e       | Connected autonomous vehicle. |
| LSTM    | Long short-term memory networks. |

I. INTRODUCTION

In the recent past, deep learning has witnessed explosive growth in terms of development of novel architectures, algorithms, and frameworks for addressing a wide range of challenging real-life problems ranging from CV to modeling to control. Among these developments, the use of DNNs for solving sequential decision-making problems within the RL framework, resulting in DRL, is considered one of the state-of-the-art frameworks in artificial intelligence [1]. This approach finds applications in CO [2], games [3], robotics [4], NLP [5], and CV [6]. The tremendous success of DRL in these applications can be credited to: 1) the ability to tackle complex problems in a computationally efficient, scalable and flexible manner, which is otherwise numerically intractable [7]; 2) high computational efficiency allowing fast generation of high-fidelity solutions that are crucial in highly dynamic environments with demand for real-time decisions [8]; and 3) the ability to understand environment dynamics and produce near-optimal actions based solely on interactions with the environment, without the need for explicit prior knowledge of the underlying system [9], [10].

While DRL’s effectiveness has most popularly been demonstrated in games, it is rapidly being adopted in various other real-life applications. Many of these applications involve environments exhibiting explicit structural relationships that can be represented as graphs [11]. For example, a network of cities in the TSP or an incomplete KG are inherently characterized by a graph-based arrangement of the different entities. Methods developed for handling data in the Euclidean space are not well-suited for such environments that require special treatment in terms of encoding the nodes or aggregating the information from different agents. These aspects are systematically modeled with GNNs, detailed in Section II. Incorporation of such structural relationship serves as an auxiliary input and further improves the quality of solutions.

Recently, researchers have been exploring the idea of fusing powerful GNN models with DRL to efficiently tackle such graph-structured applications. There are two primary advantages of this hybrid approach. First, DRL is rapidly being adopted in various applications that involve environments and formulations exhibiting structural relationships [2], [9]. GNNs are very effective in capturing such relationships and thus improves the naive model performance. More importantly, GNNs offer a computationally efficient and powerful framework for large-scale complex DRL environments, which is not feasible with other modeling paradigms. Second, there are several optimization-related tasks in GNN that can be efficiently handled with DRL compared to any other paradigm. For instance, tweaking graphs (by adding nodes and edges) in order to obtain robust models against adversarial attacks. Therefore, a thorough review of these hybrid works could be extremely beneficial in identifying challenges and determining future research directions.

A. Related Work

Several review works either related to DRL or GNN in general are continuously being published [2], [5], [6], [7], [9], [10], [12], [13], [14], [15], [16]. Most of these review articles include only a small paragraph or section that briefly presents the combination of the two approaches. For instance, Wu et al. [17] explained various types of GNN architectures and their specific applications, but it discusses nothing about the GNN other than the fact that it can be used to generate graphs of desirable properties. Similarly, in [18], RL is described briefly in the context of path finding and relation extractor for the KG in order to reason. DRL is primarily presented as a process or algorithmic steps, and the review lacks a comprehensive understanding of the applicability and significance of the particular DRL formulation. In a nutshell, the majority of these surveys are conducted via the lens of a particular application domain. As a result, they are confined to specific approaches that ignore holistic perspectives across domains. To the best of our knowledge, comprehensive reviews dedicated to the study of the combined potential of DRL and GNN do not exist in the current literature.

B. Contributions

This article focuses on a systematic literature review of the fusion of DRL and GNN and makes the following contributions.

1) A rigorous review of articles spanning novel algorithmic developments (Section III-A) and multiple application domains (Section III-B) is conducted for hybrid DRL and GNN paradigm, which is rapidly gaining momentum and proving to be useful in several applications.

2) A categorization of symbiotic and application-specific contributions of the integrated DRL–GNN efforts is developed (Section III). To this end, various attributes are identified for classifying and analyzing existing works (Section IV).

3) The survey takes a holistic approach to review the literature with a special focus on critical aspects of algorithms.

4) Both DRL and GNN are still in the early stages of development, as is the study of their fusion. Therefore, a thorough investigation of the associated challenges is performed and future research directions are identified, which can advance the state-of-the-art. (Section V).
This review is limited to articles indexed in IEEE Xplore, Scopus, and Google Scholar. Initially, the keywords “DRL” and “GNN” are used to select articles from databases. This search led to more than 100 articles from the year 2017 to 2022. The resulting list is filtered to identify articles that include both DRL and GNN, which finally led to approximately 40 articles. Most of the articles in this hybrid paradigm appear in the past few years, which indicates the relatively recent history and relevance of this trending research topic. Among the 40 articles, 22 come from conference proceedings, eight come from journals, and the remaining ten are preprint manuscripts.

This article is organized as follows. In Section II, we offer a brief methodological background of both DRL and GNN to equip readers to understand the fundamentals prior to looking at the fusion of those techniques. Section III presents a comprehensive review of existing literature, including classification based on different novel attributes. In Section IV, we discuss our findings in terms of the applicability and unique offerings of an approach involving GNNs and DRL. Section V highlights key limitations in the existing literature as well as potential future directions for research. Section VI concludes this study.

II. OVERVIEW OF DRL AND GNN

This section begins with the definition of a graph, a mathematical model best suited to describe networked systems, and provides the foundations of two powerful learning paradigms, namely, DRL and GNN. We introduce RL and then present its extension with DNNs, i.e., DRL. Then, we briefly explain the fundamentals of GNN algorithms. Fig. 1 shows these two learning paradigms through an infinite shape loop. The left loop shows the DRL paradigm, and the right loop exhibits the framework of GNN. Through Fig. 1, we first illustrate the contrasting view of these two frameworks in general by leveraging three attributes, namely, “goals,” “key elements,” and “algorithms.” “Goals” primarily represents the objectives or tasks that are typically accomplished with the framework. “key elements” describe the process involved in obtaining the algorithm. “Algorithm” highlights the widely known architectures developed under the framework. We also marked the intersection area of two loops, where the two paradigms come together in a fundamentally holistic way to solve complex problem in an elegant way. This section will equip the readers with the required background knowledge to follow the hybrid works on DRL and GNN (discussed in (Section III).

A graph $G$ is a mathematical model for representing a networked system. It is typically denoted via a tuple $G = (\mathcal{V}, \mathcal{E})$ of a set of $n_V$ nodes $\mathcal{V}$ and $n_E$ links $\mathcal{E}$. The nodes represent the entities of the underlying networked system (e.g., users in an online social network) and the links denote their relationship (e.g., friendship). Each node $u_i, i \in [1, n_V]$, of the graph may consist of features, typically represented through a vector $h_i$. Similarly, the link $e_{ij}$ between $u_i$ and $u_j$ could also be associated with a set of features represented as $h_{ij}$, which can, for example, signify the strength of the link. The interconnections in the graph can also be represented with an adjacency matrix $A \in \mathbb{R}^{n_V \times n_V}$ such that $A(i, j) = 1$ if there exists a link between $u_i$ and $u_j$, and 0 otherwise.

A. Deep Reinforcement Learning

RL is considered the third important branch of machine learning with supervised and unsupervised learning serving as the other two [19]. RL is a sequential decision process where agents are trained to take optimal actions for different scenarios of an environment. The action transitions the environment to a new state, and meanwhile, the agent gets some reward that quantifies how good or bad the action was. To formulate the sequential decision process, RL employs a well-known mathematical concept of MDP [19]. Typically, an MDP is defined by $(X, A, p)$, where $X$ is a finite state space, $A$ is the action space for each state $x \in X$, $p$ is the state
transition probability from state $x^t$ at time $t$ to state $x^{t+1}$ at time $t + 1$, and $r$ is the immediate reward value obtained after an action $a \in A$ is performed. The agent’s primary goal is to interact with its environment (take state as input) at each time step to find the optimal policy $\pi^*$ (return action for the current state) in order to reach the goal while maximizing the cumulative rewards (expected return) over the entire time period. The agent takes the state $X$ as input and returns an action $a$ to be taken. At a particular time step $t$, the expected return $R^t$ is the sum of rewards from the current time step onward until the last time step $T$. When taking an action, an agent must choose between taking the best action based on previous experiences (exploit) and gathering new experiences (exploration) in order to make better decisions in the future. A common approach to account for the tradeoff is the epsilon greedy strategy, where the agent takes a random action with a probability $\epsilon$ [19].

In addition, there are real-life situations where agents lack sufficient knowledge about the environment for holistic learning. Therefore, POMDP is designed for these conditions [19]. A POMDP is an MDP where the agent only possesses a partial view of the state, and therefore, policy functions map the history of observations (belief states) to actions. Its typical expression is similar to that of MDP ($X, a, \Omega, T, p, O, r$, and $b_o$) with some extra elements. The new elements include $\Omega$ that denotes observation, $T$ signifies time, $O$ represents observation probabilities, and $b_o$ is the initial probability distribution of states. Every time the agent takes the action, it receives an observation $o \in \Omega$, which depends on the new state of the environment, the just taken action, and probability $O(o|X, a)$.

Traditional RL records the state, action, and reward values for different actions taken at the time of training in a tabular format. However, this tabular approach is not scalable to a larger number of state–action pairs or to a continuous state–action space [20]. This challenge can be addressed by employing a DNN to approximate the state–action values, leading to the paradigm of DRL. The DNN in a DRL framework estimates the state values and avoids a tedious record-keeping, thus providing an elegant solution to the scalability challenge. The added advantage of using DNN in DRL is that it allows the agent to generalize the value of states it has never seen before or has partial information about, by leveraging the values of similar states. As a result, DRL algorithms are far more generalizable and practical for use across a wide range of applications involving vast state spaces. The other advantage of DNN over conventional linear approximators is that it can model high-level abstraction in trajectories and thus can learn policies for complex environments and interactions directly from the data. Thus, DNN enables the RL agent to have a good perception of its environment [21]. Furthermore, DNN can also be used to approximate the reward function for following a certain policy path [22].

There are several ways to classify existing DRL algorithms, such as model-free versus model-based, value versus policy-based, and offline versus online learning. In the following, we will provide the fundamental concepts of these algorithms across different categories.

