IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 13 October 2024, accepted 8 November 2024, date of publication 14 November 2024,
date of current version 26 November 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3497955

RESEARCH ARTICLE

Reinforcement Learning-Based Formulations With Hamiltonian-Inspired Loss Functions for Combinatorial Optimization Over Graphs

REDWAN AHMED RIZVEE, RAHEEB HASSAN, AND MD. MOSADDEK KHAN
Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh
Corresponding author: Redwan Ahmed Rizvee (rizvee@cse.du.ac.bd)

ABSTRACT Quadratic Unconstrained Binary Optimization (QUBO) is a versatile approach used to represent a wide range of NP-hard Combinatorial Optimization (CO) problems through binary variables. The transformation of QUBO to an Ising Hamiltonian is recognized as an effective method for solving key optimization problems using quantum algorithms. Recently, PI-GNN, a generic framework, has been proposed to address CO problems over graphs based on QUBO with Hamiltonian loss function to train the underlying GNN architecture. Though PI-GNN is highly scalable, it exhibits a noticeable decrease in terms of the number of satisfied constraints with higher graph densities. In this paper, firstly, we identify the limitations and empirically present our strategy to improve PI-GNN's performance. Secondly, we formulate and evaluate two strategies to integrate QUBO-Hamiltonian as the generic loss function in Reinforcement learning-based (RL) frameworks. The major contribution of our work lies in understanding the feasibility and quality of the QUBO-based generic reward function in an unsupervised RL setup in addressing graph-based CO problems. Empirically, through our empirical evaluation (Our implementation can be found in https://tinyurl.com/5apnymz7), we have observed up to 44% improvement in terms of the number of satisfied constraints over PI-GNN in a representative Max-Cut problem.

INDEX TERMS Deep reinforcement learning, graph neural network, Hamiltonian function, Monte Carlo tree search.

I. INTRODUCTION

Combinatorial Optimization (CO) deals with finding the best possible solution from a finite set of possibilities to optimize some objective function subject to a set of constraints. A key challenge associated with CO is that, as the problem size grows, the number of possible solutions increases exponentially, making it computationally infeasible to explore all possibilities. Therefore, CO seeks efficient algorithms and techniques to find near-optimal solutions. Due to the definition and nature of the problem, CO has found its applications in numerous domains, including logistics, finance, and artificial intelligence [1], [2].

One powerful framework for addressing CO problems is Quadratic Unconstrained Binary Optimization (QUBO), which transforms complex combinatorial problems into a quadratic equation involving binary variables [3]. QUBO has become the standard format for optimization using quantum computers [4], [5]. QUBO and Ising Hamiltonian have a very close relationship in modeling a diverse set of optimization problems and are often used interchangeably or together [3]. Ising Hamiltonian focuses on modeling the energy stability of a system in the form of identifying the spins ($\{-1, 1\}$) of the qubits. After converting a QUBO objective or loss function into an Ising Hamiltonian, various quantum optimization algorithms are used to solve it [6]. Due to this close connection, the relevant problems are addressed as problem Hamiltonian, and the objective functions are regarded as Hamiltonian Objective or Cost function [3], [7].

In 1, we present the generic formation of QUBO-based Hamiltonian, H_{QUBO}. Here, Q represents the encoded problem in a matrix bearing the constraints, and X =

The associate editor coordinating the review of this manuscript and approving it for publication was Frederico Guimarães.

$(x_1, x_2, x_3, \ldots, x_n)$ represents the binary decision variables or node labels for a graph having n nodes, from the point of view of CO. With a higher number of satisfied constraints, this generic objective function provides a higher value. Research has shown that quantum solution systems struggle to scale with the number of variables and to maintain precision in the solutions they provide [7]. Thus, recent works have been addressing different strategies to model QUBO-based Hamiltonians [3].

$$H_{\text{QUBO}} = \boldsymbol{X}^T \boldsymbol{Q} \boldsymbol{X} = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i Q_{ij} x_j, \tag{1}$$

Over the past few years, Graph Neural Networks (GNNs) have risen in popularity due to their ability to analyze and process graph-structured data while capturing complex relationships across a diverse array of applications [8], [9], [10]. As stated in 2, in a particular layer (k) of GNN, it updates a node's (v) feature vector (h_v^k) by combining the messages or feature vectors of its adjacent nodes $(h_u^{k-1}, u \in \mathcal{N}(v))$ and itself (h_v^{k-1}) of the previous layer $(k-1)$. Here f represents non-linearity.

$$h_v^k = f_\theta^k(h_v^{k-1}, \{h_u^{k-1} | u \in \mathcal{N}(v)\}), \tag{2}$$

The strategies for combining the messages differ in different variations of GNN. For example, in 3 and 4, we present the node feature vector update rules for Graph Convolutional Network (GCN) and Graph Attention Network (GAT), respectively. Here, W and B denote the learnable weight and bias parameters, respectively. The α shown in 4 denotes the attention coefficients of GAT and is usually learned by the relationship shown in 5 [11].

$$h_v^k = \sigma(W_k \sum_{u \in \mathcal{N}(v)} \frac{h_u^{k-1}}{|\mathcal{N}(v)|} + B_k h_v^{k-1}), \tag{3}$$

$$h_v^k = \alpha_{vv} W^k h_v^{k-1} + \sum_{u \in \mathcal{N}(v)} \alpha_{vu} W^k h_u^{k-1} \tag{4}$$

In 5 Z^k denotes another learnable parameter associated with the attention mechanism for k^{th} layer.

$$\alpha_{vu}^k = \frac{\exp(\sigma(z^{k^T} \left[W^k h_v^k, W^k h_u^k\right]))}{\sum_{w \in \mathcal{N}(v)} \exp(\sigma(z^{k^T} \left[W^k h_v^k, W^k h_w^k\right]))}. \tag{5}$$

When it comes to addressing CO problems, [3] has introduced a pioneering approach (PI-GNN) harnessing the power of GNNs to address the usage of Hamiltonian cost function during training. They showcase how GNNs can effectively model a QUBO formulation in terms of generating node probability distribution. The key innovation lies in their utilization of a relaxation strategy applied to the problem Hamiltonian, resulting in a differentiable loss function that serves as the foundation for unsupervised training. Their proposal was generic, very scalable, and provided on-par performance with various existing problem-specific solvers. In 6, we provide the concerned loss function (L_{QUBO}) to train the respective GNN architecture, inspired from QUBO-Hamiltonian, stated in 1. Here $p_i(\theta)$ denotes the node probability of the i^{th} node reported by the final layer of GNN which is often a softmax layer. PI-GNN falls into the unsupervised domain in addressing CO problems compared to the other counterpart of supervised algorithms, which require a good amount of labeled data and often provides challenges [23], [24].

$$H_{\text{QUBO}} \rightarrow L_{\text{QUBO}}(\theta) = \sum_{i,j} p_i(\theta) Q_{ij} p_j(\theta), \tag{6}$$

In a recent critical review paper by Boettcher [12], it was demonstrated that PI-GNN, performs less effectively than traditional greedy algorithms when solving the Maximum Cut or Max-Cut problem. Angelini and Ricci-Tersenghi [13] raised similar concerns about GNN-based solutions, particularly their relative inefficiency compared to classical greedy algorithms, as discussed in the context of the Maximum Independent Set (MIS) problem. Both pieces of literature highlight the notion that in very narrowly defined problem scenarios, such as Max-Cut and MIS, traditional greedy algorithms may outperform PI-GNN. However, it is noteworthy that the authors of PI-GNN have responded to these critiques in their subsequent work [7], arguing that focusing solely on the performance in these specific curated scenarios overlooks the broader generality and scalability of their proposed framework. To bolster their perspective, they have presented empirical results that illustrate the advantages of PI-GNN in more general and scalable problem settings, suggesting that a comprehensive evaluation of the framework is necessary to appreciate its full merit.

