Exploring Graph Neural Networks for Quantum-inspired Operational Research

Panithi Suwanno

30 April 2024

1 Motivation

This investigation is centered on the importance of Quadratic Unconstrained Binary Optimization (QUBO) in quantum computing. These problems give hope for solving complex optimization tasks in various industries. However, solving them takes a lot of time and resources because they are computationally complex.

The Traveling Salesperson Problem (TSP) is a well-known optimization problem that vividly illustrates the difficulties of solving Quadratic Unconstrained Binary Optimization (QUBO). Computing accurate solutions for TSP is very computation-intensive due to its quadratic nature [1]. As a result, businesses waste time and struggle with suboptimal logistics, route planning, and resource management.

If we can find ways to speed up solving QUBO problems, it will open up a world of possibilities in business. When things are computed more quickly, enterprises can act on time, save costs through optimized resource allocation, and enhance operational efficiency. Take TSP, for example; if computations are done faster, routes can be optimized in real-time, leading to fuel and cost savings and better delivery schedules.

Additionally, when we tackle the computational challenges of QUBOs, we set the ground for broader usage of quantum-inspired computing technologies. As optimization tasks become more dependent on quantum computing by businesses [2], innovations will be made around QUBO solutions' computation, leading to competitiveness and industry-specific application development. In conclusion, companies need to speed up their calculations for QUBOs to see new levels of efficiency, productivity, and strategic decision-making abilities.

2 Background

Modern methods for solving complicated problems in optimization and decision-making frequently include computer tools, algorithmic methodologies, and mathematical models. Pursuing effective solutions propels innovation and the uptake of cutting-edge techniques in fields ranging from supply chain management and logistics to computational biology and finance.

Three main paradigms have emerged in optimization research and practice. These are Quadratic Unconstrained Binary Optimization (QUBO), the Traveling Salesperson Problem (TSP), and Graph Neural Networks (GNNs). Each has different insights and abilities in solving optimization problems, ranging from combinatorial optimization to graph-based learning tasks.

This subsection examines these paradigms by discussing their principles, applications, and implications for present-day optimization challenges. We start with QUBO, a powerful way of representing optimization problems as binary quadratic equations. Then, we consider TSP, a classic combinatorial optimization problem that has attracted decades of research and invention. Lastly, we explore the new field of GNNs, where graphs are used for effective learning over complex datasets through inference.

2.1 Quadratic Unconstrained Binary Optimization

The concept of solving optimization problems by converting them into binary quadratic equations is called Quadratic Unconstrained Binary Optimization (QUBO). The word binary means that QUBO variables are either 0 or 1 [3]. This makes it useful for discrete decision-making.

Quantum computing is related to QUBO by expressing optimization problems as binary quadratic equations. According to [4], quantum computing operates based on quantum mechanics principles, and algorithms that can be solved using these principles can be represented in terms of QUBO.

Unconstrained refers to the lack of explicit constraints in formulating the objective function. The

initial idea behind QUBO came from physicists attempting to use quantum technology for mathematical problem-solving. To avoid complicating things, they started with an unconstrained version where no additional constraints were imposed on the objective function. Such a foundational approach allowed for more straightforward mathematical representation, facilitating research into optimization applications of quantum computing.

Nonetheless, intricate constraints and conditions in real-life optimization problems compel us to discard constraint-less solutions [5]. Given these problems' inherent complexity and difficulty, it is necessary to add restrictions that will help guide optimization procedures. However, just because there are no explicit constraints in QUBO formulations does not mean that we should ignore constraints altogether. Instead, they can still be considered by treating them as penalty terms in an objective function. This method permits one to work with constrained optimization under the more general and flexible framework QUBO provides.

This subsection will expand on this approach while showing how QUBO formulations may include penalties for representing constraints to increase their relevance in solving real-world optimization problems.

2.2 Traveling Salesperson Problem

Logistics concerns are all about optimization. Incorporating QUBO in logistics solutions has allowed companies to develop strategies that save time and money. This has allowed businesses to make better decisions and improve their performance [6]. In this aspect, QUBO is a handy tool in enhancing supply chain and logistic operations, leading to increased resource efficiency and reduced operational costs.

When we understand what QUBO is and why it is essential, we can look at the equations used to solve the TSP derived from general transport systems. Here, we are supposed to find the best route to reduce costs (usually represented by financial expenses) as much as possible while ensuring timely delivery. These equations employ a binary variable x, whose definition is based on whether something happens or does not happen at any location, and d, which stands for the distance between two places or the cost involved when moving between them.

Where $x_{i,t} \in \{0, 1\}$ and $x_{i,t}$ is equal to 1 when node i is visited as a t-th node. And $d_{i,j}$ is the distance from node i to node j. We will get the following quadratic equation:

$$\text{Maximize } y = \sum_{i,j,t} d_{i,j} \cdot x_{i,t} \cdot x_{j,t+1}$$

- Typically, when employing QUBO for the TSP, outcomes are presented in matrix form. This matrix features a city or node axis aligned with the time steps throughout the journey see Table 1. In line with common sense, if there are N cities, the matrix will encompass N time steps.

City/time	t_0	t_1	t_2	t_3	t_4
A	1	0	0	0	0
B	0	1	0	0	0
C	0	0	0	1	0
D	0	0	1	0	0
E	0	0	0	0	1

Table 1: One example of a TSP answer format to give a better idea.

- This stems from the imperative condition we must address: the salesperson or vehicle (depending on individual interpretation) can only occupy a single unit of time.

$$\sum_{i} x_{i,t} = 1 \ \forall \ t$$

- Additionally, it's crucial to ensure that each node or city is visited only once, a prerequisite for establishing the Hamiltonian cycle.

$$\sum_{t} x_{i,t} = 1 \ \forall \ i$$

Concerning the part in the last subsection, since we have two conditions in this minimization equation, it can be transformed into a single equation without restrictions to correspond to the QUBO definition. We do this by converting each linear constraint into a quadratic equation and adding it to our main objective equation. These additional terms are known as penalty terms.

$$\begin{aligned}\text{Maximize } y = &\sum_{i,j,t} d_{i,j} \cdot x_{i,t} \cdot x_{j,t+1} \\ &+ \mathcal{A} \sum_{i} (1 - \sum_{t} x_{i,t})^2 \\ &+ \mathcal{B} \sum_{t} (1 - \sum_{i} x_{i,t})^2\end{aligned}$$

- The variables $\mathcal{A}$ and $\mathcal{B}$ are commonly referred to as Lagrange multipliers, used to weigh the penalty terms and enforce constraints in optimization problems.