1) Value-Based DRL: Value-based methods aim to learn the value of the state or state–action pair and then select actions accordingly. The state–action value function $Q_\pi(x, a)$, expressed in (1), is the expected return starting from state $x$, taking action $a$, and thereafter following a policy $\pi$. DQN is one of the widely used algorithms in this category [23]. Q-learning enables the agent to choose an action $a \in A$ with the highest $Q$ value available from state $x \in X$ based on a DNN model, which maps discrete state–action space with $Q$ values. The parameters of DQN network is updated every time step following the Bellman optimality equation as shown in (1). DQN is an “off-policy” algorithm, where a target policy is used to take action at the current state $X$ and a different behavior policy is used to select action at the next state

$$Q_\pi(x, a) = E_\pi(R^t|X^t) = E_\pi(\sum_{k=0}^{\infty} \gamma^k r^{t+k}|X^t)$$

$$= (1 - \alpha)Q(x, a) + \alpha \left[ r^{t+1} + \gamma \max_{a' \in A} Q(x^{t+1}, a') \right]$$

(1)

in which $r^t$ and $R^t$ are the reward and cumulative discounted reward at time $t$, respectively. $\gamma$ is a discount factor and $\alpha$ is the learning rate, which takes the values between 0 and 1. A key feature of DQN training is the replay buffer, which stores trajectory information $(x^t, a^t, r^t, x^{t+1})$ during each step of the training. In DQN, DNN is trained using a minibatch of a randomly selected sample (experiences) from replay buffer, which offers various advantages in terms of sample efficiency, low variance, and large learning scope. For each sample, the input (state) is passed through the current DNN to generate an output $Q(x, a; \theta)$. The target $Q$ value corresponds to the Bellman optimality equation in (1) and is used to minimize the following loss function [19]:

$$L(\theta) = E \left[ R + \gamma \max_{a' \in A} Q(x^{t+1}, a') - Q(x, a; \theta) \right].$$

(2)

DQN has many variations to improve its current design, including double DQN and dueling DQN. The max operator in the DQN update equation selects and evaluates an action including double DQN and dueling DQN. The max operator overestimates the value function. Double DQN addresses this problem by employing two distinct networks, one for action selection and the other for action evaluation [24]. Similarly, dueling Q network approximates the $Q$ function by decoupling the value function and the advantage function [25].

2) Policy-Based DRL: These methods learn the policy directly unlike value-based methods that learn the values first and then determine the optimal policy. Typically, a parameterized policy $\pi_\theta$ is chosen with parameters constantly updated by minimizing the expected return using a gradient-based approach also known as policy gradient theorem [26]. They are particularly suitable for very large action space (continuous problems) and learning stochastic policies. The policy can be written as

$$\pi(a|x, \theta) = \Pr(a|x, \theta)$$

(3)

which denotes the probability of taking action $a$ being at state $x$, and policy is parameterized by $\theta$. Now, we need an objective
function to assess the performance of this policy. The objective function can be defined as

$$ J(\theta) = v_{\pi_0}(x_0) $$

where $v_{\pi}(x_0)$ is the true value function for the policy $\pi(a|x, \theta)$ and $x_0$ is the start state. In short, the maximizing $J(\theta)$ means maximizing $v_{\pi}(x_0)$, which is equivalent to $\nabla J(\theta) = \nabla v_{\pi}(x_0)$. According to the policy gradient theorem

$$ J(\theta) \propto \sum x \mu(x) \sum a q_{\pi}(x, a) \nabla \pi(a|x, \theta) $$

where $\mu(x)$ is the distribution under $\pi$, $q(x, a)$ is the action value function, and $\nabla v_{\pi}(x_0)$ is the gradient of $\pi$ given $x$ and $\theta$. Thus, the theorem says that $J(\theta)$ is proportional to the sum of the $q$ function times the gradient of the policies for all feasible actions at the states that we might be at. However, to compute this gradient, we need to find $\nabla \pi(a|x, \theta)$. It turns out that the gradient can be expressed as

$$ \nabla \theta \pi_{\theta}(x, a) = \pi(x, a) \nabla \log \pi_{\theta}(x, a) $$

where $\nabla \log \pi_{\theta}(x, a)$ is the scoring function. Since this is a gradient method, the update of the parameters (that we are trying to optimize) will be conducted in the following way:

$$ \Delta \theta = \alpha \nabla \theta J(\theta). $$

There are various ways to define the policy such as softmax activations of a neural network output or parameterized Gaussian distribution [19]. Next, we discuss three widely used policy-based methods.

1) **REINFORCE [27]**: Here, the parameter updates at a given time step involve only the action taken from the current state—the update relies on estimated return by the Monte Carlo method using episode samples. Since it relies on expected return from the current time step, it works only for the episodic tasks [7].

2) **TRPO [28]**: Add Kullback–Leibler (KL) divergence constraints for enabling the trust region in the optimization process. It makes sure that the new updates policy is not far away from the old policy (i.e., constraints the change of the policy network during parameter updates) or we can say that the new policy is within the trust region of the old policy. This constraint is in the policy space rather than in the parameter space. The KL constraint adds additional overhead in the form of hard constraints to the optimization process. Hence, there is a simpler approach to this problem in terms of PPO.

3) **PPO [29]**: It relies on the clipped surrogate objective function to reduce the deviation between the new policy and the old policy. Basically, it defines the probability ratio between the new policy and old policy and led this ratio to stay within a small interval around 1. The clip function truncates the policy ratio between the range $1 - \epsilon$ and $1 + \epsilon$, where $\epsilon$ is a hyperparameter. The objective function of PPO takes the minimum value between the original value and the clipped value. It is relatively simpler in implementation and empirically performs on par with TRPO [7].

Furthermore, these algorithms (REINFORCE, TRPO, and so on) leverage value estimates for parameter updation. There is another line of work that assumes the presence of model information and uses techniques from dynamic programming and approximate dynamic programming to develop model-aided RL techniques such as actor-only policy or direct controller [30].

3) **Actor–Critic DRL**: Both value- and policy-based algorithms have some limitations. While value-based algorithms are not efficient for high-dimensional action space, policy-based algorithms have high variance in gradient estimates. To overcome these shortcomings, an actor–critic method has been proposed, which combines the two approaches [31]. Fundamentally, the agent is trained with two estimators: first, an actor function that controls the agent’s behavior by learning the optimal policy, i.e., provides the best action $a'$ for any input state $X'$, and second, a critic function that evaluates the action by computing the value function.

Some of the popular variants of algorithms under this category are discussed next. A2C consists of two DNNs—one for actor and one for critic [32]. The term “advantage” corresponds to the temporal difference (TD) error as shown in the following:

$$ \text{TD}_\text{error} = \text{TD}_\text{Target} - V(x) $$

where $\text{TD}_\text{target}$ is the predicted value of all future rewards from the current state $x$ and $V(x)$ is the value of the current state evaluated from the critic network. Besides A2C, asynchronous A3C executes different agents in parallel on multiple instances of the environment instead of experience replay as in A2C [32]. Although A3C is memory efficient, its updates are not optimal as different agents work with different versions of model parameters. DDPG is an extension over deterministic policy gradient (DPG), which is designed for continuous action space [33]. DPG defines the policy to be the function $\mu_{\theta} : X \rightarrow A$, where the state space $X$ is the input and output corresponds to action space $A$ following parameters $\theta$. Thus, instead of computing the integral over actions as seen in stochastic policy, it is required to only sum over the state space as action is deterministic. DDPG employs a parameterized actor function with a parameterized critic function that approximates the value function using samples. In this way, DDPG can tackle large variances in policy gradients of actor-only methods.

**B. Graph Neural Network**

Learning with graph-structured data, such as KGs, biological, and social networks, has recently attracted a lot of research attention. There are numerous benefits of representing data as graphs, such as systematic modeling of relationships and simplified representation of complex problems. It is, however, challenging to interpret and evaluate such graph-structured data by employing conventional DNN-based learning methods. The fundamental mathematical procedures such as convolutions are difficult to implement on graphs due to their uneven structure, irregular size of unordered nodes, and dynamic neighborhood composition. GNNs address these shortcomings.
by extending DNN techniques to graph-structured data [34].
GNN architectures can jointly model both structural information
and node attributes. They provide significant performance
improvement for graph-related downstream tasks such as node
classification, link prediction, community detection, and graph
classification [35]. Typically, GNN models can be viewed as
performing information diffusion over the layers. Specifically,
they consist of a message-passing scheme that propagates
diffuses) the feature information of the nodes to its neighbors
until a stable equilibrium is reached [17]. This process can
mathematically be expressed as
\[
h^{(l)}_u = f(h^{(l-1)}_u), \quad v \in N(u) \cup u
\]  
where \( h \) is the node representing (embedding) vector and \( N(\cdot) \)
denotes the neighborhood. Several GNN algorithms have been
proposed to improve this message-passing technique [36],
[37], [38], [39], [40], [41], [42], [43]. We discuss some of
the key approaches as follows.

1) GCN: It is the first effort that incorporates convolution
operations on graphs similar to convolutional operation of
convolutional neural network (CNN) on images [36]. The
core idea behind any GNN is to generate a unique Euclidean
representation of nodes/links in the graph. Conventionally,
spectral methods generate node representation vectors using
eigendecomposition but are computationally inefficient and
are not generalizable [36]. GCN overcomes these challenges
with its powerful approximation, where the update equation
of the node representation vector \( h_u \) at a particular layer \( l \) is
given by
\[
h^{(l)}_u = f\left(\theta^{(l-1)} h^{(l-1)}_u A^\star\right) = g\left[\theta^{(l)} h^{(l-1)}_u D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right]
\]  
where \( A \) is the adjacency matrix, \( D \) is the degree matrix, \( \theta \)
is the learnable parameter, and \( g \) is the activation function. \( A^\star = D^{-\frac{1}{2}} A D^{\frac{1}{2}} \) is normalized in this way to scale the node
features and ensures numerical stability at the same time. It is
important to note that GCN relies on the entire graph (i.e., full
adjacency matrix) for learning node representation, which can
be inefficient as the number of neighbors of a node can vary
from one to thousands or even more and cannot be generalized
to graphs of different sizes.

