# VN-Solver: Node Embedding using VGAE and PointNet for Classification to solve Combinatorial optimization problems over graphs

Dhana Lakshmi Kankanala

## ABSTRACT

In our research, we investigate an innovative approach to resolving complex combinatorial optimization problems on graphs, specifically focusing on the differentiation between Hamiltonian and non-Hamiltonian graphs. Traditional methodologies predominantly employ adjacency matrices as the primary input for neural combinatorial solvers, leveraging discernible data patterns to navigate the computational complexity inherent to these problems. Contrary to conventional practices, our study introduces a groundbreaking vision-based strategy that converts adjacency matrices into three-dimensional point clouds, facilitating the classification of these problems within a 3D space.We convert the given adjacency graph data into 3D point clouds and train the PointNet for binary classification of graphs.Our results suggest that adding more additional features and providing the 3D coordinates of points in point cloud, the PointNet architecture can be used for the graph classification tasks.

## KEYWORDS

VN-Solver, Neural Combinatorial Optimization, Pointcloud, VGAE (Variational Graph Auto encoder)

**ACM Reference Format:**

Dhana Lakshmi Kankanala. 2025. VN-Solver: Node Embedding using VGAE and PointNet for Classification to solve Combinatorial optimization problems over graphs. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Combinatorial optimization problems on graphs are crucial across various domains.The combinatorial optimization problems over graphs, such as the traveling salesman problem, pose significant challenges due to their complexity and the computational resources required for their resolution. Traditional methods rely on adjacency matrices to represent these problems, which, while effective for small-scale instances, rapidly become computationally impractical as the size and complexity of the problem increase. This limitation underscores the need for more efficient and scalable solutions.

The introduction of the VN-Solver framework marked a significant advancement in this domain by employing a 2D vision-based approach. By transforming graph data into visual representations, this method leverages the power of convolutional neural networks to interpret and classify graph structures, offering a notable improvement in computational efficiency and accuracy. However, this approach is not without its limitations, particularly in capturing the multi-dimensional complexity of graph data, which can lead to oversimplification and potential inaccuracies in classification tasks.

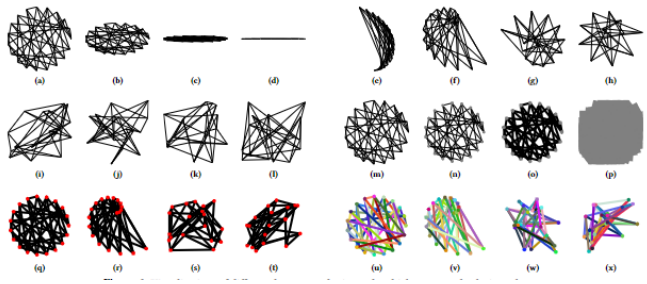

**Figure 1: Images of the graphs in 2D VN-solver with different visualization, node size and coloring schemes**

Addressing these limitations, our extension of the VN-Solver framework through the integration of 3D point clouds represents a pivotal development. The use of 3D point clouds for representing graph data preserves the spatial relationships and enhances feature representation, providing a more accurate and intuitive understanding of complex graph structures. This approach not only addresses the shortcomings of the 2D vision-based method by offering a more detailed representation but also sets a new benchmark in the classification of Hamiltonian and non-Hamiltonian problems, demonstrating the potential of 3D spatial representations in advancing the field of combinatorial optimization.

## 2 VN-SOLVER METHODS

In this paper, our approach to solving combinatorial optimization problems on graphs involves transforming adjacency matrices into 3D point clouds to better capture the spatial relationships and complexities of graph structures. Initially, we generate node embeddings using VGAE model that encapsulate key graph metrics such as connectivity and node centrality, forming a feature matrix that represents each node's vector. These embeddings are then converted into 3D point clouds by generating the edge points extracting edge relationships from the graph data, creating a visual and spatial representation of the graph. Finally, we employ PointNet to classify these point clouds into Hamiltonian and non-Hamiltonian graphs, leveraging the detailed spatial features and relationships captured in the 3D space. This methodology offers a novel way to understand and solve complex graph problems by combining advanced neural



*Conference'17, July 2017, Washington, DC, USA*



ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

network architectures with 3D spatial data processing.We explain each step briefly below

## 2.1 Graph Embedding

The feature matrix includes metrics like degree centrality, clustering coefficient, graph density, number of connected components, betweenness centrality, eigenvector centrality, and PageRank. These features collectively capture both the local and global structural properties of nodes within a graph.This comprehensive feature matrix is crucial for learning node embeddings, capturing both local and global graph characteristics essential for tasks like graph classification or visualization.