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) is a new and exciting approach in the broader area of machine learning (ML) and artificial intelligence (AI) for dealing with Operations Research (OR) problems that are inherently complex [7][8]. GNNs based on deep learning can be used to model complex graph structures effectively, thus making them an ideal choice for solving various combinatorial optimization problems. GNNs capture more detailed links between different parts of graphs than any other technique, which is why they represent such significant potential for OR challenges where efficiency and scalability matter most.

Mathematical graph structures consist of nodes and edges representing objects and their connections. They embed this information in a GNN by processing it repeatedly across different points and links. GNN can learn complex pattern recognition via iteration on graphs because it collects local neighborhood awareness to produce node and edge embeddings that encode global graph properties. This mechanism of spreading outwards allows GNNs to grasp tangled-up dependencies within the graph to discover big samples easily overlooked by human observers who would struggle with such intricacy head-on.

3 Related Work

3.1 Graph ConvNet

This exploration is thought-provoking because it focuses on using Artificial Intelligence (AI) as a different method for solving the Traveling Salesman Problem (TSP), apart from traditional methods. They have performed graph sparsification [9] in this experiment and used machine learning techniques to estimate all possible distances, i.e., $d_{i,j}$ distance matrix variables as described in Section 2.2.

After obtaining the sparsified graph, a beam search algorithm was employed to find the shortest path with minimum cost. In our work, we want to investigate how Quantum Computing with Quadratic Unconstrained Binary Optimization (QUBO) can be beneficial. However, QUBO poses computational slowness challenges due to the quadratic equations involved. Furthermore, hardware improvements in Quantum Computing are still under development [10].

Nonetheless, we think this study's first conclusions seem to have the potential for reducing the computation weight of QUBO. In other words, if there is a way to cut down on some possibilities, it could simplify the problem so that it could be solved by quantum machines within their capabilities. We suppose that even a minor decrease in complexity might enable us to solve the problem with Quantum Computing effectively while maintaining our understanding of its depth. Such a thing would let us tackle industry challenges quickly as soon as quantum computing becomes available.

3.2 GPS: A new TSP formulation

There are several approaches to consider when tackling the TSP, one of which is the recently proposed Graph Positioning System (GPS), contrasted with the perspective outlined in Section 2.2, where our variables denote $x_{i,t}$ representing destinations viewed as time frames to determine the optimal location for the salesperson or vehicle at time t.

In this research, a novel interpretation of binary decision variables has been proposed [11], viewing them as $x_{u,v,t}$. It signifies a transition from station u to v at time step t. Furthermore, it has been suggested that the temporal dimension t can be reduced to just $\mathbb{R}^3$ dimension, redefined as $x_{u,v,r}$ where $r \in \{0, 1, 2\}$.

Because there are so many variables, they no longer consider each time step. Instead, they only care about the order of the nodes or stations, which allows them to combine variables. However, this means that the constraints become much more complicated, and we won't get into that here since it's too similar to what they did.

It's unfortunate that despite reducing the variables in the final dimension, the total number of variables remains substantial, estimated at around $N \times N \times 3$, compared to the $N \times N$ variables used in our previous equation $x_{i,j}$. Consequently, this approach may not see widespread adoption compared to the last method.

4 Hypothesis and Hypothesis Testing

We can reduce the decision variables in Quadratic Unconstrained Binary Optimization (QUBO) computations, decreasing the time required to compute the optimal solution and prove its optimality. This involves determining which variables can be pruned without needing to be considered by the quantum or quantum-inspired solver. This task could be delegated to Graph Neural Networks (GNNs) for decision-making.

However, preliminary testing of our hypotheses is essential to verify: **(i) whether reducing**

variables leads to a reduction in computation time, and **(ii) whether the integration of GNNs yields favorable results**. We cannot afford to overlook these considerations, as they may exceed the boundaries of capability, such as the absence of discernible patterns or the ability to infer, among others. These two factors are vital to ensure the study is heading in the right direction.

The following subsection will present the preliminary results of our initial testing to validate the hypotheses mentioned earlier. It is important to note that these results may be inaccurate or contain errors due to the author's limited knowledge and lack of peer review. However, we take pride in presenting what we have implemented and the ideas we have explored in a tangible form.

We drew inspiration from Graph ConvNet papers [9], as discussed in Section 3.1. We implemented this paper to suit our understanding, with the underlying code simplified from the original paper equations to a degree appropriate for our usage. Our implementation will aid in running on Quantum-inspired hardware to solve QUBO.

4.1 Implementation

4.1.1 Node Embedding

Node embeddings are very important in the hope of helping GNNs understand and navigate the structural dynamics of tour problems. By representing each node's spatial coordinates with a feature-rich embedding, we can enable GNNs to grasp spatial relations inherent in the domain of the tour problem. These embeddings contain fundamental geometric properties and topological insights, which help networks understand how nodes are connected or close to one another within a given graph representing the problem.

Additionally, using x and y coordinates as node features, as shown in Figure 1, is advantageous because it gives an intuitive representation of where nodes are located in Euclidean space. Any model dealing with routing optimization problems, such as TSPs, should not ignore this information about the distance between points or areas. Such details guide the decision on which edges can be removed during simplification based on underlying geometry knowledge about these types of problems. Our approach involves embedding high-dimensional spaces with lots of contexts related to location, thus allowing for better utilization by GNNs that may lead to more efficient solutions when applied to edge sparsification tasks, which require understanding the geometric relationship between pairs of points

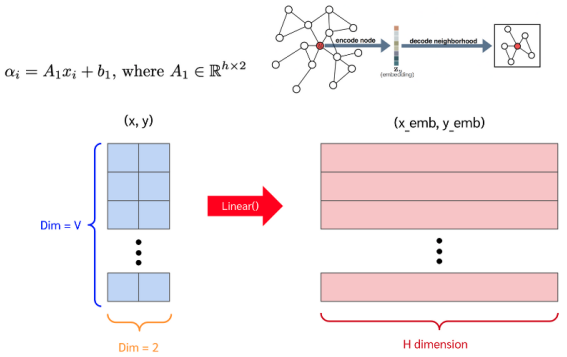

Figure 1: Embed node in Euclidean space $\mathbb{R}^2$ into embedding space $\mathbb{R}^H$.