2) GraphSAGE: It is an inductive node embedding
approach that exploits node attributes to learn an embedding
function [37]. It supports simultaneous learning of topological
structure as well as distribution of node features within a
confined neighborhood. The fundamental premise is to train a
neural network that can recognize structural properties of node
neighborhood, thereby indicating its local role in the graph
along with global position. Initially, the algorithm samples
node features in the local neighborhood of each node in the
graph-structured data. This is followed by learning appropriate
functional mappings to aggregate the information received
by each node as it propagates through the GNN layers.
This inductive learning approach is scalable across graphs
of different sizes as well as subgraphs within a given graph.
The operation performed at the \( l \)th node embedding layer is
given by
\[
h^{(l)}_u = f^{(l)}\left(h^{(l-1)}_u, h^{(l-1)}_{N(u)}\right) = g\left[\theta_C h^{(l-1)}_u + \theta_A \tilde{A}\left(h^{(l-1)}_{N(u)}\right)\right]
\]  
where \( \tilde{A} \) represents the aggregation operation. \( g[\cdot] \) specifies
the activation function, \( h^{(l)}_u \) denotes the node embedding of
node \( u \) at the \( l \)th layer, \( N(u) \) describes the neighborhood of
node \( u \), and \( \theta_C \) and \( \theta_A \) are the parameters of the combination
and aggregation operation of GNN, respectively.

3) GAT: It assumes that interactions of neighboring nodes
to the target node are neither fixed like GCN nor identical
like GraphSage. GAT adopts attention mechanisms to learn
the relative weights between two connected nodes [44]. The
graph convolutional operation according to single-head GAT
is defined as follows:
\[
h^{(l)}_u = \left[\sum_{v \in N(u) \cup u} \alpha_{uv} \theta^{(l)} h^{(l-1)}_v\right]_{a_u}
\]
\[
a^{(l)}_{uv} = \text{softmax}\left(\theta^T h^{(l-1)}_u |\theta^T h^{(l-1)}_v\right)
\]  
where the attention weight \( \alpha_{uv} \) quantifies the connection
strength between node \( u \) and its neighbor \( v \). The attention
weight is learned across all node pairs using the softmax
function that ensures weights sum up to one over all neighbors
of the node \( u \). \( a \) represents the weights of a single-layer
feedforward network associated with attention mechanism.
For the case of multihead attention, the update equation for
the node embedding is expressed as
\[
h^{(l)}_u = \left\|_{k=1} \left[\sum_{v \in N(u) \cup u} \alpha^{(k)}_{uv} \theta^{(l), k} h^{(l-1)}_v\right]\right\|
\]  
where \( \| \) represents concatenation, \( \alpha^{(k)}_{uv} \) are normalized atten-
tion coefficients computed by the \( k \)th attention mechanism,
and \( \theta^{(l), k} \) is the corresponding linear transformation’s weight
matrix. This mechanism selectively aggregates the neighbor-
hood contributions and suppresses minor structural details.

III. CATEGORIZATION OF DRL + GNN METHODS

DRL and GNN have emerged as extremely powerful tools
in modern deep learning. While DRL exploits the expressive
power of DNNs to solve sequential decision-making problems
with RL, GNNs are novel architectures that are particularly
suited to handle graph-structured data. We identify two broad
categories of research articles that jointly make use of GNN
and DRL, as shown in Fig. 2. The first category of articles
makes algorithmic contributions, where DRL and GNN com-
plement each other with an objective of addressing each other’s
shortcomings. On the other hand, the second category of
articles makes application-specific contributions that leverage
a combined GNN-DRL formulation to address problems specific
to different applications. The first category involves novel fun-
damental formulations, which enhances the performance of the
algorithms in a generic way, irrespective of the application/use
cases, whereas papers in the second category combined DRL
and GNN to achieve a certain objective. Therefore, these works entail application-specific formulations that are suitable for particular use cases. The summary of surveyed DRL and GNN fused works is shown in Table I and individual components of surveyed papers are outlined in Table II. "Dynamic," "scalable," "generalizability," and "multiagent" are four important

and GNN to achieve a certain objective. Therefore, these works entail application-specific formulations that are suitable for particular use cases. The summary of surveyed DRL and GNN fused works is shown in Table I and individual components of surveyed papers are outlined in Table II. "Dynamic," "scalable," "generalizability," and "multiagent" are four important

| Reference       | Category | Dynamic | Scalable | Generalizable across envs | Multiagent | Source code | Publication venue-year |
|------------------|----------|---------|----------|---------------------------|------------|-------------|------------------------|
| AGNN [45]        | AD       |         |          |                           |            |             | archiv-2019             |
| GraphNAS [46]   | AD       |         |          |                           |            |             | DCAL-2020              |
| RG-Explorer [47]| AD       |         |          |                           |            |             | NeurIPS-2021           |
| RL-S2V [52]     | AD       |         |          |                           |            |             | ICML-2018              |
| Sun et al. [53] | AD       |         |          |                           |            |             | WWW-2020               |
| G2Net [63]      | AG       |         |          |                           |            |             | AAAI-2020              |
| DCG [64]        | AG       |         |          |                           |            |             | ICML-2020              |
| GraphComm [65]  | AG       |         |          |                           |            |             | ICASLP-2021            |
| SRL-AC [66]     | AG       |         |          |                           |            |             | Elsevier-2021          |
| CSGA [67]       | AG       |         |          |                           |            |             | IEEE SMC-2021          |
| NerveNet [69]   | AG       |         |          |                           |            |             | ICLR-2018              |
| SMP [70]        | AG       |         |          |                           |            |             | ICML-2020              |
| SynNet [74]     | AG       |         |          |                           |            |             | ICML-2020              |
| SR-DRL [75]     | AG       |         |          |                           |            |             | archiv-2021            |
| GCOMM [77]      | PC       |         |          |                           |            |             | NeurIPS-2020           |
| Khalili et al. [78]| PC     |         |          |                           |            |             | NeurIPS-2017           |
| GraMeR [81]     | PC       |         |          |                           |            |             | archiv-2022            |
| RLGN [83]       | PC       |         |          |                           |            |             | ICML-2021              |
| GCN-RL [84]     | PC       |         |          |                           |            |             | DAC-2020               |
| Rnet-DQN [87]   | PC       |         |          |                           |            |             | PRSA-2021              |
| Dori et al. [94]| PT       |         |          |                           |            |             | ICMLA-2020             |
| L2I [96]        | PT       |         |          |                           |            |             | ICLR-2020              |
| Ha et al. [97]  | PT       |         |          |                           |            |             | Elsevier-2020          |
| IG-RL [100]     | PT       |         |          |                           |            |             | IEEE-2021              |
| Shang et al. [103]| PT     |         |          |                           |            |             | IEEE-2022              |
| Lima et al. [116]| FR      |         |          |                           |            |             | IFAC-2020              |
| Wang et al. [121]| PR      |         |          |                           |            |             | IFAC-2020              |
| GNN-MARL [108]  | PR       |         |          |                           |            |             | CRIP-2021              |
| Park et al. [109]| PR      |         |          |                           |            |             | TAP-2021               |
| GCQ [114]       | PR       |         |          |                           |            |             | Wiley-2021             |
| GRL [127]       | PK       |         |          |                           |            |             | Elsevier-2020          |
| 3D-MoKNNR [128] | PL       |         |          |                           |            |             | ICLR-2022              |
| GTPN [130]      | PL       |         |          |                           |            |             | KDD-2019               |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
attributes that are selected to characterize the existing works in Table I. Dynamic and scalable are evaluated for GNN, where dynamic determines whether the framework is applicable for time-varying system (configuration and parameter), and scalable indicates the ease (memory and computational) of building models for large-scale inputs and systems. On the other hand, generalizability and multiagent are defined for DRL algorithms. Generalizability shows the applicability of trained model across similar RL environments. Multiagent determines whether the DRL algorithm involves single agent or multiple agents to accomplish the desired task.

A. Symbiotic DRL–GNN Frameworks

In this section, we discuss the articles that focus on developing novel formulations or algorithms to improve DRL or GNN. In these articles, either GNN is used to improve the formulation and performance of DRL or DRL is used to improve the applicability of GNN.

1) DRL Enhancing GNN: The articles that make use of DRL for improving GNN are used for diverse purposes, including NAS, improving the explainability of GNN predictions and designing adversarial examples for GNN.

a) Neural architecture search: It refers to the process of automatically searching for an optimal architecture of a neural network (e.g., the number of layers and the number of nodes in layer) to solve a particular task. A DRL-based controller that makes use of exploration guided with conservative exploitation is used in [45] to perform an efficient search of different GNN architectures. The search space is made up of hidden dimension, attention head, attention, aggregation, combination,
and activation functions. The authors introduce homogeneity of models as a method to perform a guided parameter sharing between offspring and ancestor architectures. Along with computational gain, the unique characteristics of DRL-based search engine is in terms of generalizability, i.e., the same model can be used to quickly determine the optimal parameters for various use cases (high generalizability). The superiority of the proposed method is demonstrated with better performance on benchmark datasets and baselines [46].

b) Explaining GNN predictions: Generating explanations for DNN predictions is an important task in improving the transparency of ML models [47], [48]. Shan et al. [47] used DRL to improve the existing methods of explaining GNN predictions. The problem of generating explanations for GNN predictions involves identifying the subgraph that is most influential in generating a prediction. The authors devise a DRL-based iterative graph generator that starts with a seed node (the most important node for a prediction) and adds edges to generate the explanatory subgraphs. The reward of the DRL model is obtained as the mutual information between the original predicted label and the label made by the generated graph. The distribution of predictions is based solely on the explanatory subgraph to learn a subgraph generation policy with policy gradient. The authors show that the proposed method achieves better explainability in terms of qualitative and quantitative similarity between the generated subgraphs and the ground-truth explanations. The unique exploitation versus exploration search mechanism of DRL (epsilon greedy) inherently allows the model to find relevant subgraphs without getting stuck in local regions and perform better compared to conventional approaches.

c) Generating adversarial attacks for GNN: Recent studies [49], [50], [51] have shown that GNNs are vulnerable to adversarial attacks that perturb or poison the data used for training them. DRL has been used to learn strategies to make adversarial attacks on GNNs, which in turn can be used to devise defense strategies to such attacks. RLS2V [52] is one of the first frameworks that uses DRL to perform an attack aimed at evading detection during classification. Poisoning refers to the malignation of graph-structured data. Specifically, it employs a Q-learning and structure-to-vector-based attack methodology that learns to modify the graph structure (adding or dropping existing edges) with only the prediction feedback (reduction in accuracy) of the target classifier. While Sun et al. [53] considered a novel NIPA on graph data considering nodes as well as links. Specifically, it injects fake nodes (e.g., fake accounts in social networks) into the graph and uses carefully crafted labels for the fake nodes together with links between them and other (fake as well as genuine) nodes in the graph to poison the graph data. NIPA frames the sequential addition of adversarial connections and the design of adversarial labels for the injected fake nodes as an MDP and solves it with deep Q-learning. To effectively cope with the large search space, NIPA adopts hierarchical Q-learning and GCN-based encoding of the states into their low-dimensional latent representations to handle the nonlinearity of the mapping between states and actions. Furthermore, the underlying Q network in these approaches learns the sensitivity (relationship) between the changes in the input graph and the final performance, rather than exact mapping. This enables them to predict performance for various possible configurations of attacks for different families of graph, i.e., high generalizability, which is not feasible with conventional graph optimization approaches.