On the other hand, another widely explored approach to solving optimization problems is Reinforcement Learning (RL). From traditional methods of RL to quantum algorithms, RL has found its application in numerous problems and applications [28], [29], [30], [31]. Reference [31] has provided a summarized article denoting the applicability, performance, and challenges related to applying different RL algorithms in the context of energy systems. Lin et al. [30] proposed a deep RL-based quantum adiabatic algorithm to address NP-complete problems. Their main contribution was that their proposed RL algorithm automatically generates an adiabatic algorithm that brings significant improvement in the resultant success probability. Reference [29] shows how a quantum environment can be combined with deep RL to significantly enhance the performance in terms of speed-ups for large action spaces, which is often a crucial bottleneck for RL problems. Reference [28] addresses the satellite planning problem in space by combining quantum algorithms with RL and obtaining significantly better performance than classical greedy algorithms.

Specifically, RL aims to learn policies that maximize expected rewards, which can often be derived from the objective function of the optimization problem [14], [25]. In this setting, an agent's states represent current solutions, actions correspond to decisions, and rewards measure solution

quality. RL algorithms like Q-learning or deep reinforcement learning (DRL) methods guide the agent to explore and exploit solutions efficiently [25]. Reference [15] extended the pointer network architecture to create an actor-critic RL framework for training an approximate Traveling Salesman Problem (TSP) solver, using tour length as a reward signal. Subsequent work [16] introduced improvements in accuracy using a graph attention network for two-dimensional Euclidean TSP, approaching optimal solutions for problems of up to 100 nodes. Additionally, a multi-level RL framework was used to analyze TSP variants with hard constraints. In terms of a general solving framework, [17] proposed a technique that uses a GAT base encoder architecture to encode and generate the node feature vectors, upon which they apply an attention-based decoding mechanism to greedily select the nodes and apply the node labelings. The approach, however, requires reward functions to be specifically tailored for each problem and thus hampers generality.

Taking the previous discussion into account, it is obvious that trivial GNN-based methods such as PI-GNN, although widely applicable and versatile, often struggle with accuracy. Meanwhile, despite their high accuracy and scalability, RL-based methods are not as widely applicable. In this paper, we try to bridge this gap by addressing the CO problem by extending some of the aforementioned methodologies. In particular, we suggest two distinct improvements to this manuscript:

1) Firstly, we identify a critical flaw in the early stopping strategy of PI-GNN, which leads to underperformance as the density of graphs increases. In this regard, we state plausible reasoning and experiment with a fuzzy early-stopping strategy as an alternative to the fixed tolerance value strategy found in their work. Empirically and analytically, we present how the fuzzy early-stopping strategy improves the performance of PI-GNN.
2) Secondly, we formulate generic loss functions integrating the QUBO-Hamiltonian in the form of rewards in RL-based setups to address the compatibility. In this regard, we also design two RL-based approaches - An Encoder-Decoder with a GAT-based Generic Reinforcement Learning Strategy (GRL_{QUBO}) and A Monte Carlo Tree Search with a GNN-based Strategy (MCTS-GNN). This ascribes the major contribution of this manuscript to understanding the feasibility of a QUBO-based generic reward function of RL setups for CO problems.

In terms of underlying contribution, this article tries to understand how a generic Hamiltonian QUBO-based loss function in an unsupervised RL setup can guide the solution quality and convergence pattern of graph-based CO problems, which has been considered a crucial metric in different literature [25]. We have evaluated our architectures and strategies in the Maximum Cut (Max-Cut) problem as a representative problem of CO and achieved up to 44% improvement in terms of the number of satisfied constraints compared to PI-GNN.

The remaining discussion of this manuscript can be summarized as follows. In Section II, we formulate and discuss the problem in a generalized manner that we have addressed in this manuscript. In Section III, we present our proposals, including the mathematical formulations and architectural descriptions. In Section IV, we present the results and a follow-up discussion by evaluating the proposals' merits in various metrics. Finally, we conclude the study by presenting some concluding remarks in Section V.

II. PROBLEM FORMULATION

The first challenge of this manuscript is to combine the idea of QUBO-Hamiltonian with RL formulation, and the second challenge will be to design corresponding architectures. Traditionally, RL-based setups circle around the terminology of States (s), Actions (a), and Reward (r). In this setup of CO problems in graphs, we denote s as the set of two binary vectors, X_B and X_L of n variables where n denotes the number of nodes in the input graph G, $X_B \in \{0,1\}^n$ and $X_L \in \{0,1\}^n$. For a node v, if the corresponding entry in X_B ($X_B[v]$) is set to 1, it will mean that node v has already been labeled, and the label will be found in $X_L[v]$ as either 0 or 1, keeping relevance with the CO problems.

From s, an action a will be selecting a node v and labeling it with either 0 or 1 where $X_B[v] = 0$. This will create a new state s', where with other values of X_B and X_L in s we update with newly values for $X_B[v] = 1$ and $X_L[v]$. With each such action a, there will be a reward r, which will be a function of H_{QUBO} stated in 1. Mathematically, we are interested in designing a loss function for any arbitrary state s in the RL setup that combines the idea of reward r with the parameters of the architecture. Additionally, r will be a function of H_{QUBO} associated with the problem-encoded matrix Q and the binary variables X_B and X_L. In this literature as per convention, in the equations, X_B and X_L are represented in bold to denote their vector representations.

$$L_s(\theta, r) = L_s(\theta, f(Q, \boldsymbol{X_B}, \boldsymbol{X_L}, x_v = \{0,1\})) \tag{7}$$

The following sections discuss how we formulate the f function in different architectures to solve the CO problem. Another point to note here is that the loss function presented in 7 should work for any arbitrary state, including the terminal state.

For ease of understanding, in Table 1, we present the important notations that are repeatedly used in the manuscript to present the ideas.

III. OUR PROPOSALS

This section delineates all the proposals and formulations presented in our article. First, we discuss PI-GNN [3], the base literature upon which we build our contribution. First, we briefly present the idea of PI-GNN, then we discuss the crucial concerns related to it and present our proposals to address the issues. Followingly, we present

TABLE 1. Notations.

Notations	Descriptions
Q	QUBO encoded matrix
θ, ϕ	Architecture Parameters
$L_s(\theta, r)$	Generic Loss function for RL-based formulation
$(.)^T$	Matrix transpose operation
$[.]$	Accessing an index of a vector
r_t	Reward obtained at or up to the t^{th} timestamp or iteration
x_v	When node v is selected node, $x_v = 1$ else $x_v = 0$
X_B, X_L	Binary vectors $X_B[i] = 1$ denotes i^{th} node has been labeled $X_L[i] = 1$ denotes i^{th} node has been labeled as 1
h_v	Embedding or feature vector of node v
h, h'	Bundle of all the node embeddings or feature vectors
$p_i(\theta)$	Probability of node i reported by an archicture

two RL-based setups, a GAT with encoder-decoder-based architecture inspired by the work of [17] and a Monte Carlo Tree Search with a single GNN-based solution. For both of the strategies, we discuss how we formulate the loss function stated in 7 and the corresponding architectural design.