4.1.2 Edge Embedding

Edge embeddings are essential for our goal of improving QUBO optimization through ML-based sparsification methods. We encode problem edges' fundamental properties and relatedness into dense feature representations. These features allow GNNs to navigate the intricate topology of our problems more accurately and effectively.

Distance Matrix Embeddings The distance matrix provides insights into how nodes in a problem graph are arranged spatially, giving a quantitative indication of their similarity or dissimilarity. This information can be used as edge embeddings by us such that GNNs can recognize and utilize spatial relationships inherent in the structure of graphs, thereby enabling them to learn meaningful representations reflective of underlying Figure 2 topologies.

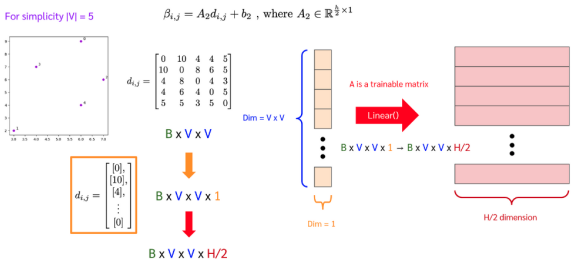

Figure 2: Embed distance matrix information.

Neighborhood Embeddings We know that the distance matrix is essential for embedding edges, but we should also pay attention to neighborliness to increase the expressive ability of our GNNs model. We want to catch the local structures' characteristics and relationships between

them shown in Figure 3 by considering k nearest neighbors of each point of interest (POI) node, thereby reflecting graph topology. This model includes information about the vicinity, which allows for recognizing and exploiting spatial dependencies between neighboring nodes, enriching learned embeddings, and creating more context awareness. We can adjust the value of k according to specific graph structure features and the required level of detail while capturing local connectivity patterns.

Therefore, regarding the edge embedding process within our GNNs framework, both distance matrices knowledge and neighborhood associations should be considered since they provide the complete understanding of a given graph topology, resulting in a powerful representative capacity.

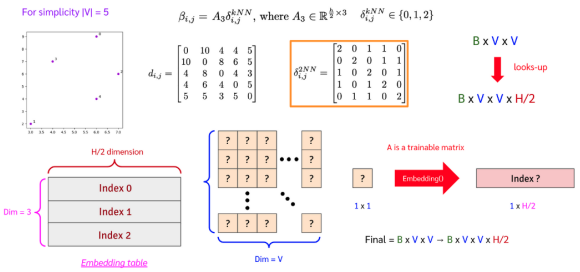

Figure 3: Embed neighborhood information.

Combine Edge Embeddings The reason why we use both the distance matrix information and k-nearest neighbor (k-NN) information in Figure 4 while deriving edge embeddings is to enrich the representation of graph topology in our graph neural network (GNN) model. This means that by introducing global geometric relationships among nodes, brought about by a distance matrix, information can be gathered about where things are positioned overall and arranged spatially within any given graph. Conversely, it provides insight into nearest neighbors' identification and corresponding distances by incorporating local connectivity patterns via k-NNs.

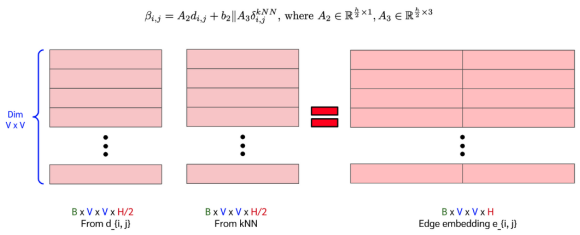

Figure 4: Concatenate distance and neighborhood information into one edge embedding information.

4.1.3 Edge Features

Weight matrices W_1 and W_2 are used with $e_{i,j}$, x_i, and x_j to calculate the edge features. These properties reflect not only the edge embedding qualities themselves but also those of connected nodes. A good representation of edges should include information about different parts of a graph; this is achieved by combining them.

$$W_1 e_{i,j} + W_2(x_i + x_j)$$

The second part of edge feature generation is focused on intrinsic attributes possessed by an individual relationship between two vertices in a network, such as weight distance or neighborhoods. It is where we learn matrix weights for transforming raw input into another form with more dimensions that can capture relevant characteristics during training

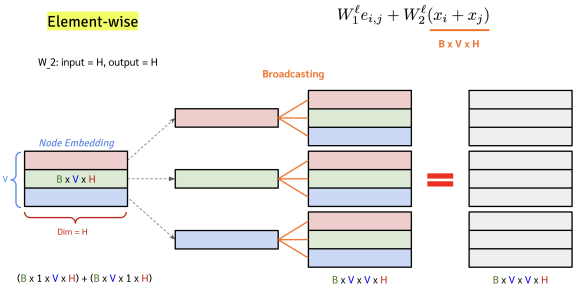

Figure 5: This figure illustrates the process of element-wise broadcasting sum in PyTorch, aimed at expanding the dimensions of a tensor.

The second part $W_2(x_i + x_j)$ applies node embeddings x_i and x_j to involve information about the connected nodes. It adds up the two nodes' embeddings using broadcasting sum as illustrated in Figure 5 and then applies another learned weight matrix W_2 to produce a combined representation of nodes that takes into account both features of participating nodes as shown in Figure 6.

Generally, edge features reflect various relations between pairs or sets among vertices within a graph, which may be attributed to some local properties of edges and global characteristics concerning vertices. Hence, such an inclusive description allows our model to effectively discover hidden graph patterns and make reasonable decisions at different stages during optimization.

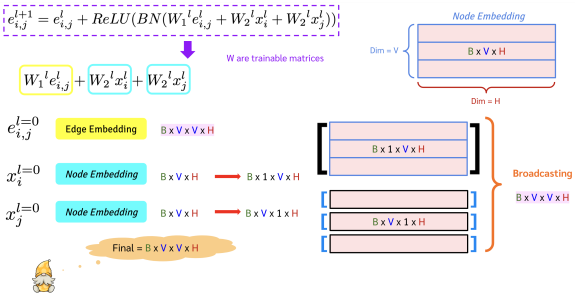

Figure 6: Edge Features combine information from two node embeddings and the edge embedding, as each edge comprises two nodes.