2) GNN Enhancing DRL: This section discusses the papers related to the algorithmic improvement of DRL. Specifically, we focus on efforts wherein GNN has been used for RDRL problems for effective modeling of the relationship among: 1) different agents in an MADRL framework and 2) different tasks in an MTDRL framework.

a) Modeling relationship among agents in MADRL: In MADRL, a group of agents cooperate or compete with each other to achieve a common goal. This framework has recently been used for a number of challenging tasks, including traffic light control, autonomous driving, and network packet delivery [54], [55], [56]. In such scenarios, the communication among agents offers additional information about the environment and state of other agents. Several methods have been proposed to learn this communication. The first body of work in capturing these relationships is related to attention-based approaches [57], [58], [59], [60]. ATOC [61], DGN [62], and COMA-GAT [58] provide communication through the attention mechanism. Along these lines, G2ANet [63] employs hard attention to filter out irrelevant data and soft attention to focus on relevant information. DCG [64] employs coordination graphs, which uses the message-passing mechanism to coordinate the behaviors of agents. For each agent, these attention-based algorithms learn the distribution of contributions of other agents’ actions in deciding its own action. The authors in GraphComm [65] explore both static and dynamic relations simultaneously among agents. Specifically, it leverages the relational graph module to incorporate static relationships via relational graph provided by prior system knowledge and uses proximity relational graph for dynamic relationships. Agents’ Q values are learned in a CTDE manner via multiple-layer perception (MLP) and GRU network along with RGCN and GAT to exchange messages among the agents for static and dynamic relations, respectively.

Similarly, Zhang et al. [66] proposed a structural relational inference actor–critic (SRI-AC) framework for CTDE that can automatically infer the pairwise interaction between agents and learn a state representation. The model is used to predict which agents need to interact in advance and then supply the most relevant agent observation information to the critic network. In particular, each agent has a critic, which leverages information from the combined action as well as appropriate observational data during training. Then, a variational autoencoder (VAE) is used to infer the pairwise interaction and state representation from observed data, followed by a critic network that employs GAT to integrate the knowledge from neighboring agents. In a similar vein, Yun et al. [67] presented a novel state categorization method for CTDE DRL. Basically, it separates the state into agents’, own observations, allies’ partial information, and opponents’ information specific to the Starcraft game and then leverages GAT to learn the correlation and relationship among the agents.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
b) Modeling relationship among tasks in MTDRL: This framework provides an elegant way to exploit commonalities between multiple tasks in order to learn policies with improved returns, generalization, data efficiency, and robustness. One of the inherent assumptions in a majority of MTDRL works is compatible state–action spaces, i.e., the same dimensions of states and actions across multiple tasks. However, this is violated in many practical applications such as CO and robotics [68]. This issue has been addressed by using GNNs that are capable of processing graphs of arbitrary size, thereby supporting MTDRL in incompatible state–action environments [68]. Since GNNs provide the flexibility to incorporate structural information, it enables the integration of additional domain knowledge, where states are characterized as labeled graphs. The use of GNN in MTDRL has been demonstrated in continuous control environments that exploit the physical morphology of the RL agents for constructing input graphs [69], [70]. Here, limb features are encoded in the form of node labels and edges represent the physical connections between the corresponding limbs. In this way, the structure of the agents is explicitly modeled in the form of graphs. NerveNet [69] serves as the policy network, which first propagates information over agent structure, followed by predicting actions for different parts of the agent. Huang et al. [70] formulated a single global policy that can be represented as a collection of modular neural networks called shared modular policies (SMPs), each of which is designated to handle tasks related to its corresponding actuator.

c) Relational symbolic input for RDRL: The fundamental premise of RDRL is to integrate DRL with relational learning or inductive logic programming [71], in which states, actions, and policies are represented by a first-order/relational language [72]. The tasks in this space are characterized by variable dimensional state and action spaces. In these problems, it is difficult to find a fixed-length representation that is required by a majority of the existing DRL methodologies. This issue can be handled using GNN by formulating relational problems in terms of graph-structured data. The mechanics of a relational domain are typically represented by RDDL [73]. Garg et al. [74] proposed SymNet for automated extraction of objects, interactions, and action templates from RDDL. Specifically, an RDDL problem instance is transformed into an instance graph, for which the embeddings of states and important object tuples are generated using GNN. Furthermore, these are decoded into scores corresponding to individual ground actions. Node embeddings are generated using GNN and action templates are applied over object tuples to create a probability distribution. This is followed by updating the model using a policy gradient method. However, SymNet is computationally expensive and is applicable only when the RDDL domain definition is available since predefined transition dynamics are required to construct the graphs. Symbolic relational DRL (SR-SRL) [75] addresses these limitations by considering an enriched symbolic input comprising objects and relations along with their features in the form of a graph. This does not require information about transition dynamics and generalizes over any number of object tuples. The unique aspect of GNN in terms of representing nodes/links in high-dimensional embedding space makes them a powerful candidate for quantifying relationship among the large space of agents and tasks as described in Sections III-A2.a–III-A2.c. As a result, they are very successful in assimilating the relational information compared to any other relational learning methodologies. In a nutshell, the review of papers in part A (i.e., symbiotic developments) indicates that the first set of methods primarily addresses challenges associated with GNN models. Specifically, these challenges (such as explainability of GNN predictions and computational complexity of training and inference process) typically involve large optimization problems, which can be effectively handled with DRL. The unique “learning from interaction” approach makes DRLE a suitable candidate for this kind of sequential identification problem. Furthermore, it exhibits all the desirable advantages of a learning paradigm, i.e., low computational effort, large scalability, and high generalizability compared to the conventional graph optimization methodologies. On the other hand, the second class of works focuses on improving the encoding aspects of DRL algorithms. Specifically, the complexity of the decision problems is continuously increasing where DRL is adapting to multiagent and multitask configurations. This demands a powerful encoding mechanism to represent the underlying systems and scenarios, which can be efficiently accomplished with GNN. The sophisticated message-passing mechanism combined with attention-based learning process and representation of nodes/links in high-dimensional embedding space makes them a powerful candidate for quantifying relationships among the large space of states, distributed multiagents, and multitasks in complex DRL environments. Furthermore, since GNN is a learning-based module, their predictions can be quickly obtained and utilized while training the DRL models.

B. Application-Specific Implementations

The second broad category of articles exploits the versatility of DRL along with the flexible encoding capability of GNNs to address interesting challenges in different application domains. These domains span a wide spectrum, including CO, transportation, control, KGs, and life sciences, which we briefly review next.

1) Combinatorial Optimization: Many CO problems are computationally expensive and require approximations and heuristics to solve in polynomial time [76]. There has been an increasing interest in solving CO problems using DRL since they naturally fit into the problem. A typical CO problem can be expressed as

$$\min_{a, w} f_{obj}(a, w)$$
$$\text{s.t. } f_i(a, w) \leq 0$$

where $a \in \mathbb{R}^n$ and $w \in \mathcal{C}$ represent the continuous and discrete variables, respectively, with $\mathcal{C}$ representing the feasible set for $w$. Here, $f_{obj}$ is the objective function to be minimized and $f_i(\cdot)$ represents the constraints. In this regard, CO problems are often framed as an MDP where the optimal actions/solutions
can be learned with DRL. Furthermore, the underlying environment is represented as a graph that is processed using GNN. The problem can be formally written as follows. Given a graph $G = (V, E)$ and a budget $b$, find a set $S^* \subseteq V$ of $b$ nodes such that $f(S^*)$ is maximized, where $f(S)$ is the cost function specific to the objective. In DRL, this problem is typically formulated as an MDP, where the state represents the current solution set $S$ and the action refers to the process of selecting a node from $V$ and appending to the solution set. The process is repeated until the cardinality $|S|$ reaches $b$. Reward quantifies the benefit of taking an action, and it is proportional to $f(S)$. The articles addressing these challenges can further be divided into the following subcategories.

a) Solving CO problems in operations research: Manchanda et al. [77] used GNN to capture the structural information of CO problems and thus addressed the poor generalizability and scalability of existing DRL-based approaches. They learned a construction heuristic for a budget-constrained MVC problem by combining supervised learning and DRL. GCN is first utilized to find appropriate candidate nodes by learning the scoring function computed using the probabilistic greedy approach. Then, the candidate nodes are used in an algorithm similar to [78] to sequentially construct a solution. Since the degree of nodes in large graphs can be rather high, importance sampling based on computed score is used to select the neighboring nodes while determining embeddings, thereby reducing the computational complexity. Extensive experiments on random/real-world graphs reveal that the proposed method marginally outperforms S2V-DQN and scales to much larger graph instances up to 100,000 nodes. In addition, it is significantly more efficient in terms of low computation effort due to a relatively smaller number of learned parameters.

