nature portfolio

https://doi.org/10.1038/s42256-022-00468-6

# Supplementary information

# Combinatorial optimization with physics-inspired graph neural networks

In the format provided by the authors and unedited

# Supplemental Material for:
# Combinatorial Optimization with Physics-Inspired Graph Neural Networks

Martin J. A. Schuetz,[1,2,3] J. Kyle Brubaker,[2] and Helmut G. Katzgraber[1,2,3]

[1] *Amazon Quantum Solutions Lab, Seattle, Washington 98170, USA*
[2] *AWS Intelligent and Advanced Compute Technologies, Professional Services, Seattle, Washington 98170, USA*
[3] *AWS Center for Quantum Computing, Pasadena, CA 91125, USA*

## I. CORE GCN CODE BLOCK

Listing 1. Core code block of example script based on the DGL library. The first block defines a two-layer GCN architecture Ansatz; the second code block defines the loss function as described by Eq. (6). Further details can be found in the main text, as well as in Ref. [S117].

```
# Import required packages
import dgl
import torch
import torch.nn as nn
from dgl.nn.pytorch import GraphConv

# Define two-layer GCN
class GCN(nn.Module):
  def __init__(self,in_feats,hidden,classes):
      super(GCN, self).__init__()
      self.conv1 = GraphConv(in_feats,hidden)
      self.conv2 = GraphConv(hidden,classes)

  def forward(self,g,inputs):
      h = self.conv1(g,inputs)
      h = torch.relu(h)
      h = self.conv2(g,h)
      # binary classification
      h = torch.sigmoid(h)
      return h

# Define custom loss function for QUBOs
def loss_func(probs_,Q_mat):
  """
  function to compute cost value for given
  soft assignments and predefined QUBO matrix
  """
  # minimize cost = x.T * Q * x
  cost = (probs_.T @ Q_mat @ probs_).squeeze()

  return cost
```

## II. HYPERPARAMETERS FOR G-SET EXPERIMENTS

In this section, we provide details for the specific model configurations (hyperparameters) as used to solve the Gset instances with our physics-inspired GNN solver (PI-GNN). The results achieved with these model configurations are displayed in Tab. I; the corresponding hyperparameters are given in Tab. II. Our base GCN architecture with tunable number of layers $K$ is specified in Listing 2.

| graph | PI-GNN | embedding $d_0$ | layers $K$ | hidden dim $d_1$ | hidden dim $d_2$ | hidden dim $d_3$ | learning rate $\beta$ | dropout |
|---|---|---|---|---|---|---|---|---|
| G14 | 3026 | 369 | 1 | 5 | — | — | 0.00467 | 0.0 |
| G15 | 2990 | 394 | 1 | 5 | — | — | 0.00587 | 0.0 |
| G22 | 13181 | 419 | 2 | 1909 | 3401 | — | 0.00103 | 0.4498 |
| G49 | 5918 | 2167 | 3 | 2338 | 1955 | 8 | 0.00058 | 0.3554 |
| G50 | 5820 | 208 | 3 | 218 | 3582 | 566 | 0.00488 | 0.2365 |
| G55 | 10138 | 278 | 3 | 8412 | 8352 | 5499 | 0.00161 | 0.1062 |
| G70 | 9421 | 109 | 3 | 1233 | 7048 | 11869 | 0.00139 | 0.3912 |

TABLE II. Numerical results for MaxCut on Gset instances, with hyperparameters specified for the PI-GNN solver.

Listing 2. Base GCN architecture used for solving MaxCut on Gset problem instances.

```
# Define GNN object
class GCN_dev(nn.Module):
    def __init__(self, in_feats, hidden_sizes, dropout, num_classes):
        super(GCN_dev, self).__init__()
        # Combine all layer sizes into a single list
        all_layers = [in_feats] + hidden_sizes + [num_classes]
        # slice list into sub-lists of length 2
        self.layer_sizes = list(window(all_layers))
        # reference to ID final layer
        self.out_layer_id = len(self.layer_sizes) - 1
        self.dropout_frac = dropout
        self.layers = OrderedDict()
        for idx, (layer_in, layer_out) in enumerate(self.layer_sizes):
            self.layers[idx] = GraphConv(layer_in, layer_out).to(DEVICE)
    def forward(self, g, inputs):
        for k, layer in self.layers.items():
            if k == 0: # reference to ID final layer
                h = layer(g, inputs)
                h = torch.relu(h)
                h = F.dropout(h, p=self.dropout_frac)
            elif 0 < k < self.out_layer_id: # intermediate layers
                h = layer(g, h)
                h = torch.relu(h)
                h = F.dropout(h, p=self.dropout_frac)
            else: # output layer
                h = layer(g, h)
                h = torch.sigmoid(h)  # binary classification
        return h
```