4.1.4 Node Features

The node features are computed using a combination of learned weight matrices W_3 and W_4 as follows:

$$W_3 x_i + \sum_{j \in \mathcal{N}(i)} W_4 x_j$$

The first part of the node features, $W_3 x_i$, is concerned with the inherent properties of the node. In this case, a weight matrix W_3 is learned during training and applied to change a raw node embedding x_i into a higher-dimensional representation that captures pertinent features.

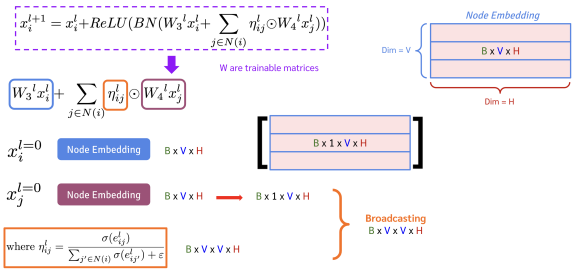

Figure 7: The first term of node features is about the information about itself (point of interest). And the second term of node features is about aggregating their neighbor information.

The second part, $\sum_{j \in \mathcal{N}(i)} W_4 x_j$, utilizes the embeddings of neighboring nodes x_j within the neighborhood $\mathcal{N}(i)$ of a given node i. We take these aggregated neighboring node embeddings and apply one more trained weight matrix W_4 to obtain a joint representation that reflects both node attributes and those of its immediate environment as illustrated in Figure 8.

To put it differently, what we know about nodes tells us much not only about them but also about their relationships within graphs, which in turn enables our model to discover underlying structures more effectively and make smarter choices during the optimization process

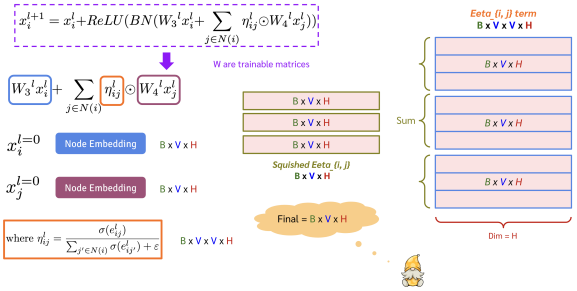

Figure 8: Incorporating the neighboring information of node i into consideration is crucial. During the aggregation process, as we aim to retain the relational aspects between nodes, the data may become intertwined with edge features. Therefore, we need to ensure the coherence of the tensor dimensions during addition. However, in this study, we prefer to keep it simple and do not consider any normalization term $\eta^{\ell}_{i,j}$ yet.

4.1.5 Graph Convolutional Layers

The significance of graph convolutional layers (GCN) lies in their ability to collect information from a graph's different nodes and edges. This information can then be used for feature representation that makes sense during subsequent processing stages. Based on our optimization framework, these layers help extract high-level features indicative of the underlying structural properties within a graph under consideration.

$$e^{\ell=0}_{i,j} = \beta_{i,j}$$

On each layer ℓ, multiple iterations are done to compute edge feature information ($e^{\ell}_{i,j}$) within the GCN. Initially ($\ell = 0$), edge features $e^{\ell=0}_{i,j}$ are set as precomputed edge embeddings $\beta_{i,j}$. In every next layer, updated versions of these features are computed by utilizing node embeddings x^{ℓ}_i, x^{ℓ}_j, along with learned weight matrices W^{ℓ}_1, W^{ℓ}_2. This is achieved through several steps such as matrix multiplication, batch normalization (BN), rectified linear unit (ReLU) activation function, and element-wise addition.

$$e^{\ell+1}_{i,j} = e^{\ell}_{i,j} + ReLU(BN(W^{\ell}_1 e^{\ell}_{i,j} + W^{\ell}_2 x^{\ell}_i + W^{\ell}_2 x^{\ell}_j))$$

Analogously, the information of node features is computed over many layers (ℓ) of GCN. In the starting layer ($\ell = 0$), the initialisation of node

features $x_i^{\ell=0}$ is done through pre-calculated node embeddings α_i.

$$x_i^{\ell=0} = \alpha_i$$

In subsequent layers, a combination of learnt weight matrices W_3^ℓ and W_4^ℓ, along with embeddings of neighboring nodes x_j^ℓ are used for updating the node features.

$$x_i^{\ell+1} = x_i^\ell + ReLU(BN(W_3^\ell x_i^\ell + \sum_{j \in \mathcal{N}(i)} W_4^\ell x_j^\ell))$$

The input of the last-layered GCN is the feature of the edge $e_{i,j}^\ell$ and node x_i^ℓ at each layer (ℓ), which are processed to produce updated edge and node features $e_{i,j}^{\ell+1}$ and $x_i^{\ell+1}$. This is done in each layer (ℓ) successively until a certain feature extraction level is achieved. The final feature representations that capture the essential characteristics of the graph can be obtained by this output from the last layer $(\ell = L)$.

$$e_{i,j}^{\ell+1}, x_i^{\ell+1} = GCN(e_{i,j}^\ell, x_i^\ell) \; \forall \; \ell \in L$$

As shown in Figure 9, ultimately, the last layered GCN's output is given into multilayer perceptron (MLP) for predicting the optimal TSP route. MLP takes edge features as input $e_{i,j}^L$, generating predicted probabilities for each edge in graph $p_{i,j}^{TSP}$. These probabilities tell about how likely it would be to include any given edge on an ideal TSP route, enabling efficient and effective route planning.

$$p_{i,j}^{TSP} = MLP(e_{i,j}^L)$$

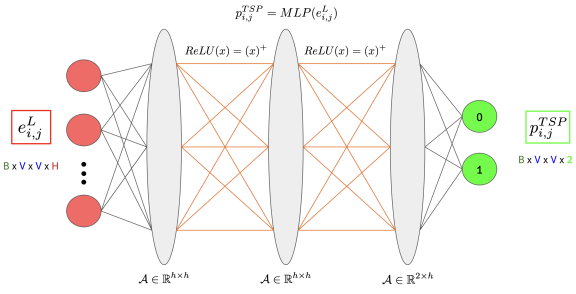

Figure 9: MLPs with ReLU activation functions after each layer except the last one.