Another interesting application of DRL and GNN can be seen in solving diffusion processes in graphs, such as influence maximization and epidemic test prioritization, among others [79], [80]. The goal is to identify a set of nodes on a temporally evolving graph such that the global objective of curbing spread or maximizing information spread is achieved [79], [81], [82]. Various graph-theoretic algorithms have been developed to address this class of problems. However, they are inefficient when scaling to larger graphs. Furthermore, the added difficulty is that the states are partially observed, for instance, we might not know the ground-truth infection status for every node in the graph at any point in time. To address these challenges, Meirou et al. [83] posed the problem of controlling a diffusive process on a temporally evolving graph as a POMDP. The problem of selecting a subset of nodes for dynamical intervention is formulated as a ranking problem, and an actor–critic PPO is employed to solve it. Specifically, the architecture contains two separate GCN modules; one updates the node representation according to the dynamic process and the other is in charge of long-range information propagation. The results on various real-world networks, including COVID-19 contact tracing data, show the superior performance of this approach along with high scalability to large-size networks.

b) Solving design problems: Several design problems, especially electronic circuit design, are CO problems, which can benefit from a DRL-GNN formulation. For instance, automatic transistor sizing is a challenging problem in circuit design due to the large design space, complex performance tradeoffs, and fast technological advancements [84], [85]. Wang et al. [84] presented the GCN-RL circuit designer, which uses DRL to transfer knowledge between different technology nodes and topologies. GCN is employed to learn the circuit topology representation. The GCN-RL agent retrieves the features of the topology graph with transistors as nodes and wires as links. The actor–critic approach with continuous space algorithm DDPG is used. The generalizability of DRL enables training on one technology node and then applying the trained agent to search the same circuit under different technology nodes. GCN extracts the circuit features that enable the transfer of knowledge between different topologies sharing similar design principles, for example, between two- and three-stage transimpedance amplifiers.

A similar problem is logic synthesis for combinational circuits, in which the lowest equivalent representation for Boolean logic functions is sought. A widely used logic synthesis paradigm represents the Boolean logic with standardized logic networks, such as and-inverter graphs (AIGs), which iteratively conducts logic minimization operations on the graph. To this end, Zhu et al. [86] posed this problem as an MDP and DRL is incorporated with GCN to explore the solution space. Specifically, this work leverages the Monte Carlo policy gradient-based RL algorithm, REINFORCE. Since circuits and AIG logic can be naturally modeled as graphs, they leverage GCN to extract the current state’s features.

c) System robustness: Recently, Darvariu et al. [87] demonstrated a novel application of DRL with GNN, where they used DRL to search for optimal graph topology for a given graph objective. Essentially, the construction of a graph is framed as a sequential decision process of adding a fixed number of links to the current graph one at a time such that the robustness score of the final graph is the maximum among all feasible combinations of the given graph and edges. In particular, the state represents the current graph, while the action corresponds to a new node that needs to be added. They are encoded using the S2V variant of GNN [88], and DQN constitutes the underlying DRL engine. Although this approach is more computationally efficient than conventional approaches (such as greedy and Fiedler vector [89]), it requires an iterative algorithm at each step of the episode to compute the global score of the current graph (robustness in this case), which demands some computational effort. This can be avoided by using learning-based models to compute intermediate rewards, i.e., global graph scores [90], [91].

2) Transportation: Transportation problems that are handled with DRL and GNN can be broadly classified into two classes, namely, routing and speed prediction.

a) Vehicle routing: Routing problem is usually formulated by considering a graph $G(V, E)$, in which $V = C \cup [0, n + 1]$ is the set of nodes associated with customers in $C$ and to the depot nodes 0 and $n + 1$. Nodes 0 and $n + 1$ represent the same depot and impose that all routes must start on 0 and return to $n + 1$. Set $E$ contains the links $(u, v)$ for each pair of nodes $u, j \in N$. The cost of crossing a link $(u, v) \in E$ is...
denoted by $c_{uv}$. Each node has a demand $q_i$. The objective of the problem is to determine a set of minimal cost route for each of the $k$ vehicle satisfying all the requirements. In DRL, this problem is typically formulated by initializing a solution set $S$ with the fixed starting and ending node representing depot. The action would be to select the city node and fill the intermediate values in the solution set $S$. The reward will be proportional to both the cost of the links ($c_{uv}$) and demand of the selected node $a$. The process is repeated until the aggregated demand of all the selected cities crosses the net limit. This problem can also be posed in a multiagent setting where each agent (here each vehicle) allot its routes by exchanging the information among them. TSP, a subclass of VRPs, is one of the first problems to be solved with DRL and GNN. Here, the objective is to find the shortest possible route that visits each node in the graph exactly once and returns to the source node [92]. Here, the state is denoted by a graph embedding vector that describes the tour of the nodes until time step $t$, whereas the action is defined as selecting a node from the nonvisited pool and the reward is the negative tour length. GNN with attention mechanism is used as an encoder followed by a pointer network decoder [93]. The described encoder–decoder network’s parameters are updated using the REINFORCE algorithm with a critic baseline. The methodology proposed in [94] adopts a GNN representation to offer a general framework for model-free RL that adapts to different problem classes by altering the reward. This framework uses the edge-to-vertex line graph to model problems and then formulates them in a single-player game framework. The MDPs for TSP and VRP are the same as in [95]. Rather than employing a full-featured Neural MCTS, Drori et al. [94] represented a policy as a graph isomorphism network (GIN) encoder with an attention-based decoder, which is learned throughout the tree search operation. Furthermore, Lu et al. [96] proposed to learn the improvement heuristics (methods that start from an arbitrary policy and improve iteratively) for VRP in a hierarchical manner. They devised an intrinsic MDP that includes not only the present solution’s features but also the running history. The REINFORCE method is used to train the policy, which is parameterized by a GAT.

Another important problem of cooperative CO in TSP is related to the optimization of the multiple TSPs (MTSPs). Hu et al. [97] developed an architecture consisting of a shared GNN and distributed policy networks to learn a common policy representation to produce near-optimal solutions for the MTSP. Specifically, Hu et al. [97] used a two-stage approach, where REINFORCE is used to learn an allocation of agents to vertices and a regular optimization method is used to solve the single-agent TSP associated with each agent.

b) Speed/flow control: The second class of transportation problems deals with the prediction of speed/flow in the road network commonly referred to as traffic signal control (TSC) [98]. In recent years, TSC has been modeled as an MDP and researchers have adopted DRL to control the traffic signals [99], [100], [101], [102]. Yang et al. [101] proposed an inductive heterogeneous graph multiagent actor–critic (IHG-MA) algorithm consisting of three steps: 1) sampling of heterogeneous nodes via fast random walk with restart approach; 2) encoding heterogeneous features of nodes in each group using Bi-GRUs; and 3) aggregating embeddings of groups using a graph attention mechanism. Finally, the proposed MA framework employs the actor–critic approach on the obtained node embeddings to compute the $Q$-value and policy for each SDRL agent and optimizes the whole algorithm to learn the transferable traffic signal policies across different networks and traffic conditions. Shang et al. [103] proposed to use a DQN agent to effectively combine the predictions of GCN and GAT, thus improving the overall space-time modeling capabilities and forecasting performance. The DQN provides weights to combine the GCN and GAT predictions, where the weights are adaptive to different network topologies, weather conditions, and other relevant attributes of the traffic data. Apart from the road network, traffic is also monitored in communication domains such as optical network. Almasan et al. [104] proposed to use of message-passing GNN architecture to solve routing optimization in optical networks. The key contribution of this work lies in its generalizability as it is able to achieve outstanding performance over several real-world networks that are never seen during training.

3) Manufacturing and Control: DRL has also been explored in modern manufacturing systems because of the increasing complexity and interdependency across processes and system levels [105], [106], [107]. Recently, Huang et al. [108] proposed an integrated process system model based on GNN. Here, the manufacturing system is represented as a graph, where machines are treated as nodes and material flow between machines is treated as links. GCN is used to encode machine nodes and obtain a node’s latent representation that reflects both the local condition of the machine (i.e., parameters of neighboring machines) and the global status of the entire system. Each machine is modeled as a distributed agent, and MARL is trained to learn an independent adaptive control policy conditioned on the node’s latent feature vector. The latent characteristics of the node, machine process parameters, and total yield with defects serve as the state, action, and reward of the underlying MDP, respectively. Specifically, C-COMA [58] has been deployed by employing the A2C framework in a distributed setting and is easily compatible with GNN.

In manufacturing, JSSP is also an important problem that aims to determine the optimal sequential assignments of machines to multiple jobs consisting of series of operations while preserving the problem constraints. Park et al. [109] proposed a framework to construct the scheduling policy for JSSPs using GNN and DRL. They formulate the scheduling of a JSSP as a semidefinite programming problem (SDP) in a computationally efficient way by representing the state of a JSSP as a disjunctive graph [110], where nodes represent operations, conjunctive edges represent precedence/succeeding constraints between two nodes, and disjunctive edges represent machine-sharing constraints between two operations. Then, they employ a GNN to learn node embeddings that summarize the spatial structure of the JSSP and derive a scheduling policy that maps the embedded node features to scheduling action. PPO algorithm, a variant of policy-based RL, is used
to train the GNN-based state representation module and the parameterized decision-making policy jointly [29].

Another key application of DRL is in the control of CAVs [111], [112], [113]. However, DRL-based controllers in most existing literature address only a single or fixed number of agents with both fixed-size observation and action spaces. This is because of the highly combinatorial and volatile nature of CAV networks with dynamically changing the number of agents (vehicles) and the fast-growing joint action space associated with multiagent driving tasks, which pose difficulty in achieving cooperative control. Recently, Chen et al. [114] presented a DRL-based algorithm that combines GCN with DQN to achieve efficient information fusion from multiple resources. A centralized multiagent controller is then built upon the fused information to make collaborative lane changing decisions for a dynamic number of CAVs within the CAV network.

The efficient allocation of communication resources in wireless networks that are commonly used in modern control systems to exchange data across a vast number of plants, sensors, and actuators is also addressed with DRL [115], [116]. However, these techniques do not scale well with the network size. To overcome this issue, Lima et al. [117] employed a GNN to parameterize the resource allocation function. In particular, Gama et al. [118] used random-edge GNNs since the underlying communication graph is randomly distributed and then coupled it with REINFORCE for a continuous action space.