The modified Variational Graph Auto-Encoder (VGAE) model enhances point cloud classification by incorporating initial node embeddings from various layouts (random, circular, spiral) with the encoder's output before the decoder's input. This strategic concatenation leverages spatial structures of initial layouts to ensure generated embeddings maintain essential structural and spatial relationships, thereby producing more meaningful, project-specific node embeddings. The GNN Encoder and VGAE Decoder classes, pivotal in transforming graph data to a latent space and reconstructing node features or connections, ensure the model captures both local and global graph properties effectively.

During training, the model learns to encode graph features and topology into latent representations and then decodes these to reconstruct adjacency matrices and 3d coordinates of nodes.The loss function for the VGAE model combines binary cross-entropy (BCE) and Kullback-Leibler (KL) divergence. BCE measures the difference between the predicted adjacency matrix and the true adjacency matrix, ensuring accurate graph structure reconstruction. KL divergence penalizes the deviation of the latent variable distributions from a standard normal distribution, encouraging efficient and meaningful latent space organization. Together, these components balance reconstruction accuracy and latent space regularization, crucial for effective graph embedding learning.

## 2.2 Point cloud Generation

After generating graph embeddings using a Variational Graph Auto-Encoder (VGAE), the resulting node embeddings can be used to produce a point cloud representation of the graph. To create a 3D representation of a graph's structure, we calculate points along the connections between nodes, forming edge points. These points, combined with node positions, compile into a point cloud that visually represents the graph's topology and connections. This method ensures the 3D model accurately reflects the nodes and their interrelations, offering a detailed view of the graph. Using PyntCloud, we construct this 3D point cloud, incorporating specific features for each point, and save the model in the PLY format for comprehensive structural visualization.

## 2.3 Point Cloud Data Processing for Graph Classification

The goal is to organize and process point cloud data obtained from 3D graph representations, specifically distinguishing between non-Hamiltonian and Hamiltonian graphs. Define the directories for point clouds corresponding to non-Hamiltonian and Hamiltonian

**Figure 2: Hamiltonian graphs**

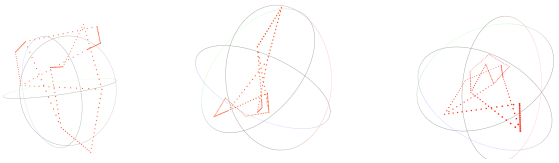

**Figure 3: Non-Hamiltonian graphs**

graphs, incorporating the chosen visualization and model size parameters. The random seed is set for reproducibility. Shuffle file names of all point clouds. For each point cloud file, determine its classification label based on the filename. The point cloud data is read and processed using the PyntCloud library. The extracted coordinates and associated class labels are compiled into a list.This dataset is used for training and evaluating PointNet, to classify between Hamiltonian and non-Hamiltonian graphs.

## 2.4 Point Cloud features

The generated 3D point cloud encapsulates a diverse set of features for each node, offering a detailed representation of the underlying graph. The spatial coordinates (x, y, z) of each node in the 3D space are determined based on the chosen visualization method. Nodes are distinguished from edge points through a binary node type parameter. The node index, source, and target nodes provide essential information about the node's position in the original graph and its connectivity. Additional features such as the number of targets, edge length, and various graph features (degree, degree centrality, connectivity, local features, and density) enrich the point cloud with insights into node centrality, connectivity patterns, and local graph characteristics. These features collectively contribute to a comprehensive portrayal of the graph's structural attributes, facilitating in-depth analysis and interpretation of the 3D point cloud.