Coding in PyTorch To show you Figure 10, torch-info summary gives an overview of the model's architecture; it states layer types, input, output shapes, and the number of parameters. This helps us to understand how data moves through our network and where we can optimize it. It is also useful when verifying that the model has been configured correctly and will work.

Our Single GNN Layer

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
myGCNlayerV1 (myGCNlayerV1)	[1024, 20, 300]	[1024, 20, 300]	--	True
├─Linear (W1)	[1024, 20, 20, 300]	[1024, 20, 20, 300]	90,300	True
├─Linear (W2)	[1024, 20, 300]	[1024, 20, 300]	90,300	True
├─Linear (W3)	[1024, 20, 300]	[1024, 20, 300]	90,300	True
├─Linear (W4)	[1024, 20, 300]	[1024, 20, 300]	90,300	True
├─BatchNorm2d (BNedge)	[1024, 300, 20, 20]	[1024, 300, 20, 20]	600	True
├─BatchNorm1d (BNnode)	[1024, 300, 20]	[1024, 300, 20]	600	True
├─ReLU (ReLU)	[1024, 20, 20, 300]	[1024, 20, 20, 300]	--	--
├─ReLU (ReLU)	[1024, 20, 300]	[1024, 20, 300]	--	--

Total params: 362,400
Trainable params: 362,400

Figure 10: Summary of our model architecture in one GNN layer.

Loss function We evaluate our models during training using the Negative Log Likelihood (NLL) loss function shown in Figure 11. NLL loss is commonly used in classification tasks because it measures the difference between actual label values and predicted probabilities. Our method accurately estimates each TSP route edge occurrence probability by minimizing NLL loss. Additionally, Adam's optimizer is utilized at this stage. Adam modifies model parameters well enough for gradients from the loss function since it is an adaptive learning rate optimization approach. The training process convergence sped up thanks to changes made by Adam on learning rates for each parameter, adaptively improving the overall performance of the model itself based on this gradient descent algorithm

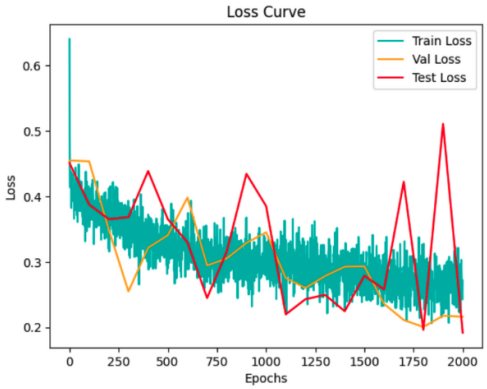

Figure 11: Our first model training loss curves (without any well-fine-tuning procedure).

4.2 Deployment

In this part, we will integrate the concepts discussed in Section 4.1 with the objective equations (with quadratic penalty terms) presented in Sec-

tion 3.2's paper [11], as follows:

$$\text{Minimize } y = \sum_{i,j,r} d_{i,j} \cdot x_{i,j,r} + \sum_{i,j,k} \lambda \cdot (x_{j,i,2} \cdot x_{k,j,2} + x_{k,i,2} - x_{j,i,2} \cdot x_{k,i,2} - x_{k,j,2} \cdot x_{k,i,2})$$

We should note that this formula might not work well when used directly. However, we think it will perform better than the usual TSP QUBO formulas if we apply it when GNNs indicate some variables that can be ignored in the graph. But before doing so, let us consider an example output given by GNNs in Figure 12.

The figure shows that the GNNs are confident in discarding sure edges from the graph. Consequently, since we already have variables $x_{i,j,r}$ representing the existence of edge $e_{i,j}$, the equation mentioned earlier can be effectively utilized with our GNNs in the long run. This is because the traditional equations in Section 2.2 have variable growth of $N \times N$. If our GNNs perform well enough, we will have fewer variables because the variables will grow according to the remaining edges in the graph instead of the constant number of nodes N throughout time, which would be $|E|_{GNNs} \times 3$. This setup will test our hypothesis regarding the reduction in the number of variables and its impact on solving the QUBO problem.

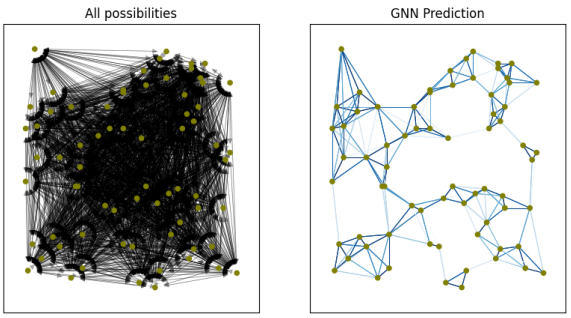

Figure 12: (Left) A distance matrix of size $|V| = 70$ vertices with a total of $|E| = 4,830$ edges excluding self-loops, (Right) The graph where our GNNs have pruned edges, with the remaining edges shown as a heatmap. The darker the color, the more confident the model is that the edge will occur in the optimal route. After pruning, there are $|E| = 392$ remaining nodes.

The dataset used to test this experiment is called ST70, sourced from TSPLIB95, a reputable dataset commonly used for benchmarking TSP problems [12].

This experiment was performed with the Gurobi solver version 11.0.1. This solver can receive quadratic equations as inputs. We will let the solver solve the problem until it finds the best answer with proven optimality. The graph on the right-hand side of Figure 12 can be modified by reducing the number of declared $x_{i,j,r}$ variables. That is, we trust GNNs that if an edge of the graph does not appear in the heatmap, we will not initialize a variable to represent that edge, significantly reducing our variables' usability. In detail, the previous TSP formula used 70 nodes to solve this problem with $N \times N = 4,900$ variables, but our GNNs allow declaring only $|E|^2_{GNNs} \times 3 = 1,176$ variables. The experiment results are in Figure 13, which took only 12.68 seconds to get a good-looking answer.