Another interesting application can be found in multiagent formation control. Although a number of algorithms can achieve formation control effectively, they ignore the structure feature of the graph formed by agents [57], [119], [120]. Wang et al. [121] proposed a model named MAFCOA building on the framework of GAT. In particular, the model can be divided into two parts, including formation control and obstacle avoidance. The first part uses GAT and focuses on cooperation among agents, while the second part focuses on obstacle avoidance with multi-LSTM models. The multi-LSTM allows the agents to take obstacles into consideration in the order of distance and avoid an arbitrary number of obstacles [122]. Moreover, in order to scale to more agents, the parameters are shared to train all the agents in a decentralized framework. The actor and critic approach is used with MADDPG to learn the optimal control policy for multiple agents.

4) Knowledge Graph Completion: KGs are increasingly being used to represent heterogeneous graph-structured data in a wide variety of applications, including recommendation systems, social networks, information extraction, and semantic parsing. One of the key problems in real-world knowledge bases is that they are notoriously incomplete, i.e., a lot of relationships are missing. KGC is a knowledge base completion process that aims to fill-in the incomplete real-world knowledge bases by inferring missing entries with the help of existing ones. The entities and corresponding relations are represented by means of triplets consisting of head nodes (h), relations (r), and tail nodes (t). The problem of KGC entails the prediction of missing tails for given pairs of head nodes and relations. Traditional RL-based methods do not consider the generation of new subgraphs within existing KGs (e.g., new or missing target entities) and are also not effective for reasoning tasks. In addition, the issue of reward sparsity leads to a large variance in sampling methods and low learning efficiency. Therefore, various DRL-based methods have been proposed to overcome the limitations [123], [124], [125], [126]. Wang et al. [127] proposed a GAN-based DRL (GRL) framework. They divided the problem into two scenarios: when the target entity can be located within limited time steps and when the target entity cannot be found from the original KG, while there are still time steps to go, in which case a new subgraph is formed. KGC being defined as an MDP explores the rules that can be introduced in both the state transition process and rewards to better guide the walking path under the optimization of GAN. LSTM is employed as a generator of GAN, which not only records previous trajectories (of states, actions, and so on) but also generates new subgraphs and trains policy networks with GAN. Furthermore, to better generate new subgraphs, a GCN is used to embed the KG into low-dimensional vectors and parameterize the message-passing process at each layer. In addition, GRL also applies domain-specific rules and utilizes DDPG to optimize rewards and adversarial loss.

5) Life Sciences: Along with engineering applications, recent advancements in ML have also demonstrated the potential to revolutionize various life sciences applications such as drug discovery [128], [129], [130] and brain network analysis [131]. To this end, McNaughton et al. [128] proposed a new method for designing antiviral candidates coupling DRL to a deep generative model. Specifically, they used the actor–critic approach, in which a scaffold-based generative model is leveraged as the actor model to build valid 3-D compounds. For the critic model, parallel GNNs are used as a binding probability predictor to determine whether the generated molecule actively binds with a target protein [129]. The results demonstrated that the model could produce molecules with higher drug likeness, synthetic accessibility, water solubility, and hyrdrophilicity than current baselines. Do et al. [130] proposed a graph transformation policy network (GTPN) that combines the strengths of DRL and GNN to learn reactions directly from data with minimal chemical knowledge. Their model has three key components: a GNN, a node pair prediction network, and a policy network. The GNN is responsible for obtaining the atom’s representation, the node pair prediction network is responsible for computing the most possible reaction atom pairs, and the policy network is responsible for determining the optimal sequence of bond changes that transforms the reactants into products. In addition, the model’s step-by-step creation of product molecules allows it to exhibit intermediate molecules, greatly improving its interpretability. Zhao et al. [131] proposed a novel brain network representation framework (i.e., BN-GNN) that utilizes DRL for automatic prediction of optimal number of feature propagations (indicated by the count of layers in GNN) desired for a given brain network. Concretely, a customized aggregation for different networks is achieved to improve the performance of conventional GNNs in brain network representation learning. Experimental analyses have demonstrated
that BN-GNN outperforms state-of-the-art baselines on eight brain network disease analysis tasks.

IV. DISCUSSION AND LESSONS LEARNED

Supported by an extensive review, we observe that the use of GNNs in a DRL framework is becoming increasingly popular from an algorithmic development perspective and applications of machine learning to complex problems. In this section, we present our perspectives in terms of applicability and advantages of fusing these learning frameworks.

A. Advantages of Fusing DRL and GNN

As discussed before, GNN and DRL are fused on two different fronts, i.e., symbiotic enhancement where methodologies are enhancing each other and applications where algorithms are supporting each other. This fusion has several advantages that can be summarized as follows.

1) On moving from single agent to multiagent or from single task to multitask scenarios in DRL, the complexity of problem drastically increases. Therefore, various new approaches are continuously being proposed to improve the model performance. However, there is always a scope to incorporate auxiliary information for further improvement. Since MADRL/MTDRL involves multiple agents, incorporating the relational information among these agents in the core model with a GNN architecture can improve its performance. Since GNNs are inherently designed to capture topological/attributed relationships, they are powerful models that allow capturing the multiagent and multitask relationships relative to other models.

2) GNN, such as other DNN models, requires further improvements in terms of automatic setting generation, improving model explainability and enhancing robustness against adversarial attacks. These tasks can easily be handled via DRL due to inherent sequential nature. DRL is well-suited compared to traditional optimization-based approaches for these tasks since it offers a computationally lightweight framework to tackle large problem spaces in a scalable and generic way.

3) The performance of DRL in applications involving graph-structured environment, such as KGs and transportation networks, depends on the encoder to a large extent. Typically, in graph-structured data, the information is disseminated throughout the data space, and encoders learn to aggregate the information specific to the use cases. Therefore, GNNs are used to represent trajectory information in such environments and also act as a function approximator. GNNs are very effective in representing/encoding graphs compared to other techniques such as graph signal processing or spectral graph theoretic approaches. Furthermore, they are flexible and generic enough to work for different graph families and sizes.

B. Problem-Specific Applicability of DRL and GNN Methods

A fusion of GNN and DRL has found itself a set of niche problems that span diverse applications while sharing common features. These common features are: 1) sequential decision-making setting of the problem, wherein learning occurs via interactions with the environment in a closed loop manner; 2) the learning agent exploits its acquired knowledge at any time while also striking a balance between exploring multiple options for a potentially better solution; 3) the learning is aimed at achieving long-term goals and avoids making myopic decisions; and 4) the underlying system is most efficiently represented as a graph, thereby making GNNs the natural choice for representing such systems. A widely studied example of such a problem is the TSP, where the process of finding the optimal route is a sequential process of identifying nodes that lead to the minimum total distance traveled. Furthermore, the underlying problem possesses a graph structure with nodes being destinations and links representing connection between them.

The majority of applications in the literature involve static systems so that a single GNN module can serve as both a function approximator and an encoder for the environment. However, depending on the nature of the problem, an appropriate GNN algorithm must be chosen for the best performance. Environments involving large graphs should rely on GraphSAGE [37] rather than GCN [36], as GraphSAGE is a subgraph-based inductive learning approach that is scalable to larger networks. Similarly, applications where the position of a node with respect to the entire graph is vital, PGGNs [132] are preferred. PGGNs explicitly make use of anchor nodes along with neighboring subgraph to improve the effectiveness of node embeddings. Furthermore, the expressive power of most GNNs is upper bounded by the 1-Weisfeiler–Lehman (1-WL) graph isomorphism test, i.e., they cannot differentiate between different d-regular graphs [133]. Therefore, it is recommended to explore identity-aware GNN [134] for complex graph-structured environments, which inductively considers nodes’ identities during message passing.

Along with static graphs, there are certain applications that involve dynamic graph-structured environments. For instance, in a CAV network, the number of vehicles dynamically changes. Under this scenario, an appropriate strategy would be to use LSTMs fused with GNNs for capturing graph evolution as well as DRL trajectories. At any instant, the spatial information of the environment can be gathered via GNN and fed to an LSTM cell state for learning long-range spatiotemporal dependency. Furthermore, separate GNNs can be used to encode topological changes and long-range dependencies individually. Dynamic graphs can also be handled via novel graph neural odes (GDE) formulation, where input–output relationship is determined by a continuum of GNN layers, blending discrete topological structures and differential equations [135], [136]. The structure-dependent vector field learned by GDEs offers a data-driven approach to the modeling of dynamical networked systems, particularly when the governing equations are highly nonlinear and therefore challenging to approach with analytical methods [137]. Autoregressive GDE can adapt the prediction horizon by adjusting the integration interval of the ODE, allowing the model to track the evolution of the underlying system from irregular observations [135]. In addition to the type of environment, the problem of interest...
can have a single learning agent or multiple agents. In a multiagent application without any interaction between agents, traditional MADRL algorithms are most suited. However, in certain scenarios, the agents might interact with each other in search of a better solution. These interactions can further be predefined or might appear as an agent interacts with the environment. GNN models can be used to capture such relationships and provide auxiliary information to the agent in order to further improve the model performance. This also applies to multitask situations where different tasks are correlated or structurally related.

GNNs encompass various tuning parameters in their architecture. Thus, NAS in them is very effective. DRL algorithms such as DQN are an appropriate choice for searching since the search methodology is generic and applicable across different architectures. In fact, any applications involving search operations in GNN such as adversarial attack can be tackled very effectively with DRL. The use of multimodal data has led to heterogeneous graph-structured data in various applications, including KGs and recommender systems. Conventional GNNs are not designed to handle heterogeneity. Therefore, it is recommended to employ customized GNNs, such as relational GCN [138], heterogeneous GAT [139], and HetSANN [140] for encoding. Fundamentally, all these works demand separate aggregation and combination functions (model parameters) for each node/link type, i.e., node attributes or link relationships, so that a powerful node representation can be learned.