## 2.5 Classification Model

PointNet is a neural network architecture designed for processing unordered point cloud data, and it can be adapted for graph-based tasks.The input to the pointNet should be the 3D coordinates of nodes and any additional features. While pointNet is proficient in distinguishing between distinct shapes and patterns in point clouds, the ability of PointNet to classify graphs based solely on 3D coordinates without additional features that signify the graph's connectivity or other attributes might be limited. For graph classification, features that capture the relationships and structural properties of the graph are crucial, which might not be fully represented through 3D coordinates alone.By adding additional features to the point.With these additional features, the classification model can better understand the underlying patterns and relationships in

the data, leading to improved accuracy and the ability to distinguish between more subtle differences in the point cloud structures.The PointNetCls architecture consists of three main components:

**Point Classification Network**: Additional convolutional and fully connected layers (conv1, conv2, conv3, fc1, fc2, fc3) for point classification.ReLU activation function (relu) is applied between layers.

## 2.6 Input Transformation (TNet)

(conv1) : Conv1d(15, 64, kernelsize = (1, ), stride = (1, ))
(bn1) : $BatchNorm1d(64, momentum = 0.1, affine = True)$
(conv2) : Conv1d(64, 128, kernelsize = (1, ), stride = (1, ))
(bn2) : $BatchNorm1d(128, momentum = 0.1, affine = True)$
(conv3) : Conv1d(128, 1024, kernelsize = (1, ), stride = (1, ))
(bn3) : $BatchNorm1d(1024, momentum = 0.1, affine = True)$
(fc1) : Linear(in_features = 1024, out_features = 512, bias = $True$)
(fc2) : Linear(in_features = 512, out_features = 256, bias = $True$)
(fc3) : Linear(in_features = 256, out_features = 225, bias = $True$)

## 2.7 Feature Transformation (TNet)

(conv1) : Conv1d(15, 64, kernelsize = (1, ), stride = (1, ))
(bn1) : $BatchNorm1d(64, momentum = 0.1, affine = True)$
(conv2) : Conv1d(64, 128, kernelsize = (1, ), stride = (1, ))
(bn2) : $BatchNorm1d(128, momentum = 0.1, affine = True)$
(conv3) : Conv1d(128, 1024, kernelsize = (1, ), stride = (1, ))
(bn3) : $BatchNorm1d(1024, momentum = 0.1, affine = True)$
(fc1) : Linear(in_features = 1024, out_features = 512, bias = $True$)
(fc2) : Linear(in_features = 512, out_features = 256, bias = $True$)
(fc3) : Linear(in_features = 256, out_features = 225, bias = $True$)

## 2.8 PointNet Classification Head

(conv1) : Conv1d(15, 64, kernelsize = (1, ), stride = (1, ))
(bn1) : $BatchNorm1d(64, momentum = 0.1, affine = True)$
(conv2) : Conv1d(64, 128, kernelsize = (1, ), stride = (1, ))
(bn2) : $BatchNorm1d(128, momentum = 0.1, affine = True)$
(conv3) : Conv1d(128, 1024, kernelsize = (1, ), stride = (1, ))
(bn3) : $BatchNorm1d(1024, momentum = 0.1, affine = True)$
(fc1) : Linear(in_features = 1024, out_features = 512, bias = $True$)
(fc2) : Linear(in_features = 512, out_features = 256, bias = $True$)
(fc3) : Linear(in_features = 256, out_features = 2, bias = $True$)
(relu) : ReLU()

# 3 DATASET AND TRAINING

The dataset discussed in the paper involves solving the Hamiltonian cycle problem using a vision-based neural solver, VN-Solver, focusing on two datasets: Small and Large. The Small dataset contains 4,115 graphs with 4 to 20 nodes, including 2,277 Hamiltonian and 1,838 non-Hamiltonian graphs. The Large dataset includes 13,192 graphs with 20 to 50 nodes, comprising 7,453 Hamiltonian and 5,739 non-Hamiltonian graphs. These datasets, sourced from the House of Graphs, enable the exploration of VN-Solver's effectiveness in classifying graphs based on visual representations rather than traditional matrix-based approaches, demonstrating the potential of vision-based methods in combinatorial optimization problems over graphs.

The number of points in the point cloud are calculated dynamically, representing the desired number of points in each point cloud batch.This process involves iterating through the dataset to find the average number of points per point cloud, establishing a uniform point count across batches.This average is then used as the total points in point cloud for the data loader. The goal is to ensure a consistent size for the point clouds in each batch, addressing the challenge of varying point cloud densities.