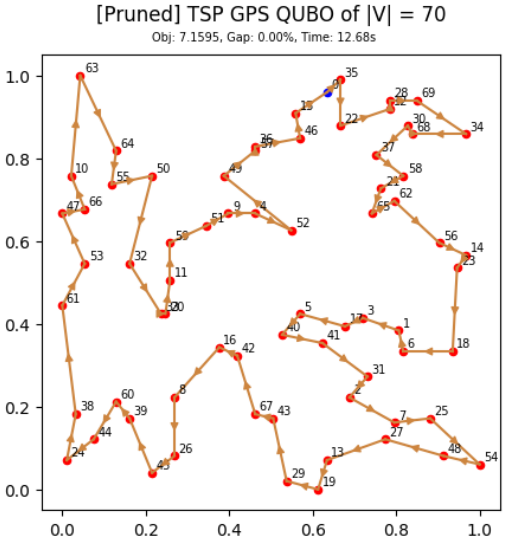

Figure 13: Experimental results of SD70 with the GPS QUBO equation mentioned above, used in conjunction with GNNs.

Nevertheless, our discoveries must be measured against the best solution. We can find the best answer from the TSPLIB95 library because they have already calculated the answer I have displayed in Figure 14. It can be seen that the format of the tour is not the same because some edges were not found in the heatmap, and we did not declare those variables. From this experiment, we can conclude that by using GNNs to cut out the variables, we can get the answer to QUBO promptly that does not affect the fast-paced plans of the business world. Even though we may not have the best answer, we still have a feasible answer that can be used interchangeably.

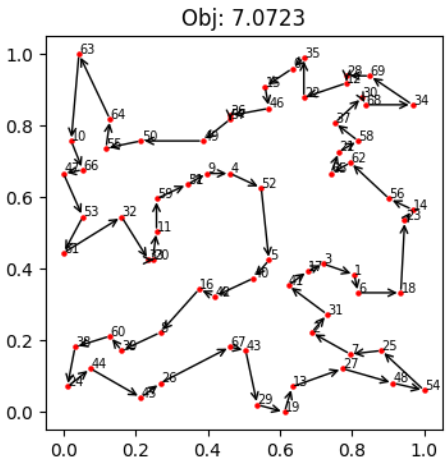

Figure 14: The optimal tour answer of ST70 can be found from TSPLIB95.

Finally, in this section, I would like to try using the image on the left-hand side of Figure 12 using the conventional TSP QUBO method, where the variables are $x_{i,t}$. We cannot reduce the variables because their dimension represents a destination we cannot ignore in the routing problem. From the experimental results in Figure 15, the results do not look good and are unlikely to be truly usable. The results of this experiment confirm our hypothesis stated earlier. It also shows the trend of interoperability of GNNs that can help accelerate the QUBO calculation of this problem.

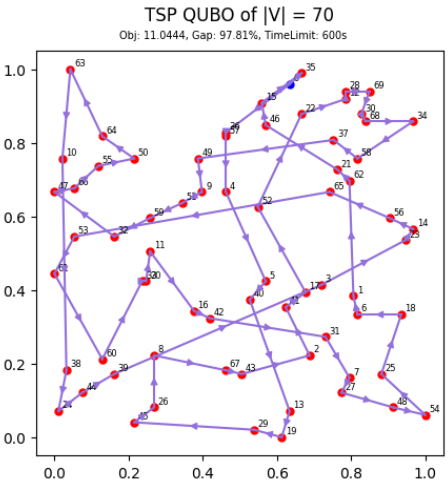

Figure 15: In this experiment, because the problem is very difficult to solve, we set the time limit to 600 seconds. This is the result of running the full-time limit, and we can see that the answer is not good and is unable to be used for logistics.

5 Proposed Method

We propose using GNN to reduce the search space of the TSP by pruning low-probability edges (paths that are unlikely to occur in the optimal solution) before formulating the problem as a QUBO.

In this work, we adopt the GPS formulation (Section 3.2) rather than the conventional node-at-time TSP QUBO. The conventional approach has $\mathbb{R}^2$ dimensionality, declaring $N \times N$ binary variables regardless of graph sparsity, edge likelihoods can bias the solution, but cannot reduce the number of declared variables. By contrast, the GPS formulation operates in $\mathbb{R}^3$, defining variables on edges and their travel order, requiring only $|E|_{GNNs} \times 3$ variables. If the GNN effectively prunes edges, $|E|_{GNNs}$ can become significantly smaller than $N \times N$, directly reducing QUBO size.

This property is particularly valuable in large-scale, real-world settings where preprocessing time can exceed solver time. Reducing the number of variables before optimization can therefore yield both computational and business benefits.

Following the initial promising results from Section 4, we extend this approach using real business destination data (with normalized coordinates for privacy). We also stress-test the model by increasing the number of destinations per driver to evaluate both the GNN's pruning quality and the solver's performance on the reduced QUBO.

5.1 Small Tweaks

5.1.1 GraphSAGE LSTM

For the edge and node features, we replaced the pure trainable matrix–based node embedding with an LSTM layer to enhance the model's expressiveness. We hope that LSTM would allow the network to better capture sequential patterns and contextual relationships within the logistics graph data, which is particularly useful when the order or path between nodes influences the outcome. Figure 16 illustrates this modification, which helped the model adapt more effectively to variations in the input graph structure.

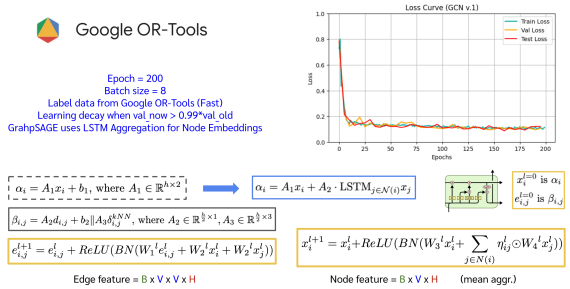

Figure 16: Replacing the pure trainable matrix–based node embedding with an LSTM layer for improved expressiveness.