A. PI-GNN AND PI-GNN WITH FUZZY STRATEGY

PI-GNN uses K layers of GCN architecture with a softmax layer at the end to generate the node probability distributions $p(\theta)$ to calculate the loss function stated in 6. These node probability distributions are directly used to generate the final set of binary node labels. Additionally, PI-GNN uses the Strict Stopping strategy stated in Definition 1 to early stop the training. There are some important underlying concerns. Such as,

1) Absence of node projection strategy: The actual binary node projection strategy from the node probability distribution is absent in the loss function of 6. It is quite plausible that the set of node probability values from which the loss function has converged does not achieve a good number of satisfied constraints after projecting to the actual node labels based on some projection strategy.
2) Definition 1 states that when there is no improvement or very insignificant improvement in the loss function that can be observed for a consecutive number of epochs, the training can be stopped. In general cases, this observation produces a good result. However, as per our experiments, an insignificant change does not provide a guarantee that significant variation in the loss function would not occur in the next epochs. Empirically, we have observed that, this strategy deteriorates the performance of PI-GNN, specially in graphs with higher densities.

Definition 1 (Strict Stopping): During training, for the loss function $L_{QUBO}(\theta)$, if no successive improvement is observed for consecutive p epochs or the successive variation is lesser than τ occurs for consecutive p epochs, the training can be stopped.

As per our findings, during PI-GNN training, the objective function often starts with a high positive value and then, with gradual training, moves into larger negative values. During the transition from positive to negative loss, there is often a period when the loss variation or improvement is minimal (e.g., $\leq 10^{-7}$). As a result, the τ value outlined in Definition 1 may prompt an early halt to training, which can significantly diminish performance quality. Therefore, we suggest the use of a fuzzy-stopping strategy as formally described in Definition 2, which provides a more lenient stopping criterion. The rationale for the gradual loss variation is that nodes with a large number of connected edges tend to have feature vectors that are closely clustered in the vector space, leading to a uniform distribution of node probabilities. With additional iterations, these feature vectors are refined, resulting in more distinct patterns. It is also important to note that pinpointing an improved tolerance value during this gradual transition is challenging, rendering the early stopping approach in Definition 1 less effective.

Definition 2 (Fuzzy Stopping): Assume L^*_{QUBO}, represents the optimal value observed for the loss function L_{QUBO} at any point during training iterations. If L_{QUBO} does not show any improvement over L^*_{QUBO} for consecutive p epochs, it is advisable to terminate the training process.

It is apparent that Definition 2 provides a fuzzier criterion compared to Definition 1. This is because the latter allows for any sort of transition or improvements in the loss objective, including both faster and slower transitions during training, by removing the strict dependency over τ. The results presented in Section IV of PI-GNN are generated using this strategy.

In the following two sections, we describe two RL-based setups along with discussions of how we work with the loss function L_s stated in 7 for each architecture.

B. AN ENCODER-DECODER WITH GAT-BASED GENERIC RL FRAMEWORK FOR CO:(GRL_{QUBO})

This architecture is inspired by the work of [17]. We have adopted their base architectural design in our work and then modified specific portions of it along with designing the loss function as per 7. As the proposed work of [17] is also a generic framework, for discussion brevity, we address it as *GRL* and, as per with it, address our modified version as GRL_{QUBO}. Like *GRL*, GRL_{QUBO} also follows a model-free policy optimization-based RL approach, which is stochastic in nature. Through policy gradients, we update the networks' parameters through a loss function. As per novelty with *GRL*, GRL_{QUBO} devises a QUBO-Hamiltonian based generic reward function that partially accumulates the rewards in each step.

In major terms, there are mainly four segments that can be addressed to discuss *GRL* and GRL_{QUBO}. We discuss them through the following points,

1) **GAT as an encoder architecture:** We employ GAT as the encoder architecture, utilizing K stacked layers to generate the node feature vectors h. Its last layer consists of a sigmoid non-linear activation layer to

generate the node probabilities. These probabilities help to select the first node to initiate the decoding with the attention mechanism. We directly adopt this from *GRL* to our GRL_{QUBO}.

2) **Calculation of Reward (r) and Loss function (L_s):** GRL_{QUBO} has a significant amount of modification in the calculation of r and L_s compared to *GRL*. As per the discussion presented for *GRL*, for each specific problem, they have to manually design a score or value for each specific state along with another set of searches to check the validity of the problem's constraints. Their final loss function can be found in 8. Here, a_t and r_t denote the node chosen at the t^{th} step of decoding and the corresponding additional reward achieved at that time stamp t. b denotes a baseline value and can be calculated using some problem-specific heuristic algorithms. $p(a_t)$ denotes the probability of choosing action a_t at time t.

$$L(\theta) = \sum_{t=1}^{n} (r_t - b) \times p(a_t) \tag{8}$$

In conjunction with 7, we modify the reward and loss function as follows. Let us assume we are concerned with the loss at state s, at the t^{th} step of sequential decoding where we have selected node v (e.g., $a_t = v$) and achieved reward r_t. How r_t is calculated using QUBO-Hamiltonian is expressed via Equations 10 to 13. The underlying idea is that in this setup, we interpret different segments or collections of terms of 1 as different amounts of rewards at different states. We use 1, in a permutation invariant manner and as a generic reward objective for any QUBO-Hamiltonian formulated problem to the corresponding RL setups for CO. In other words, this approach iteratively accumulates the rewards, gradually extend the loss function, guide it for the following steps [24], [25], [26].

After selecting a node v and setting up its label, we accumulate the newly calculatable terms of 1 to consider them as rewards. The loss objective of 9 is generic for any state, including the final state, quite similar to 8 where we have relaxed the baseline variable b for simplicity and to reduce dependency over heuristical algorithms. Upon experimentation, we have observed competitive performance in this setup which we will present in the upcoming sections. Aligning with *GRL*, we conduct the backpropagation after accumulating all the results at the end of sequential decoding for an episode.

$$L_s(\theta, r_t) = \sum_{t=1}^{n} r_t \times p(a_t) \tag{9}$$

$$r_t = f(Q, \boldsymbol{X_B}, \boldsymbol{X_L}, x_{v,t}) \tag{10}$$

$$= r_{t,v\leq j} + r_{t,j<v} \tag{11}$$

$$r_{t,v\leq j} = \sum_{\forall v\leq j, X_B[j]=1} X_L[v] Q_{vj} X_L[j] \tag{12}$$

$$r_{t,j<v} = \sum_{\forall j<v, X_B[j]=1} X_L[j] Q_{jv} X_L[v] \tag{13}$$

3) **Attention-based decoding strategy:** The Decoding step is used to sequentially select a node v, to label it, and to expand the solution state. In 14, we provide the mathematical formulation to calculate the attention or weight for node v as α_v. Here, $\phi_1 \in \mathbb{R}^{d_h \times d}$, $\phi_2 \in \mathbb{R}^{d_h \times d}$ and $\phi_3 \in \mathbb{R}^{d_h \times n}$ are weights or architecture parameters. h_v denotes the node feature vector (row vector) for the node v reported from the GAT layer. C, d and d_h all are hyperparameters. n denotes the number of nodes in the input graph. Through applying a sigmoid non-linear activation over α_v, we generate node probability distribution $p_v(\theta)$. Over $p_v(\theta)$, we apply a probability threshold β (*e.g.*, $\beta = 0.5$) to fix the node labels, e.g., $p_v(\theta) \geq \beta$ leads to $x_i = 1$. Here θ denotes the complete set of trainable architecture parameters and T is used to denote the transpose of a vector.