To process batches of point clouds for deep learning models, a general approach involves ensuring uniformity in the number of points per point cloud. This method typically includes padding shorter point clouds with zeros to match the specified point count, ensuring that all inputs to the model have consistent dimensions. This preprocessing step is crucial for batch processing in neural networks, allowing for efficient and effective training. The processed point clouds, alongside their corresponding labels, are then converted into tensors suitable for model input, facilitating the learning process in tasks such as classification or segmentation.

We further creates instances of a custom dataset class, PointCloudDataset, for the training, validation, and test sets. Finally, DataLoader objects are instantiated using the custom function, specifying the desired batch size and shuffle behavior.

This configuration is specifically designed to handle 3D point cloud data and get it prepared to be fed into a PointNet. To handle the different sizes of point clouds in a batch and provide a constant input shape for model training, a custom collate function is required. The PointNet model's requirements and the available computational resources can be used to modify the batch size and the number of points per point cloud.

# 4 RESULTS AND EVALUATION

Define a cross-entropy loss criterion and an Adagrad optimizer with a learning rate scheduler of 0.001. The training loop runs for a specified number of epochs, involving the forward pass, backpropagation, and optimization steps. Performance metrics such as accuracy, area under the ROC curve (AUC), F1 score, and recall are calculated and stored at each epoch. An early stopping mechanism is implemented to prevent overfitting. The entire process is repeated for a total number of epochs. the stepLR scheduler with gamma value of 0.8 is used.

Please check the below results table for binary classification using PointNet: The last row of the results are using VGAE model for node embeddings.

**Table 1: Main results. First row is the results of VN-Solver using ResNet and remaining rows are results using PointNet for different seed values.The sample size for training is from** {100, 200, 1000}

| | | **Dataset: Small** | 100 | | | 200 | | | 1000 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | AUC | Accuracy | F1 | AUC | Accuracy | F1 | AUC | Accuracy | F1 |
| VN-Solver | Random Uniform color | Random | 0.52 ± 0.03 | 0.45 ± 0.0 | 0.62 ± 0.00 | 0.51 ± 0.02 | 0.43 ± 0.00 | 0.60 ± 0.00 | 0.51 ± 0.02 | 0.48 ± 0.00 | 0.65 ± 0.00 |
| | | Random(seed=3) | 0.48 | 0.50 | 0.66 | 0.51 | 0.50 | 0.0 | 0.48 | 0.50 | 0.66 |
| | | Random(seed=7) | 0.48 | 0.48 | 0.67 | 0.48 | 0.50 | 0.67 | 0.48 | 0.51 | 0.66 |
| | | Random(seed=11) | 0.48 | 0.50 | 0.66 | 0.47 | 0.51 | 0.66 | 0.48 | 0.51 | 0.66 |
| | | Random(seed=13) | 0.50 | 0.48 | 0.64 | 0.49 | 0.52 | 0.65 | 0.48 | 0.51 | 0.66 |
| | | Random(seed=29) | 0.47 | 0.50 | 0.67 | 0.49 | 0.48 | 0.65 | 0.48 | 0.51 | 0.66 |
| | | PointNetAvg seed Random | 0.5 ± 0.02 | 0.50 ± 0.2 | 0.65 ± 0.03 | 0.5 ± 0.01 | 0.50 ± 0.02 | 0.52 ± 0.29 | 0.48 ± 0.0 | 0.51 ± 0.01 | 0.66 ± 0.00 |
| | Circular Uniform color | Circular ResNet | 0.69 ± 0.13 | 0.61 ± 0.09 | **0.63** ± 0.09 | 0.75 ± 0.14 | 0.68 ± 0.12 | **0.69** ± 0.04 | 0.93 ± 0.02 | 0.85 ± 0.02 | **0.83** ± 0.03 |
| | | Circular(seed=3) | 0.7606 | 0.6760 | 0.7523 | 0.8309 | 0.5820 | 0.7310 | 0.8919 | 0.8080 | 0.8411 |
| | | Circular(seed=7) | 0.7631 | 0.6780 | 0.7557 | 0.8026 | 0.7100 | 0.7611 | 0.8919 | 0.8180 | 0.8444 |
| | | Circular(seed=11) | 0.7762 | 0.6940 | 0.7685 | 0.7828 | 0.6860 | 0.7361 | 0.8980 | 0.8180 | 0.8423 |
| | | Circular(seed=13) | 0.7699 | 0.5940 | 0.7360 | 0.7601 | 0.6920 | 0.7484 | 0.8930 | 0.8200 | 0.8459 |
| | | Circular(seed=29) | 0.7724 | 0.6760 | 0.6873 | 0.8484 | 0.7160 | 0.7732 | 0.8852 | 0.8060 | 0.8336 |
| | | Circularwith PointNet | 0.77 ± 0.01 | 0.66 ± 0.04 | 0.74 ± 0.03 | 0.80 ± 0.04 | 0.68 ± 0.05 | 0.75 ± 0.02 | 0.89 ± 0.00 | 0.81 ± 0.01 | 0.84 ± 0.00 |
| | VGAE | Random(seed=3) | 0.8186 | 0.6920 | 0.7623 | 0.8076 | 0.8076 | 0.7720 | 0.8773 | 0.8180 | 0.8198 |
| | | Random(seed=7) | 0.8240 | 0.7300 | 0.7636 | 0.7849 | 0.6660 | 0.7411 | 0.8835 | 0.8320 | 0.8235 |