V. OPEN CHALLENGES AND RESEARCH OPPORTUNITIES

The articles surveyed in this work reveal the wide applicability and importance of fusing GNN and DRL (summarized in (Section IV)). This section identifies the challenges that lie ahead for widespread adoption and suggests future directions to unlock the full potential of a combined GNN-DRL framework. These challenges and opportunities are shown in Fig. 3. We identify two broad classes of challenges—those associated with fundamental algorithmic limitations of the existing approaches such as generalizability, explainability, and expressivity (interpretability), and the ones concerned with the transfer of techniques to real-life applications, e.g., standard benchmark design, sensitivity analyses, uncertainty, and reliability quantification.

A. Improving Generalizability With Emerging Properties of State-of-the-Art Frameworks

DNNs are notoriously challenging in terms of generalizing to different problems. This problem predominantly stems from the large size and hence models the capacity of modern DNNs. A combined GNN-DRL approach can fall into the pitfall of poor generalizability by not being able to generalize to slightly different environmental settings. For instance, changing the layout of environment (e.g., dynamic link connections in social network while solving combinatorial problem such as influence maximization) for a model that learns to navigate through the environment can result in the model being stuck in a segment of the network or to take large amount of time to navigate the network. This can be caused either due to the GNN or the DRL not being able to generalize to the environment.

One possible approach to address this challenge is to use graph meta RL framework where agents can quickly adapt to new tasks or environment with fewer samples [81]. Specifically, this can be achieved by providing context variables that vary with applications/environments. For example, in the case of CO, variations could be the problems that are smaller instances of the same problem, problem instances with different distributions, or even the ones from the other types of CO problems. Although certain generalization efforts can be seen recently, there is more to be done.

Another way to enhance the generalizability of DRL algorithms is by exposing agents to several graph environments that can be created with training graphs via graph augmentation techniques. One potential idea would be in leveraging GAN for graph augmentation by generating synthetic examples via perturbing input graphs in terms of adding/removing nodes and links or modifying node/link attributes [141], [142]. This enables the DRL agent to adapt by learning invariant features across varied and noisy environments. We believe that although this task is challenging, it is extremely important and a promising direction for research in hybrid DRL–GNN works. Moreover, recently, researchers brought the idea of decision transformers (similar to large pretrained language models such as BERT and GPT) to DRL algorithms [143]. This involves pretraining on a large set of environments and then fine-tuning on the specific application. It will be interesting to explore these pretraining and fine-tuning tasks for graph-structured environments.
B. Increasing Transparency by Incorporating XAI and Representation Learning

A substantial amount of XAI literature is emerging on feature relevance techniques to explain the predictions of a DNN or explaining models that consume image source data. However, it is unclear as to how XAI techniques can aid in understanding models beyond classification tasks, such as DRL. An improved interpretability and explainability of DRL (XRL) models could help shed light on the underlying mechanisms in situations where it is essential to justify and explain the agent’s behavior, which is still contemplated as a black box. The recent efforts on XRL are problem-specific and cannot be generalized to real-world RL tasks [144]. Explaining predictions of GNN and explaining reasoning behind the DRL actions are important and active area of research.

The concepts based on representation learning, such as hierarchical RL, self-attention, and hindsight experience replay, have been highlighted as a few encouraging approaches to improve explainability in DRL models [144] and can be considered as future lines of research. Furthermore, DRL is also employed for explaining node/link predictions in GNN by identifying the most influential subgraph [47]. The use of GNN to improve the prediction of DRL can also be investigated as a part of future research in this space.

C. Incorporating Network Constraints With Novel Formulations

Optimization problems for real-life applications are mostly bounded by a variety of constraints in terms of finance, time, resources, and so on. Most of the existing DRL works deal with constraints through penalties in rewards, which is appropriate if the constraints are soft, i.e., they can be violated at some cost. However, hard constraints must be strictly met and imposing a penalty cannot eliminate them and thus is not a perfect approach. An alternative way to deal with hard constraints is by masking the constraints while designing the training environment, to keep the exploration space away from constraint violation, as considered in autonomous driving [145], [146]. These constraints become further complex when the underlying environment is graph-structured, such as a transportation network. A few attempts have been made in this direction with remarkable performance such as reward shaping via hierarchical framework [147] and deferred MDP [148]. However, there is still a lot of scope to improve. Therefore, further research is needed to explore the rich literature of constrained dynamical systems and other strategies for dealing with hard constraints.

D. Adaptive Learning With Dynamic/Heterogeneous Environment

A majority of the existing GNN models perform prediction and inferencing over homogeneous graphs. However, a large number of real-world applications, such as critical infrastructure networks, recommendation systems, and social networks, involve learning on heterogeneous graphs. Heterogeneous graph-structured data can represent numerous types of entities (nodes) and relations (edges) within a common framework. It is difficult to handle such diverse graphs using existing GNN models. Consequently, developing new models and algorithms that are capable of learning with heterogeneous graphs would be highly beneficial in real-world systems, such as cybersecurity [149], text analysis [150], and recommendation engines [151]. Integrating the use of DRL techniques to achieve this goal can be considered as one of the possible future directions of research. Furthermore, existing GNN methodologies assume graph-structured data to be static, i.e., the possibility of addition and/or removal of nodes/edges is disregarded. However, many practical applications, such as social networks, encompass dynamic spatial relations that continuously evolve over time. Although spatial–temporal GNNs possess the ability to partially handle dynamic graphs [17], further work is required to integrate a thorough understanding about downstream tasks, such as node classification, link prediction, community detection, and graph classification in dynamic graphs.

E. Enabling a Seamless Transition to Real-World Networks

Most of the prevailing GNN-DRL methods are developed based on synthetic graph-structured datasets and simulated platforms. However, real-life scenarios can be far more intricate compared to simulated platforms. Thus, the resulting models need rigorous validation before being deployed confidently in real-life applications [152]. This validation is specifically more crucial for applications such as CAV and manufacturing process, where safety is of utmost importance. There are some attempts to transfer the trained DRL agent from a simulator to an actual test bed [153] in general DRL, but there is a significant gap for graph-structured environments in general. Therefore, seamless transition from a simulated setting to real-world scenarios in a cautious, protected and productive approach is an important future research direction. In this regard, more collaborative efforts from the industry and academia through partnership projects are required to understand the limitations of models in real-life applications and devise methods to improve the model’s performance. It is important to design standardized problem settings and benchmark datasets that can be used to evaluate and compare different methodologies for specific applications, enabling the faster development of the state-of-the-art solutions.

F. Assessing Transferability With Uncertainty and Reliability Quantification

From the perspective of practitioners, it is critical that any solution should be robust to slight variations in its inputs. Specifically, in the case of hybrid DRL–GNN approaches, it is important that the solutions should be robust to changes in the graph and environment. A sensitivity analysis of the model’s performance to data and environmental parameters can provide important insights into the stability of a model and help identify suitable modifications in terms of environment design, model specification, training process, and data fidelity among others to improve the real-life applicability of the model. In addition, quantifying the uncertainty of the model will allow an end user to calibrate and evaluate the
confidence in the model’s predictions. This can be achieved with Bayesian approaches such as Bayes by backpropagation, assumed density filtering, Monte Carlo dropout, or by ensemble methods [154]. While many of these studies have been performed for DNNs, a lot of progress is still required in DRL–GNN approaches to assess the applicability to real-life scenarios. Finally, generative models, such as GANs, can be used to generate different scenarios and potentially improve the model’s performance.

G. Computational Efficiency via Foundation Decision Models

It is widely known that DRL algorithms are very efficient during inference time, whereas they are quite sample inefficient (i.e., demand large interaction data for learning an effective policy) during training compared to linear/dynamic programming. Moreover, DRL also belongs to one of the few classes of algorithms where it is difficult to derive numerical bounds on the performance. This is further complicated by the function approximators such as GNN, which itself does not have strong theoretical support for convergence. Therefore, there are very few works in the direction with almost no strong theoretical result in the DRL–GNN hybrid approaches. Work [155] is one of the first works that show the asymptotic rate of convergence of standard Q-learning for discounted MDP as \( O(1/(t^{R(1-\lambda)})) \) if \( R(1-\lambda) < 1/2 \). Here, \( R = (P \min / P \max) \) is the ratio of the minimum and maximum state–action occupation frequencies. Furthermore, Potapov and Ali [156] showed how learning rate and their interactions with discount rate (\( \lambda \)) and exploration degree (\( \epsilon \)) influence the convergence rate of RL algorithms. They have few general claims as lower \( \lambda \) and larger exploration level \( \epsilon \) are better for convergence, but their optimal choice depends on the specific problem concerning the level of randomness that is deemed acceptable. In a nutshell, the computation time or convergence rate largely depends on the problem we are solving, considering various factors such as dimension of state action space, degree of alignment between the reward formulation and problem objective, and state/action encoding mechanism, among others.

Furthermore, with the growing parameter space of models [157], as well as the lack of formal complexity analysis in existing literature, it remains challenging to accurately assess the computational requirements of combined DRL and GNN models. The complexity of the hybrid models will be additive in nature, rather than multiplicative and, therefore, will be bounded by the larger of the two complexities of GNN or DRL models for a given problem setting. Nevertheless, one potential direction to counter the computational challenge and make it less sensitive to the specific use case lies in the concept of foundation model that recently emerged in the NLP and vision community [158]. The idea is to train a single large pretrained model using offline trajectory data, which can later be fine-tuned for several tasks/scenarios in a computationally (sample) efficient way [143], [159]. We believe that a single foundation model can also be developed for DRL–GNN paradigm with high few-/zero-shot learning capability. This framework can significantly reduce the computational burden imposed by training several problem-specific decision models.

VI. Summary and Conclusion

This article presents a systematic survey of literature focusing on works fusing DRL and GNN approaches. Although several reviews related to DRL have been published in recent years, most of these studies are limited to a particular application domain. This study, for the first time, presents a methodological review spanning diverse range of application domains. We have reviewed papers from the perspectives of both fundamental algorithmic enhancements as well as application-specific developments. From an algorithmic perspective, either a GNN is exploited to strengthen the formulation and improve performance of DRL or DRL is employed to expand the applicability of GNN. Recent works blending the usage of both DRL and GNN across multiple applications (broadly classified into CO, transportation, manufacturing and control, KGs, and life sciences) have been thoroughly investigated and discussed. We also highlight the key advantages of fusing DRL and GNN methodologies and outline the applicability of each of these components. Furthermore, we identify the challenges involved in effective integration of DRL and GNN and propose some potential future research directions in this area.