$$\alpha_v = C \tanh\left(\frac{h_v \phi_1^T \cdot (\boldsymbol{X_B} \phi_3^T + \sum_{j\in N(v)} h_j \phi_2^T)}{\sqrt{d_h}}\right) \tag{14}$$

$$\alpha_{vu}^{\text{dec}} = C \tanh\left(\frac{(\phi_1 h_v)^T \cdot (\phi_2 h_u)}{\sqrt{d_h}}\right) \tag{15}$$

In 14, we expand upon the context compared to 15 for updating the attention weight α_v of a node v. It is important to note that 15 details the decoding strategy of GRL, where we calculate the attention weight between two sets of nodes u and v over their feature vectors (h_u, h_v). Conceptually, 14 is more robust than 15 due to the incorporation of all the adjacent nodes ($j \in N(v)$) along with the current status of the nodes that have been greedily selected (X_B) in addition to the feature vectors (h_v, h_j). Based on these decoding scores α, we select the node u with the highest $p(\alpha_u)$ score, label it ($X_L[u]$) as per thresholding β and update $X_B[u]$ to 1.

A system diagram representing the solution has been provided in Figure 1. To present the idea textually, a graph is first given as input to the GAT encoding layer (St #1). Then, node embeddings and a node probability distribution are generated (St #2). These values are passed to a decoder (St #3, St #4). Then, sequentially, nodes are selected, and partial rewards are accumulated (St #5). Finally, based on the loss function, backpropagation is conducted to update the weight parameters of the Decoder and Encoder (St #7.1, St #7.2). By this, we complete a single iteration of a training episode.

C. MONTE CARLO TREE SEARCH WITH GNN, MCTS-GNN

In this section, we discuss how we can incorporate the Monte Carlo Tree Search (MCTS) strategy to train a single GNN to generate the node probability distributions upon building the loss function of 7. Here our main research objective was to understand how using manual perturbation or a hardcoded

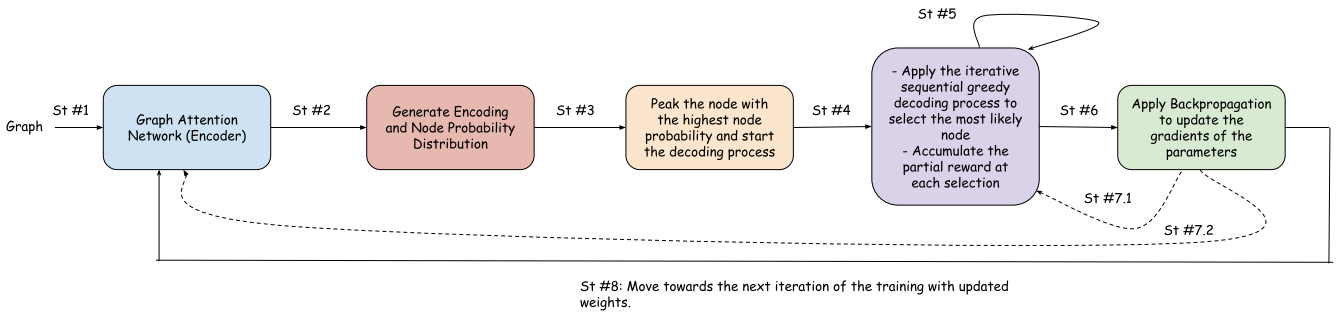

FIGURE 1. System diagram for the proposed GRL_{QUBO}.

set of node labeling can guide the training to improve the final set of results with a better number of satisfied constraints.

There are some common terminologies associated with MCTS. Each node of the MC Tree represents a state (s) or a partial set of solutions. Through a *Selection* measure, we choose the most likely child state or node to *Explore* in its subtree. After reaching a leaf state l, *Rollout* phase is conducted to approximate a possible amount of reward or utility achievable from l, and its child states are formed for future iterations for different actions a with a transition probability of $\pi(l, a)$. After approximating the reward r, it is backpropagated in the path to the root node. As per the terminology of MCTS in conjunction with RL, in our approach, we accumulate the average reward in each state of the Monte Carlo Search Tree using already labeled nodes and by approximating the unlabeled nodes from a single GNN in the rollout phase. The tree node transition probabilities are also used from the values reported by GNN. In the following paragraphs, we describe these points in detail.

The idea of state s and action a has already been highlighted in Section II along with the discussion of status and label vectors X_B and X_L, respectively. For each state s, the transition probabilities $\pi(s, a)$ and reward r are approximated using a single GNN in conjunction with QUBO-Hamiltonian and manual perturbation of node labels.

$$L_s(\theta, r) = L_s(\theta, f(Q, \boldsymbol{X_B}, \boldsymbol{X_L}, x_v)) \tag{16}$$
$$= L_s(\theta, f(Q, \boldsymbol{X_B}, \boldsymbol{X_L})) \tag{17}$$
$$= r_{FL} + r_{ML1} + r_{ML2} + r_{UL} \tag{18}$$
$$r_{FL} = \sum_{\forall i \le j, X_B[i]=X_B[j]=1} X_L[i] Q_{ij} X_L[j] \tag{19}$$
$$r_{ML1} = \sum_{\forall i \le j, X_B[i]=1, X_B[j]=0} X_L[i] Q_{ij} p_j(\theta) \tag{20}$$
$$r_{ML2} = \sum_{\forall i \le j, X_B[i]=0, X_B[j]=1} p_i(\theta) Q_{ij} X_B[j] \tag{21}$$
$$r_{UL} = \sum_{\forall i \le j, X_B[i]=0, X_B[j]=0} p_i(\theta) Q_{ij} p_j(\theta) \tag{22}$$

In Equations 16 - 22, we present the idea of corresponding loss and reward objectives for any arbitrary state s. The loss consists of four types of reward formulations, rewards from the set of fully labeled input variables (r_{FL}), from the set of unlabeled variables (r_{UL}), and rewards from the set of mixed type of variables (r_{ML1}, $r_{ML}2$). Due to the presence of the architecture parameters and being differentiable, the loss can be directly used to update the GNN architecture parameters through gradient descent in backpropagation.

As is evident, GNN plays a very important part in this design. We now present the mathematical formulation of GNN's forward pass to understand how $p(\theta)$ is generated.

$$X_{em} = E(G) \tag{23}$$
$$h' = GNN(G, X_{em}) \tag{24}$$
$$h = f_1(\boldsymbol{X_B}\theta_1^T + h'\theta_2) \tag{25}$$
$$p(\theta) = f_2(h\theta_3^T) \tag{26}$$

The complete set of formulations is presented from Equations 23 to 26. First, a set of node embedding vectors X_{em} is generated (Equation 23). Then, X_{em} and input graph G is passed to GNN to generate a set of node feature vectors h' (Equation 24). After that, more contextual information (X_B) is added with feature vector (h') to calculate the complete node feature vectors h (Equation 25). 26 represents the final equation to generate the node probability distribution $p(\theta)$. Here the set of architecture parameters $\theta = \{\theta_{GNN} \in \mathbb{R}^{d_2 \times d_1}, \theta_1 \in \mathbb{R}^{d_3 \times 1}, \theta_2 \in \mathbb{R}^{d_3 \times d_2}, \theta_3 \in \mathbb{R}^{1 \times d_3}\}$. We use θ_{GNN} to denote all the weight parameters associated with the layers of GNN. Here, f_1 and f_2 denotes ReLU and Sigmoid activation functions, respectively where f_2 is used to generate the probabilities. Apart from the node probabilities, we also use $p(\theta)$ to update $\pi(s, v)$ considering $p(\theta)$ to be the likelihood for labeling v as 1, e.g., $\pi(s, v = 1) = p_v(\theta)$ and $\pi(s, v = 0) = 1 - p_v(\theta)$. From each state S, we create child nodes for all v for all labels where $X_B[v] = 0$.