| | | **Dataset: Large** | 100 | | | 200 | | | 1000 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | AUC | Accuracy | F1 | AUC | Accuracy | F1 | AUC | Accuracy | F1 |
| VN-Solver | Uniform color | Random | 0.53 ± 0.03 | 0.48 ± 0.02 | 0.51 ± 0.29 | 0.51 ± 0.06 | 0.47 ± 0.00 | 0.64 ± 0.00 | 0.47 ± 0.08 | 0.41 ± 0.02 | 0.58 ± 0.01 |
| | | Random(seed=3) | 0.44 | 0.68 | 0.81 | 0.44 | 0.69 | 0.82 | 0.45 | 0.71 | 0.83 |
| | | Random(seed=7) | 0.52 | 0.69 | 0.82 | 0.54 | 0.69 | 0.82 | 0.46 | 0.70 | 0.82 |
| | | Random(seed=11) | 0.57 | 0.69 | 0.82 | 0.44 | 0.69 | 0.82 | 0.45 | 0.69 | 0.81 |
| | | Random(seed=13) | 0.45 | 0.68 | 0.81 | 0.44 | 0.68 | 0.81 | 0.46 | 0.70 | 0.83 |
| | | Random(seed=29) | 0.48 | 0.67 | 0.80 | 0.53 | 0.67 | 0.80 | 0.45 | 0.68 | 0.81 |
| | | PointNet Avg seed Random | 0.5 ± 0.07 | 0.68 ± 0.01 | 0.81 ± 0.01 | 0.5 ± 0.04 | 0.68 ± 0.01 | 0.81 ± 0.00 | 0.5 ± 0.0 | 0.70 ± 0.00 | 0.81 ± 0.01 |
| | Circular Uniform color | Circular ResNet | 0.83 ± 0.08 | 0.72 ± 0.16 | **0.74** ± 0.10 | 0.93 ± 0.04 | 0.90 ± 0.07 | **0.90** ± 0.08 | 0.98 ± 0.01 | 0.94 ± 0.02 | **0.94** ± 0.03 |
| | | Circular(seed=3) | 0.8675 | 0.6640 | 0.7515 | 0.8525 | 0.8120 | 0.8464 | 0.9647 | 0.9380 | 0.9455 |
| | | Circular(seed=7) | 0.3027 | 0.5340 | 0.6962 | 0.8378 | 0.7560 | 0.7626 | 0.9457 | 0.9140 | 0.9250 |
| | | Circular(seed=11) | 0.9251 | 0.6260 | 0.7332 | 0.7557 | 0.7960 | 0.8355 | 0.9402 | 0.9200 | 0.9288 |
| | | Circular(seed=13) | 0.8043 | 0.7720 | 0.8021 | 0.8574 | 0.8120 | 0.8423 | 0.9396 | 0.9120 | 0.9228 |
| | | Circular(seed=29) | 0.7442 | 0.7740 | 0.8114 | 0.8639 | 0.8100 | 0.8425 | 0.9619 | 0.9400 | 0.9472 |
| | | Circularwith PointNet | 0.73 ± 0.25 | 0.67 ± 0.10 | 0.76 ± 0.05 | 0.83 ± 0.04 | 0.80 ± 0.02 | 0.83 ± 0.04 | 0.95 ± 0.01 | 0.92 ± 0.01 | 0.93 ± 0.01 |