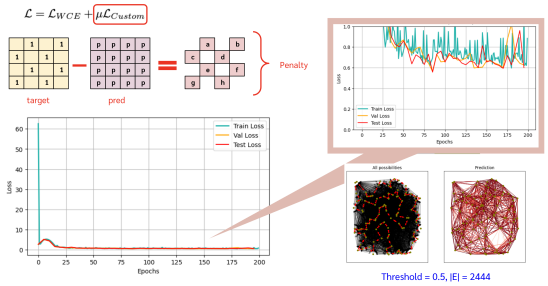

Figure 18: Modified loss function from NLL to Weighted Cross-Entropy, with an added custom penalty term to reduce false negatives and improve true positive detection.

5.1.2 Dropout

We also introduced dropout layers after each linear layer in the GCN architecture. The main goal was to reduce overfitting by randomly deactivating a portion of neurons during training. Figure 17 shows the placement of these dropout layers in our architecture.

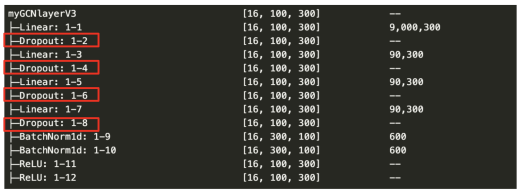

Figure 17: Dropout layers added after each linear layer in the GCN to improve generalization and reduce overfitting.

5.1.3 Custom Regularization

In addition to changing the loss function from Negative Log-Likelihood (NLL) to Weighted Cross-Entropy (WCE) to handle class imbalance, we implemented a custom regularization term. The motivation for this was to increase the penalty on false negatives, as our current model already performs well at removing unlikely usable paths from the graph, but we wanted to further improve its ability to correctly identify true positive edges. By adding this tailored penalty term, we aimed to reduce the occurrence of false negatives while maintaining a high rate of true positives. Figure 18 illustrates this adjustment in our model training process.

5.2 Interesting Findings

We found that our GNN performed well in a timely manner, even as the problem size increased, as shown in Figure 19. Since the model was already trained, we did not count the training time in our evaluation. Instead, we measured only the business logic time (how long it takes from the moment a client submits a problem to our server until the solution is computed and returned to the drivers). In terms of solution quality, if the coordinates (geographic data) change significantly from the data used in training, the underlying graph structure may differ enough to cause a higher rate of false negatives. While these false negatives do not affect the overall optimality of the solution, they do increase the pre-processing time for the QUBO model, as more variables need to be considered.

In real-life logistics, the total set of destinations rarely changes, although daily demand may fluctuate. However, these fluctuations still occur within the subset of the total destinations. This suggests that we can train the model on the complete set of destinations (a large graph) and then apply it to the relevant subgraph for each day's demand, mitigating the impact of the changes described above.

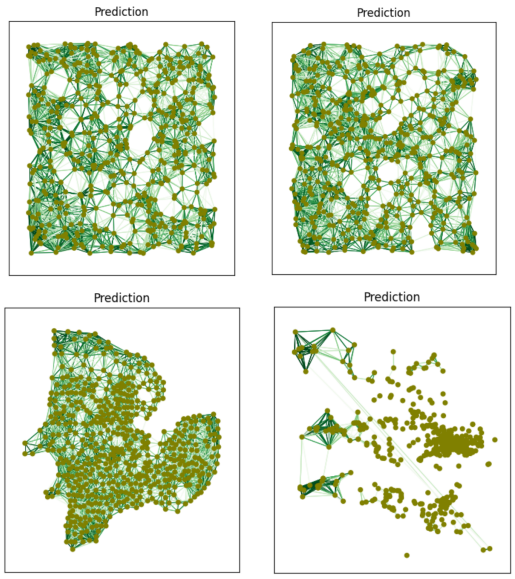

Figure 19: We generated a subgraph of 500 nodes and tested it with our GNN to evaluate its expressiveness. The resulting graphs appeared sparser and were obtained within a few seconds; however, generating the QUBO model still took about 3 hours (a limitation of our current computing resources). We anticipate that, once quantum computers are developed further, efficient processes will emerge to quickly create and map QUBO problems to qubits.

5.3 Subtour Challenges and Solutions

When applying the GPS formulation to a sparsified graph, we observed that the solver occasionally returned multiple subtours rather than a single Hamiltonian cycle. This occurs because the GPS constraints were originally designed for a fully connected graph; once many edges are removed, the formulation loses critical structural pathways that help enforce tour connectivity. The result is a mathematically valid but practically unusable set of disconnected routes (see Figure 20).

Tentative Solution. To address this, we experimented with a subtour repair process:

1. **Detect subtours** in the solver's output, e.g., by analyzing the returned edge set.
2. **Identify missing connections** between these subtours.
3. **Reintroduce edges ("clues")** into the graph to connect them.
4. **Re-run the solver** on the augmented graph.

This approach can restore optimal, connected tours (see Figure 21). However, it comes with trade-offs:

- The repair process adds computational overhead.
- The number and location of edges to be added vary per instance, making the runtime unpredictable.
- In the worst case, the process could converge to the fully connected graph, thereby removing the benefits of sparsification; however, this outcome is considered highly unlikely in practice.

Even with these caveats, the method ensures eventual feasibility and offers a path forward for combining GNN-based pruning with the GPS formulation in practical applications.

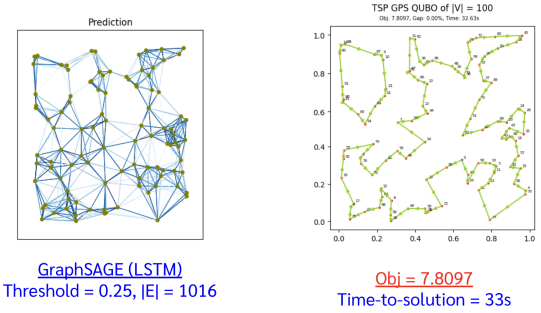

Figure 20: Example of a sparsified GPS QUBO solution producing multiple subtours instead of a single Hamiltonian cycle.

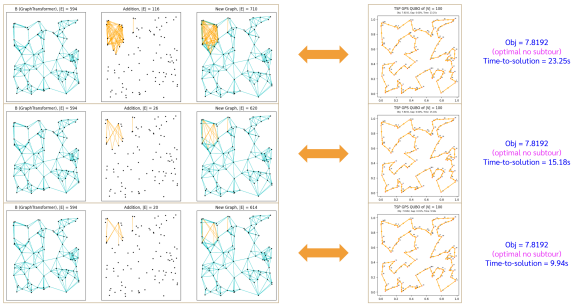

Figure 21: Example of repaired solution after subtour detection and selective edge reintroduction.