As it can be understood, using Equations 16 - 22, we apply guided search to a single GNN through manual perturbation of node labels. We hypothesize that this strategy enforces the trainable GNN to tackle different noises and make itself learn weights to avoid different local minima, resulting in improved overall reward objective. Now, to conclude the discussion, we map our previously mentioned strategies with the MCTS steps.

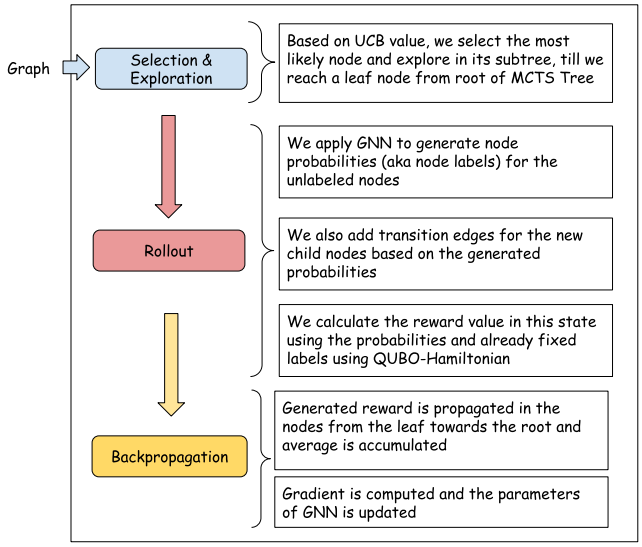

FIGURE 2. System diagram for the proposed MCTS-GNN.

From each non-leaf state S, we use 27 to select the most likely child node C_i based on the Upper Confidence Bound (UCB) metric in the *Selection* step over which we decide to explore further. Without a bound measure, the exploration in the underlying search space will be intractable, so the most likely node as per the measure is selected [27]. Here v and r_t denote the number of times a state is visited and the total amount of reward accumulated during different iterations, respectively. γ denotes the exploration hyperparameter. After selection, we continue *Exploration* in C_i's subtree in a similar fashion until we reach a leaf node l. There, we apply *Rollout* phase using GNN to update the parameters θ and calculate $\pi(S, a)$ denoting the probability of reaching state C_i from S by taking action a. After approximating the reward (r_t) using GNN (manual labeling and unsupervised generation), this value is propagated in the corresponding path to the root.

$$UCB(C_i) = \frac{C_i.r_t}{C_i.v} + \gamma * \pi(S, a) * \sqrt{\frac{\log(S.v)}{C_i.v}} \tag{27}$$

In Figure 2, we present the complete idea of MCTS-GNN using a system diagram to have a summarized representation of the flow of the proposed solution.

IV. EVALUATION

In this section, we empirically evaluate GRL_{QUBO} and MCTS-GNN with PI-GNN in addressing the Max-Cut problem for the graphs of different densities. The graphs were randomly generated, and all the graphs were undirected. For a given graph of n nodes, there can be at most $\frac{n\times(n-1)}{2}$ edges (E) where any two nodes do not have multi-edges among them. Statistics over the graphs based on their adjacency information are shown in Table 2 denoting the maximum (max_E), minimum (min_E), and average number of adjacent edges (avg_E) for each of the nodes for all the experimented graphs. We conduct the experiments based on three metrics - the number of satisfied constraints, training stability, and required time. In the Max-Cut problem, we try to divide the set of input graph nodes into two groups (0 and 1) so that the number of edges (or cut) between the two groups is maximized). All the experiments were conducted on a 64-bit machine with an AMD Ryzen 9 5950 × 16-Core Processor x 32, 128 GB RAM, and 24 GB NVIDIA GeForce RTX 3090 GPU. All the implementations were done in Python language based on the blocks provided by deep-learning libraries Pytorch [18], Pytorch Geometric [19] and LabML [20].

TABLE 2. Graph distribution statistics.

n	E	max_E	min_E	avg_E
50	89	11	1	3.56
	139	10	2	5.56
	499	27	13	19.96
100	199	11	1	3.98
	799	24	8	15.98
300	399	7	1	2.66
	899	13	1	5.99
	1299	18	1	8.66
500	799	10	1	3.2
	1499	13	1	6
	5499	45	10	22
700	1199	9	1	3.43
	1699	13	1	4.85
	4699	24	4	13.43
1000	1299	8	1	2.6
	3299	16	1	6.6
	5299	21	1	10.6
3000	3499	8	1	2.33
	4499	9	1	3
	6999	13	1	4.67

A. ARCHITECTURE DESCRIPTION, HYPERPARAMETER SETUP

In this section, we present a description of the architectures along with the value of the hyperparameters that were used during the training.

1) **Architecture Description:** To present the discussion, we mostly use the variables stated in the respective sections.
Our GNN architecture follows the similar definition provided in PI-GNN. It consists of two layers of GCN and a sigmoid layer to generate the node probabilities. The first GCN layer's node feature vectors are propagated through a non-linear ELU activation and a dropout layer before passing to the second layer of GCN. Let us assume the node feature vector size of the 1^{st} and 2^{nd} layers of GCN are d_1 and d_2. As mentioned in the base article, we follow a similar setup. If the number of nodes $n \geq 10^5$, then $d_1 = \sqrt[3]{n}$, else $d_1 = \sqrt{n}$. Also, $d_2 = \frac{d_1}{2}$.
To implement GRL_{QUBO}, we took inspiration from the study presented in the paper of GRL. Here, we had three layers of GAT as the encoding architecture and a sigmoid layer to generate initial node probabilities. To set the dimension of the node feature vectors,

TABLE 3. The number of satisfied constraints (Reward in Grey Columns) and training time in seconds for the Max-Cut problem for the graphs of different densities. (*) along with bold denotes the best value observed. The number of satisfied constraints denotes, how many edges (from E) have satisfied the constraint of Max-Cut.