REFERENCES

[1] (2017). MIT Technology Review. [Online]. Available: https://www.technologyreview.com/10-breakthrough-technologies/

[2] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, “Reinforcement learning for combinatorial optimization: A survey,” Comput. Oper. Res., vol. 134, Oct. 2021, Art. no. 105400.

[3] V. Mnih et al., “Playing Atari with deep reinforcement learning,” 2013, arXiv:1312.5602.

[4] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 49–58.

[5] J. Luketina et al., “A survey of reinforcement learning informed by natural language,” 2019, arXiv:1906.03926.

[6] A. V. Bernstein and E. V. Burnaev, “Reinforcement learning in computer vision,” in Proc. 10th Int. Conf. Mach. Vis. (ICMV), Apr. 2018, pp. 458–464.

[7] N. Parvez Farazi, B. Zou, T. Ahamed, and L. Barua, “Deep reinforcement learning in transportation research: A survey,” Transp. Res. Interdiscipl. Perspect., vol. 11, Sep. 2021, Art. no. 100425.

[8] S. K. Zhou, H. N. Le, K. Luu, H. V. Nguyen, and N. Ayache, “Deep reinforcement learning in medical imaging: A literature review,” Med. Image Anal., vol. 73, Oct. 2021, Art. no. 102193.

[9] M. S. Frikiha, S. M. Gammar, A. Lahmadi, and L. Andrey, “Reinforcement and deep reinforcement learning for wireless Internet of Things: A survey,” Comput. Commun., vol. 178, pp. 98–113, Oct. 2021.

[10] R. N. Boute, J. Gijsbrechts, W. van Jaarsveld, and N. Vanvuchelen, “Deep reinforcement learning for inventory control: A roadmap,” Eur. J. Oper. Res., vol. 298, no. 2, pp. 401–412, Apr. 2022.

[11] T. Li, G. Peng, Q. Zhu, and T. Basar, “The confluence of networks, games, and learning a game-theoretic framework for multiagent decision making over networks,” IEEE Control Syst. vol. 42, no. 4, pp. 35–67, Aug. 2022.

[12] A. T. D. Perera and P. Kamalanabhan, “Applications of reinforcement learning in energy systems,” Renew. Sustain. Energy Rev., vol. 137, Mar. 2021, Art. no. 110618.

[13] M. Glavic, “Deep reinforcement learning for electric power system control and related problems: A short review and perspectives,” Ann. Rev. Control, vol. 48, pp. 22–35, Jan. 2019.

[14] M. Botvinick, J. X. Wang, W. Dabney, K. J. Miller, and Z. Kurth-Nelson, “Deep reinforcement learning and its neuroscientific implications,” Neuron, vol. 107, no. 4, pp. 603–616, Aug. 2020.

[15] J. Zhou et al., “Graph neural networks: A review of methods and applications,” AI Open, vol. 1, pp. 57–81, Jan. 2020.

[16] Y. Zhou, H. Zheng, X. Huang, S. Hao, D. Li, and J. Zhao, “Graph neural networks: Taxonomy, advances, and trends,” ACM Trans. Intell. Syst. Technol., vol. 13, no. 1, pp. 1–54, Feb. 2022.
[70] W. Huang, I. Mordatch, and D. Pathak, “One policy to control them all: Shared modular policies for agent-agnostic control,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 4455–4464.

[71] S. Muggleton and L. de Raedt, “Inductive logic programming: Theory and methods,” J. Log. Program., vols. 19–20, pp. 629–679, May 1994.

[72] V. Zam Baldi et al., “Relational deep reinforcement learning,” 2018, arXiv:1806.03511.

[73] S. Sanner, “Relational dynamic influence diagram language (RDDL): Language description,” Uncertainty Track Seventh Int. Planning Competition, National ICT Aust. Ltd, NICTA, Mach. Learn. Group, Tech. Rep., 2010, vol. 32, p. 27.

[74] S. Garg et al., “Symbolic network: Generalized neural policies for relational MDPs,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 3397–3407.

[75] J. Dornheim, N. Link, and P. Gumbsch, “Model-free adaptive optimization using graph neural networks,” 2020, arXiv:2009.12462.

[76] M. Fischetti and A. Lodi, “Heuristics in mixed integer programming,” Wiley Encyclopedia Oper. Res. Manage. Sci., Jun. 2010, [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/9780470400531.eorms0376

[77] S. Manchanda, A. Mittal, A. Dhanaw, S. Medya, S. Ranu, and A. K. Singh, “GCOMB: Learning budget-constrained combinatorial algorithms over billion-sized graphs,” in Proc. Conf. Neural Inf. Process. Syst. (NeurIPS), vol. 33, 2020, pp. 20000–20011.

[78] J. Huang, Q. Chang, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 1–11.

[79] H. Chen, W. Qiu, H.-C. Ou, B. An, and M. Tambe, “Contingency-aware influence maximization: A reinforcement learning approach,” in Proc. 37th Conf. Uncertainty Artif. Intell., 2021, pp. 1535–1545.

[80] S. Munikoti, L. Das, and B. Natarajan, “Bayesian graph neural network for fast identification of critical nodes in uncertain complex networks,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 145–150.

[81] S. Munikoti, B. Natarajan, and M. Halappanavar, “GraMeR: Graph construction using reinforcement learning,” in Proc. CIRP Ann. Manuf. Technol., vol. 59, 2022, pp. 1–15.

[82] H. Dai, D. Bai, and D. Song, “Discriminative embeddings of latent variable models for structured data,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 2702–2711.

[83] A. Ghosh and S. Boyd, “Growing well-connected graphs,” in Proc. 45th IEEE Conf. Decis. Control. 2006, pp. 6605–6611.

[84] S. Munkvad et al., “Solving spatially embedded graph neural network-based framework for identifying critical nodes and links in complex networks,” Neurocomputing, vol. 468, pp. 211–221, Jan. 2022.

[85] S. Munkvad et al., “Symbolic reinforcement learning for optimizing multiple traveling salesman problems over graphs,” Knowl.-Based Syst., vol. 204, Sep. 2020, Art. no. 106244.

[86] S. Chen, P. Jiang, X. Wang, and S. Yang, “A learning-based iterative method for solving vehicle routing problems,” in Proc. Int. Conf. Learn. Represent., 2019, pp. 1–15.

[87] X. He, Y. Han, B. Dai, and Z. Li, “A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs,” Knowl.-Based Syst., vol. 183, Nov. 2019, Art. no. 104855.

[88] F. Devaux, R. Darroque, and L. Charlin, “IG-RL: Inductive graph reinforcement learning for massive-scale traffic signal control,” IEEE Trans. Intell. Transp. Syst., vol. 23, no. 7, pp. 7496–7507, Jul. 2022.

[89] H. Lu, X. Zhang, and S. Yang, “A learning-based iterative method for solving vehicle routing problems,” in Proc. Int. Conf. Mach. Learn. Represent., 2019, pp. 1–15.

[90] L. Zhang, K. Chen, and W. Wu, “A reinforcement learning framework for identifying critical nodes and links in complex networks,” in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), 2022, pp. 5385–5392.

[91] J. Park, J. Park, and H. Yeo, “Transferable traffic signal control with multi-step return and off-policy asynchronous advantage actor-critic graph algorithm,” Knowl.-Based Syst., vol. 183, Nov. 2019, Art. no. 104855.
Deepesh Agarwal received the B.Tech. degree in electrical engineering from the Institute of Technology, Nirma University, Ahmedabad, India, in 2018, and the M.Tech. degree in electrical engineering from IIT Gandhinagar, Palaj, India, in 2020. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS, USA. His research interests include applications of statistical machine learning and inferencing in the context of human-in-the-loop decision systems.

Laya Das received the B.Tech. degree in electronics and telecommunications from the Biju Patnaik University of Technology, Rourkela, India, in 2011, and the M.Tech. and Ph.D. degrees in electrical engineering from IIT Gandhinagar, Palaj, India, in 2014 and 2014, respectively. He is currently a Post-Doctoral Researcher with the Reliability and Risk Engineering Laboratory, ETH Zürich, Zürich, Switzerland. His research interests lie in fundamental and applied artificial intelligence, geometric deep learning, representation learning, fault diagnosis, and monitoring of electrical systems.

Mahantesh Halappanavar (Senior Member, IEEE) is currently a Chief Data Scientist at PNNL, Richland, WA, USA, where he serves as the Group Lead of the Data Science and Machine Intelligence Group. He also holds a joint appointment as an Adjunct Faculty Member in computer science at the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA. His research has spanned multiple technical foci and includes combinatorial scientific computing, parallel graph algorithms, artificial intelligence and machine learning, and the application of graph theory and game theory to solve problems in application domains, such as scientific computing, power grids, cybersecurity, and life sciences. He has coauthored a book on the design of parallel graph algorithms on shared-memory architectures and has authored over 125 technical publications for peer-reviewed journals, conferences, and workshops.

Mr. Halappanavar is a member of the Society for Industrial and Applied Mathematics (SIAM) and a Senior Member of the Association for Computing Machinery (ACM).

Balasubramaniam Natarajan (Senior Member, IEEE) received the B.E. degree (Hons.) in electrical and electronics engineering from the Birla Institute of Technology and Science at Pilani, Pilani, India, in 1997, the Ph.D. degree in electrical engineering from Colorado State University, Fort Collins, CO, USA, in 2002, and the Ph.D. degree in statistics from Kansas State University, Manhattan, KS, USA, in 2018. He is currently a Clair N. Palmer and Sara M. Palmer Endowed Professor and the Director of the Wireless Communication and Information Processing Research Group, Kansas State University. He has worked on and published extensively on modeling, analysis and networked estimation, and the control of smart distribution grids and cyber-physical systems in general. He has published more than 200 refereed journal articles and conference papers. His research interests include statistical signal processing, stochastic modeling, optimization, and control theories.

Dr. Natarajan has served on the Editorial Board for multiple IEEE journals, including the IEEE Transactions on Wireless Communications.