Recap QUBO GPS Formulation. The GPS TSP formulation represents each directed edge (i, j) with three binary variables $x_{i,j,r} \in \{0, 1\}$, where $r \in \{0, 1, 2\}$:

- $x_{i,j,0} = 1$: edge (i, j) does *not* appear, and i is reached before j.
- $x_{i,j,1} = 1$: edge (i, j) *appears*, and i is reached before j.
- $x_{i,j,2} = 1$: edge (i, j) does *not* appear, and j is reached before i.

The objective consists of two parts:

1. The travel cost: $\sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{r=0}^{2} d_{i,j}\, x_{i,j,1}$.
2. A set of quadratic penalty terms (weighted by λ) that enforce valid ordering and eliminate inconsistent triples (i, j, k) according to the GPS rules.

Constraints ensure that:

- Exactly one state $r \in \{0, 1, 2\}$ is active for each edge (i, j).
- Each vertex has exactly one incoming and one outgoing r=1 edge in the final tour.

This formulation was originally designed for fully connected graphs, where the abundance of available edges makes it easier for the constraints to enforce a single Hamiltonian cycle. In sparse graphs, however, many of the structural "clues" these constraints rely on are missing, which can lead to multiple disconnected subtours even if all constraints are satisfied.

Constraint-Based Subtour Elimination. While the tentative solution above can guarantee feasibility, it introduces additional runtime and an unpredictable number of reintroduced edges. We investigated a more direct fix by adding an explicit *subtour-join penalty* to the GPS QUBO itself, avoiding iterative repair.

Let $x_{i,j,r} \in \{0, 1\}$ be GPS variables with $r \in \{0, 1, 2\}$ as defined earlier: $r = 1$ denotes that edge $(i \to j)$ is selected, while $r = 0$ (resp. $r = 2$) encodes that (i, j) does not appear and i (resp. j) is reached earlier. The penalty term is:

$$\mathcal{P}_{\mathrm{SJ}} = \sum_{i,j} x_{i,j,1} \cdot (1 - x_{j,i,0}),$$

weighted by a constant $\gamma > 0$ in the QUBO objective. This discourages active edges from terminating in states that prevent proper connectivity, a configuration common in subtours.

In practice, adding $\gamma\, \mathcal{P}_{\mathrm{SJ}}$ to the GPS formulation eliminated subtours in all tested sparse graphs without densifying the instance. The approach keeps the graph size small and adds only a linear-size term to the QUBO.

6 Conclusion

This work explored the integration of GNN with the GPS QUBO formulation for the TSP to improve scalability through edge pruning.

The GPS formulation was chosen because, unlike the conventional node-at-time TSP QUBO (which always requires $N \times N$ binary variables), it defines variables on edges and their travel order. This results in $|E|_{GNNs} \times 3$ variables, where $|E_{GNNs}|$ is the number of edges in the pruned graph. When the GNN confidently removes low-probability edges, $|E|_{GNNs}$ can drop far below $N \times N$, reducing QUBO size and solver workload, which is something conventional formulations cannot achieve through pruning alone.

However, sparsification introduced a notable challenge when applied to the GPS formulation; the solver sometimes produced multiple disconnected subtours because structural cues present in a fully connected graph were missing. We addressed this through two strategies:

- A **tentative repair method** that detects subtours post-solve, selectively reintroduces connecting edges, and re-runs the solver until a single Hamiltonian cycle is obtained.
- A **constraint-based fix** that augments the GPS QUBO with a subtour-join penalty term, eliminating disconnected tours during optimization without post-processing.

Our experiments showed that the constraint-based fix consistently prevented subtours while preserving sparsity, offering a more direct and predictable solution than iterative repair. Nonetheless, the repair method remains a practical fallback when constraint tuning is not feasible.

Overall, this study demonstrates that combining GNN-guided pruning with the GPS formulation offers a scalable, flexible, and solver-efficient approach to large-scale routing problems. Future work will explore adaptive penalty weights, integration with alternative QUBO formulations, and hybrid classical–quantum deployments for real-time logistics optimization.

References

[1] H. Yasuoka, "Computational complexity of quadratic unconstrained binary optimization," *arXiv preprint arXiv:2109.10048*, 2021.

[2] M.-L. How and S.-M. Cheah, "Business renaissance: Opportunities and challenges at

the dawn of the quantum computing era," *Businesses*, vol. 3, no. 4, pp. 585–605, 2023.

[3] F. Glover, G. Kochenberger, and Y. Du, "Quantum bridge analytics i: a tutorial on formulating and using qubo models," *4or*, vol. 17, no. 4, pp. 335–371, 2019.

[4] M. Lewis and F. Glover, "Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis," *Networks*, vol. 70, no. 2, pp. 79–97, 2017.

[5] A. O. Fajemisin, D. Maragno, and D. den Hertog, "Optimization with constraint learning: A framework and survey," *European Journal of Operational Research*, 2023.

[6] F. Hernández, K. Díaz, M. Forets, and R. Sotelo, "Application of quantum optimization techniques (qubo method) to cargo logistics on ships and airplanes," in *2020 IEEE Congreso Bienal de Argentina (ARGENCON)*, pp. 1–1, IEEE, 2020.

[7] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," *Advances in neural information processing systems*, vol. 31, 2018.

[8] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[9] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.

[10] T. Ichikawa, H. Hakoshima, K. Inui, K. Ito, R. Matsuda, K. Mitarai, K. Miyamoto, W. Mizukami, K. Mizuta, T. Mori, *et al.*, "A comprehensive survey on quantum computer usage: How many qubits are employed for what purposes?," *arXiv preprint arXiv:2307.16130*, 2023.

[11] S. Gonzalez-Bermejo, G. Alonso-Linaje, and P. Atchade-Adelomou, "Gps: A new tsp formulation for its generalizations type qubo," *Mathematics*, vol. 10, no. 3, p. 416, 2022.

[12] G. Reinelt, "Tsplib95," *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*, vol. 338, pp. 1–16, 1995.