n	E	PI-GNN	GRL $QUBO$	MCTS -GNN	GRL $QUBO$ vs PI-GNN	MCTS -GNN vs PI-GNN	Train (Seconds) PI-GNN	Train (Seconds) GRL_{QUBO}	Train (Seconds) MCTS-GNN
50	89	72 (67.3 ± 4.5)	75 (71.1 ± 3.6)	**76*** (74.4 ± 1.5)	4.2	5.56	698	211	1066
	139	95 (73.3 ± 22.2)	105 (96.6 ± 8.76)	**107*** (104.3 ± 2.6)	10.53	12.63	691	184	835
	499	276 (241.3 ± 37.5)	301 (290.7 ± 10.13)	**314*** (310.9 ± 3.1)	10.14	13.78	3074	439	1834
100	199	147 (69.5± 70.3)	155 (148.1± 6.5)	**167*** (165.3± 1.5)	5.44	13.61	317	347	5147
	799	373 (93.3±186.5)	534 (520 ± 13.5)	**537*** (532.5± 5.9)	43.2	44	2668	602	8479
300	399	342 (78.5 ± 29.7)	360 (347.3 ± 12. 4)	**365*** (696.1 ± 1.1)	5.26	6.73	398	427	1027
	899	13 (555. 4 ± 60.4)	690 (672. 3 ± 18.9)	**699*** (696.1 ± 3.3)	12.56	14.03	667	862	5789
	1299	747 (651±99.2)	932 (897.5 ± 31.5)	**947*** (942.1 ± 6.8)	24.77	26.77	1837	1789	9035
500	799	622 (743.3 ± 78.9)	672 (640.5 ± 33.2)	**694*** (693.1 ± 1.2)	8.04	11.58	478	679	1507
	1499	1007 (± 102.3)	1143 (1093.5 ± 51.1)	**1158*** (1150.4 ± 7.6)	13.51	15	1478	1025	6478
	5499	2689 (2511. 3 ± 155.6)	3414 (3344.7 ± 78.6)	**3535*** (3521.3 ± 14.2)	26.96	31.46	2478	2247	8798
700	1199	938 (891.8 ± 45.3)	979 (961.6 ± 15.5)	**1023*** (1018.2 ± 4.7)	4.37	9.06	932	725	2017
	1699	1288 (1191.2 ± 97.2)	1308 (1277.4 ± 30.3)	**1360*** (1354.1 ± 6.3)	1.55	5.59	1027	879	4789
	4699	2422 (2288.3± 132.6)	2992 (2930± 50.5)	**3202*** (3192.2± 10.5)	23.53	32.2	2104	2578	9786
1000	1299	1104 (1095.2±80.6)	1112 (1099±10.6)	**1141*** (1137±3.5)	0.73	3.35	1265	1681	10510
	3299	2204 (2110.6±97.3)	2449 (2335.3±89.4)	**2525*** (2518± 7.6)	0.69	14.56	2333	1414	15146
	5299	3098 (2993 ± 91.8)	3716 (3589 ± 129.2)	**3750*** (3733 ± 17)	19.95	21.05	2017	2768	22147
3000	3499	2956 (2916.7 ± 40.1)	**3218*** (3198.4 ±19.7)	2996 (2992.3 ± 4.3)	8.86	1.35	1879	2147	8978
	4499	3627 (3585.2± 86.9)	**3907*** (3885.6 ± 45.2)	3885 (3870.3 ± 13.7)	7.72	7.11	2998	2378	13.147
	6999	4841 (4440.4 ± 137.9)	5550 (5473.1 ± 76.1)	**5622*** (5993 ± 25.4)	14.65	16.13	5014	4789	24789

we follow a similar kind of strategy. So, if $n \geq 10^5$ then, $d_1 = \sqrt[3]{n}$ else, $d_1 = \sqrt{n}$ and followup $d_2 = \lceil \frac{d_1}{2} \rceil$. We have used a single attention-head mechanism.

In MCTS-GNN, we train a single GNN in different rollout phases by applying different manual labeling perturbations. The setup is also completely similar. If $n \geq 10^5$, then $d_1 = \sqrt[3]{n}$ else, $d_1 = \sqrt{n}$. $d_2 = \lceil \frac{d_1}{2} \rceil$ and $d_3 = 1$.

2) **Training Description:** To train the architectures, we use Adam optimizer for each of the architectures. We use a patience value to implement fuzzy early stopping for all of them. As RL-based setups inherently exhibit abrupt behavior in reward variation, we keep a comparatively higher patience value for GRL_{QUBO} and MCTS-GNN compared to PI-GNN. As per our setups, we eventually observed almost linear variation in terms of objective functions' values at some phase of the training epochs for all the experimented architectures for different graph inputs. This supports the validity of the chosen hyperparameters for early stopping. There lies another reason behind this strategy. In the PI-GNN model, we monitor fractional variations in the loss values, contrasting with other experimented architectures where the reward values are integer. Inherently, it can be understood that the integer rewards should fluctuate more compared to the fractional loss values. This demands a higher early stopping threshold for the RL-based setups.

For PI-GNN, we choose learning rate $lr = 10^{-4}$ with a patience value of $p = 100$ as per the discussed fuzzy early stopping strategy. For GRL_{QUBO}, we choose lr to 0.001 for both encoder-decoder and the patience value p to 700. We also experimented with a lesser learning rate (0.0001) for GRL_{QUBO} but could not observe further improvement in terms of satisfying the number of constraints. For MCTS-GNN, we also chose

lr to 0.001 for the GNN architecture with the p to 100. Apart from the prior hyperparameters to control GNN training, we included another early stopping criterion p' that tracked the improvement in the reward objective. We set p' to 700, denoting that if there is no reward improvement compared to the best reward observed till the current iteration, the MCTS-GNN algorithm terminates.

B. NUMBER OF SATISFIED CONSTRAINTS

This section presents the number of satisfied constraints observed in each architecture for the graphs of different intensities. The complete result is shown in Table 3. In this study, this metric is denoted as the reward in terms of RL-based formulation. For GRL_{QUBO} and MCTS-GNN, we present the best reward observed throughout the complete training. For PI-GNN, we record the least loss and the corresponding probability distribution to approximate the node labels based on the threshold β (set to 0.5). These approximated node labels are used to calculate the QUBO-formulated Hamiltonian function. In the 6^{th} and 7^{th} columns of Table 3, we present how GRL_{QUBO} and MCTS-GNN's performance has improved compared to PI-GNN in percentage. A positive value means the result has improved, and the negative means the opposite. To generate the results, for each graph, we ran around 5 times and reported the one that achieved the highest number of satisfied constraints. Alongside, we also present the average number of satisfied constraints and the corresponding standard deviation observed across multiple iterations in 2^{nd} to 4^{th} columns of Table 3.

From the reported result in Table 3 in the 9^{th} and 10^{th} columns, we can see that the performance has comparatively improved than the base PI-GNN. In simple terms, upon applying RL-based formulation, we improved the quality of the resultant output. It can be seen from Table 3 through the relevant columns that, for a particular graph having n nodes, the performance generally improves more with it being denser.

As per our analysis, PI-GNN is quite scalable and a generic solution to address a wide range of CO problems. However, it lags behind RL-based methods in meeting constraints, especially as the graphs get denser. In GRL_{QUBO}, when the attention-based decoding scheme selects a particular node and sets its label, only its adjacent unlabeled neighbor nodes' attention weights (α) are updated. Then, using a max heap-based strategy, the next likely node is selected and labeled. As the binary vector is updated in each selection, the performance might be improved if all the nodes' attention weights are updated. However, it will add a significant runtime cost due to linear updates in each selection, so a runtime-accuracy tradeoff exists there. MCTS-GNN, intuitively applies noise during training to enforce the training to skip various local minima to reach comparatively better outcomes.

We also discuss terms of result sparsity. As we can see from Table 3, alongside the best results obtained in multiple iterations, we also report the average and standard deviation found from the trials. The data indicates that, for each algorithm, the standard deviation increases with a fixed node number n as the number of edges (or constraints) rises. Among the algorithms, MCTS-GNN maintains the most stable standard deviation, while and GNN exhibit higher variability. Particularly, as edge density increases, the performance of GNN degrades more significantly compared to GRL_{QUBO}, a pattern we have previously discussed in earlier sections.

C. TRAINING STABILITY

Now, we highlight the convergence status or training stability of the experimented architectures based on our set hyperparameters. We centralize our discussion based on two graphs of 50 nodes with 89 (50, 89) and 499 (50, 499) edges, respectively, to understand the behavior transition as the density increases. In Fig. 3(a) and (b), we present the reward variations for the graphs (50, 89) and (50, 499), respectively. Also, to understand the loss convergence status of PI-GNN, we present Fig. 3 (c) for the two graphs (50, 89) and (50, 499).

The observation that can be drawn from each is that the convergence behavior or almost linear variation is quite visible where our training has stopped for all the architectures. This supports the stability quality of the set hyperparameters, especially the patience values. Also, we do not provide all the other graphs' training convergence chart in this study, because they exhibit a similar pattern.

D. COMPLEXITY ANALYSIS

In this section, we rigorously discuss the computation-intensive segments of all the architectures.

PI-GNN, GRL, and MCTS-GNN, all the architectures use a GNN or GNN-alike architecture. To process a graph having n nodes to generate a d dimensional node feature vectors through L layers of GNN, the worst case complexity is $\mathcal{O}(Lnd^2)$[1]. GAT used by GRL takes some additional runtime costs [3] for its calculation. GRL also applies a decoding mechanism incurring additional $\mathcal{O}(n \log n)$ complexity. MCTS-GNN trains a single GNN repeatedly based on the perturbations where only a few times exhaustive training occurs and the remaining times it runs for quite a smaller number of epochs - less than 1000 found in our experiments. In summary, PI-GNN is the most scalable. The complexity of GRL_{QUBO} increases with the graphs becoming larger and MCTS-GNN increases with the number of iterations to run to expand the search tree. However, with this additional cost, we improve the number of satisfied constraints.

Apart from these theoretical aspects, other hyperparameters and input patterns control the convergence of loss functions. Based on our observations, we generally found MCTS-GNN to take the most time due to the expansion of the search tree. GRL_{QUBO} takes way less time than MCTS-GNN

[1]Here we assume that, the graph is sparse so, the number of edges $E = \mathcal{O}(n)$.

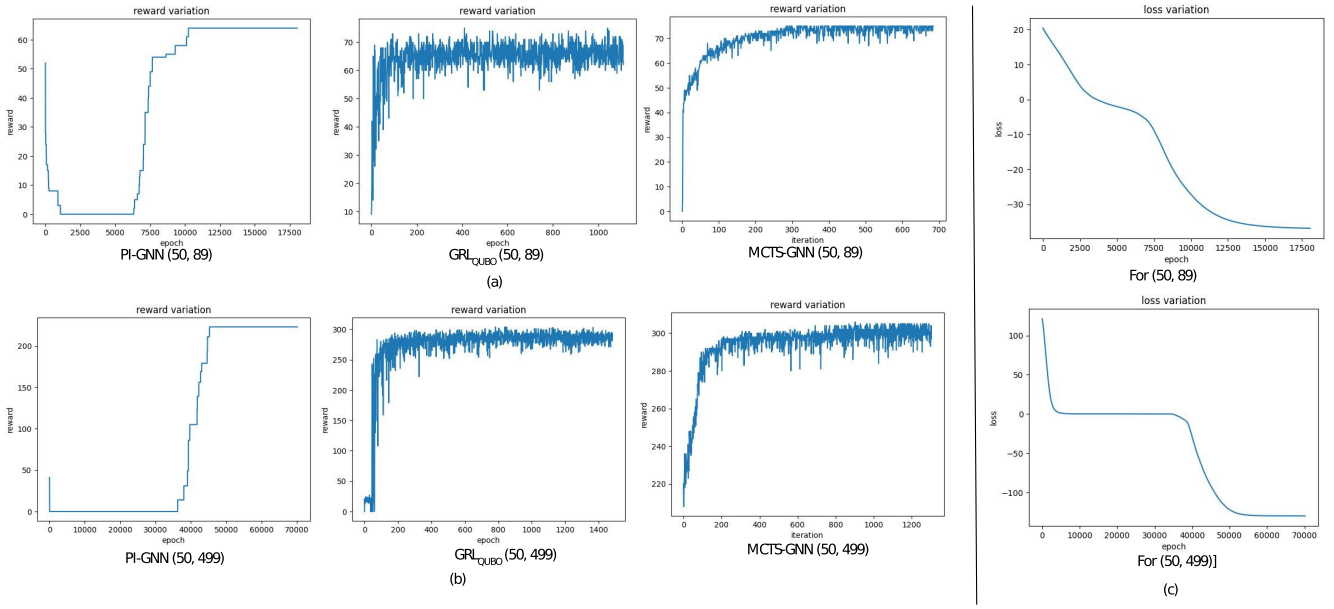

FIGURE 3. Reward variation curves for a) (50, 89) and b) (50, 499), respectively. c) Loss variation curves for PI-GNN.

and provides a competitive performance against PI-GNN - where PI-GNN takes a longer number of epochs to be trained. GRL_{QUBO} takes fewer epochs than PI-GNN where greedy selection takes additional processing in each step. PI-GNN shows worse performance than GRL_{QUBO}, especially in denser graphs where it might need a good amount of time to be converged. In Table 3 in columns 11^{th} to 13^{th}, we present some results; each value is denoted in seconds.

V. CONCLUSION AND FUTURE WORK

In this study, we improve the performance of PI-GNN in terms of the number of satisfied constraints. We have incorporated a fuzzy early-stopping criterion to improve PI-GNN's loss improvement issue in denser graphs, and designed strategies that incorporate node projection strategies directly during training. We have also formulated a generic loss function in the form of QUBO-formulated Hamiltonian to create a bridge between RL formulation and QUBO models. Additionally, we have proposed two architectures GRL_{QUBO} and MCTS-GNN, where for each we have discussed how QUBO-Hamiltonian-based reward can be integrated into the loss function to conduct the training and produce solutions in an unsupervised setup. We have also discussed the improved decoding strategy of GRL_{QUBO} and guided search of MCTS-GNN in a detailed manner. As a benchmarking problem, we have considered the Max-Cut problem and achieved up to 44% improvement in terms of the number of satisfied constraints compared to PI-GNN. The primary novelty of this work is the formulation of a graph-based CO-problem using QUBO-Hamiltonian, then using the proposed setup, an RL-based solution can be applied, and as empirically stated, the quality of the solution in terms of number of satisfied constraints can be improved. Another important motivation behind integrating with QUBO formulation was its generality and extendability to PUBO (Polynomial Unconstrained Binary Optimization), which we plan to investigate in the later phase of this work for a wide range of graph-based CO problems along with integrating other sophisticated strategies relevant to RL-based framework. Furthermore, an exciting follow-up extension of our work also includes investigating the applicability of our solutions in quantum computer simulations in terms of the quality of the output, convergence status, and associated tradeoffs.

ACKNOWLEDGMENT

We acknowledge the support provided by the University of Dhaka, which was essential in facilitating this work. The authors sincerely thank all the anonymous reviewers for their time and wonderful suggestions and comments to improve the quality of their article. They believe these points have significantly improved the quality and clarity of the current version of their manuscript.

REFERENCES

[1] A. Verma, M. Lewis, and G. Kochenberger, "Efficient QUBO transformation for higher degree pseudo Boolean functions," 2021, *arXiv:2107.11695*.

[2] M. Zaman, K. Tanahashi, and S. Tanaka, "PyQUBO: Python library for mapping combinatorial optimization problems to QUBO form," *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 838–850, Apr. 2022.

[3] M. J. A. Schuetz, J. K. Brubaker, and H. G. Katzgraber, "Combinatorial optimization with physics-inspired graph neural networks," *Nature Mach. Intell.*, vol. 4, no. 4, pp. 367–377, Apr. 2022.

[4] T. Gabor, M. Lingsch Rosenfeld, C. Linnhoff-Popien, and S. Feld, "How to approximate any objective function via quadratic unconstrained binary optimization," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Mar. 2022, pp. 1249–1257.

[5] M. Diaby and M. H. Karwan, *Advances in Combinatorial Optimization: Linear Programming Formulations of the Traveling Salesman and Other Hard Combinatorial Optimization Problems*. Singapore: World Scientific, 2016.

[6] S. Speziali, F. Bianchi, A. Marini, L. Menculini, M. Proietti, L. F. Termite, A. Garinei, M. Marconi, and A. Delogu, "Solving sensor placement problems in real water distribution networks using adiabatic quantum computation," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*, Oct. 2021, pp. 463–464.

[7] M. J. A. Schuetz, J. K. Brubaker, and H. G. Katzgraber, "Reply to: Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set," *Nature Mach. Intell.*, vol. 5, no. 1, pp. 32–34, Dec. 2022.

[8] X. Li, L. Sun, M. Ling, and Y. Peng, "A survey of graph neural network based recommendation in social networks," *Neurocomputing*, vol. 549, Sep. 2023, Art. no. 126441.

[9] H. Xu, G. Wu, E. Zhai, X. Jin, and L. Tu, "Preference-aware light graph convolution network for social recommendation," *Electronics*, vol. 12, no. 11, p. 2397, May 2023.

[10] S. Gao, X. Xing, H. Wang, M. Xin, and Z. Jia, "SRUH-GNN: Social recommendation of user homophily based on graph neural network," in *Proc. IEEE 12th Data Driven Control Learn. Syst. Conf. (DDCLS)*, May 2023, pp. 1455–1460.

[11] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[12] S. Boettcher, "Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems," *Nature Mach. Intell.*, vol. 5, no. 1, pp. 24–25, Dec. 2022.

[13] M. C. Angelini and F. Ricci-Tersenghi, "Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set," *Nature Mach. Intell.*, vol. 5, no. 1, pp. 29–31, Dec. 2022.

[14] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," 2021, *arXiv:2103.16378*.

[15] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*.

[16] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2018, *arXiv:1803.08475*.

[17] I. Drori, A. Kharkar, W. R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D. P. Williamson, and M. Udell, "Learning to solve combinatorial optimization problems on real-world graphs in linear time," in *Proc. 19th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2020, pp. 19–24.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[19] M. Fey and J. Eric Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*.

[20] V. Jayasiri, "NW: Labml.AI: A library to organize machine learning experiments," 2020.

[21] M. Ghaffari, "An improved distributed algorithm for maximal independent set," in *Proc. 27th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2016, pp. 270–277.

[22] G. Karakostas, "A better approximation ratio for the vertex cover problem," *ACM Trans. Algorithms*, vol. 5, no. 4, pp. 1–8, Oct. 2009.

[23] Z. Sun and Y. Yang, "DIFUSCO: Graph-based diffusion solvers for combinatorial optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2023, pp. 3706–3731.

[24] Z. Meng, Q. Qian, M. Xu, B. Yu, A. R. Yildiz, and S. Mirjalili, "PINN-FORM: A new physics-informed neural network for reliability analysis with partial differential equation," *Comput. Methods Appl. Mech. Eng.*, vol. 414, Sep. 2023, Art. no. 116172.

[25] S. M. Sait, P. Mehta, N. Pholdee, B. S. Yildiz, and A. R. Yildiz, "Artificial neural network infused quasi oppositional learning partial reinforcement algorithm for structural design optimization of vehicle suspension components," *Mater. Test.*, vol. 66, no. 11, pp. 1855–1863, Nov. 2024.

[26] P. Mehta, S. M. Sait, B. S. Yildiz, M. U. Erdaş, M. Kopar, and A. R. Yildiz, "A new enhanced mountain gazelle optimizer and artificial neural network for global optimization of mechanical design problems," *Mater. Test.*, vol. 66, no. 4, pp. 544–552, Apr. 2024.

[27] Z. C. Dou, S. C. Chu, Z. Zhuang, A. R. Yildiz, and J. S. Pan, "GBRUN: A gradient search-based binary Runge Kutta optimizer for feature selection," *J. Internet Technol.*, vol. 25, no. 3, pp. 341–353, 2024.

[28] S. Rainjonneau, I. Tokarev, S. Iudin, S. Rayaprolu, K. Pinto, D. Lemtiuzhnikova, M. Koblan, E. Barashov, M. Kordzanganeh, M. Pflitsch, and A. Melnikov, "Quantum algorithms applied to satellite mission planning for Earth observation," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 16, pp. 7062–7075, 2023.

[29] S. Jerbi, L. M. Trenkwalder, H. Poulsen Nautrup, H. J. Briegel, and V. Dunjko, "Quantum enhancements for deep reinforcement learning in large spaces," *PRX Quantum*, vol. 2, no. 1, Feb. 2021, Art. no. 010328.

[30] J. Lin, Z. Y. Lai, and X. Li, "Quantum adiabatic algorithm design using reinforcement learning," *Phys. Rev. A, Gen. Phys.*, vol. 101, no. 5, May 2020, Art. no. 052327.

[31] A. T. D. Perera and P. Kamalaruban, "Applications of reinforcement learning in energy systems," *Renew. Sustain. Energy Rev.*, vol. 137, Mar. 2021, Art. no. 110618.

REDWAN AHMED RIZVEE received the B.Sc. and M.S. degrees in computer science from the Department of Computer Science and Engineering, University of Dhaka, Bangladesh, in 2018 and 2021, respectively. Currently, he is a Lecturer with the Department of Computer Science and Engineering, University of Dhaka. Before joining the University of Dhaka, he was a Research Scientist with TigerIT Bangladesh Ltd., from 2021 to 2022, and a Lecturer with the Department of Computer Science and Engineering, East West University, Bangladesh, from 2022 to 2023. His research interests include data mining, theoretical deep learning, high-performance computing, and programming systems.

RAHEEB HASSAN received the B.Sc. degree (Hons.) in computer science and engineering from the University of Dhaka, Bangladesh, in 2023. During his undergraduate studies, he conducted research as a Research Assistant with the Cognitive Agents and Interaction Laboratory (CAIL), University of Dhaka, where he is now a Research Fellow (Adjunct). He is currently a Machine Learning Engineer at Therap Services LLC, working on computer vision systems to assist individuals with intellectual and developmental disabilities. His research interests include reinforcement learning, distributed computing, multi-agent systems, and the application of computer vision and multi-modal systems.

MD. MOSADDEK KHAN received the B.Sc. and M.Sc. degrees in computer science and engineering from the University of Dhaka, Bangladesh, in 2010 and 2012, respectively, and the Ph.D. degree from the University of Southampton, U.K., in 2018. He was a Lecturer with the State University of Bangladesh and the Ahsanullah University of Science and Technology, from 2011 to 2013. Then, he returned to the University of Dhaka as a Faculty Member. He is currently an Associate Professor with the University of Dhaka, where he leads his research group. His research interests include distributed problem-solving, multi-agent systems, and multi-agent reinforcement learning.