Exploring Graph Neural Networks for Quantum-inspired Operational Research

Panithi Suwanno

30 April 2024

1 Motivation

This work focuses on solving large and complex optimization problems that appear in real-world logistics. Among the many possible formulations, we choose Quadratic Unconstrained Binary Optimization (QUBO) because it can be solved by both classical algorithms and quantum computers. This dual compatibility makes QUBO an attractive long-term investment: models developed in QUBO form today can be run on classical hardware now, and later mapped directly to qubits when large-scale quantum computers become available.

The Traveling Salesman Problem (TSP) is used here as a representative example. TSP is a well-known combinatorial optimization problem and is difficult to solve exactly for large instances [1]. In many businesses, solving such problems quickly is essential for making time-sensitive decisions in route planning, scheduling, and resource allocation.

If we can reduce the time and resources needed to solve QUBO models, businesses can gain faster and more reliable solutions without waiting for quantum hardware to mature. When quantum computers become practical, these same QUBO models can be deployed to exploit their potential speedups. This approach aligns short-term operational improvements with long-term quantum readiness.

2 Background

Optimization problems in industry are often solved using computer algorithms, mathematical models, and specialized tools. Methods that improve solution speed and quality can have a major impact in areas such as supply chain management, logistics, computational biology, and finance.

This work focuses on three key concepts: Quadratic Unconstrained Binary Optimization (QUBO), the Traveling Salesman Problem (TSP), and Graph Neural Networks (GNNs). QUBO provides a flexible way to model optimization problems. TSP is a classic example of a hard combinatorial optimization problem. GNNs are a modern machine learning method that can learn from graph-structured data. Understanding each of these will help explain our approach.

2.1 Quadratic Unconstrained Binary Optimization

QUBO is a mathematical model where the objective function is a quadratic polynomial of binary variables $x_i \in \{0, 1\}$ [2]. It is a flexible way to express many discrete decision-making problems. A key reason for using QUBO in this work is that the same formulation can be solved with classical algorithms today and can be mapped directly to quantum annealers or gate-based quantum algorithms in the future [3]. This makes QUBO suitable for developing optimization models that are ready for implementation on quantum computers.

The term *unconstrained* means that there are no explicit constraints in the formulation. Instead, constraints are added as *penalty* terms in the objective function. This approach allows constrained problems to be handled in the same QUBO framework by tuning penalty weights that enforce feasibility [4].

2.2 Traveling Salesman Problem

The TSP asks for the shortest possible route that visits each city exactly once and returns to the start. In QUBO form, binary variables $x_{i,t}$ indicate if city i is visited at position t in the tour, and $d_{i,j}$ is the distance from i to j. A simple objective is:

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

Two constraints are needed: each city must be visited once, and only one city is visited at each time step:

$$\sum_i x_{i,t} = 1 \quad \forall t, \qquad \sum_t x_{i,t} = 1 \quad \forall i.$$

These constraints are added to the objective as penalties:

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

where $\mathcal{A}$ and $\mathcal{B}$ are penalty weights (Lagrange multipliers).

2.3 Graph Neural Networks

GNNs are a deep learning approach designed for graph-structured data [5, 6]. In a graph, nodes represent entities and edges represent their relationships. GNNs update node and edge features by repeatedly passing information along edges. This allows them to capture both local and global graph patterns.

Because many optimization problems, including TSP, can be represented as graphs, GNNs are well-suited for predicting properties such as edge importance or node labels. This makes them a powerful tool for guiding combinatorial optimization methods, especially when combined with QUBO formulations.

3 Related Work

3.1 Graph ConvNet

Recent research has explored the use of Artificial Intelligence (AI) as an alternative way to solve the Traveling Salesman Problem (TSP), instead of relying only on traditional methods. In one study, graph sparsification was applied [7], and machine learning techniques were used to estimate all possible distances, $d_{i,j}$, as described in Section 2.2.

After sparsifying the graph, a beam search algorithm was applied to find the shortest path with the minimum cost. In contrast, our work focuses on Quantum Computing with Quadratic Unconstrained Binary Optimization (QUBO). While QUBO can model TSP effectively, it faces computational slowdowns because of its quadratic structure, and large-scale quantum hardware is still under development [8].

However, this AI-based approach offers a useful insight: if we can remove unlikely edges before building the QUBO model, we may reduce the problem size and make it easier for quantum hardware to solve in the future. Even a small reduction in complexity could make quantum solutions more practical for industry-level problems once the technology matures.

3.2 GPS: A New TSP Formulation

Another recent approach to TSP is the Graph Positioning System (GPS) formulation [9]. This method takes a different view from the traditional formulation in Section 2.2, where variables are defined as $x_{i,t}$, meaning city i is visited at position t in the tour.

Instead, GPS uses variables $x_{u,v,t}$, representing a move from city u to city v at time step t. The authors further reduce the temporal dimension t into an order index $r \in \{0, 1, 2\}$, resulting in variables $x_{u,v,r}$ in a $\mathbb{R}^3$ space. This change combines time steps into fewer categories by focusing on the order of visits rather than the exact step number.

In a fully connected graph, GPS still requires about $N \times N \times 3$ variables, which is larger than the $N \times N$ variables in the conventional formulation. Our interest in GPS is not in reducing variable count directly, but in adopting its *edge-based* point of view. This perspective fits well with edge pruning, because variables correspond to specific edges rather than fixed node positions. In contrast, conventional TSP formulations cannot reduce their variable count through pruning, since the number of nodes is fixed by user requirements or daily demand.

By using GPS's edge-based representation together with Graph Neural Networks (GNNs) to remove low-probability edges, we aim to significantly reduce the effective number of variables before building the QUBO model. This can make the problem more suitable for future quantum hardware while preserving solution quality.

4 Hypothesis and Hypothesis Testing

We hypothesize that it is possible to reduce the decision variables in Quadratic Unconstrained Binary Optimization (QUBO) computations, thereby decreasing the time required to compute and prove the optimal solution. The idea is to identify variables that can be pruned before passing the problem to a quantum or quantum-inspired solver. This pruning task can be delegated to Graph Neural Networks (GNNs) for automated decision-making.

To validate this hypothesis, we first test two key questions: **(i) does reducing variables lead to measurable reductions in computation**

time? and **(ii) can GNN-based pruning deliver accurate and useful results?** These are critical because the method's success depends on the presence of patterns that GNNs can learn, as well as their ability to generalize. Without these properties, the approach would not yield meaningful benefits.

The following subsections present the preliminary results of our initial testing. These results are not peer-reviewed and may contain inaccuracies due to the author's limited knowledge. Nonetheless, they represent tangible progress in implementing and testing our ideas.

Our work draws inspiration from the Graph ConvNet approach in [7], as discussed in Section 3.1. We reimplemented their ideas in a simplified form to suit our needs, making them compatible with quantum-inspired hardware for solving QUBOs.

4.1 Implementation

4.1.1 Node Embedding

Node embeddings play a crucial role in enabling GNNs to capture the structural and spatial characteristics of TSP instances. By representing each node's (x, y) coordinates as features (Figure 1), we provide the model with both geometric and topological information. This allows GNNs to reason about distances, connectivity, and spatial relationships, which are essential for identifying edges that can be safely pruned.

Embedding nodes in $\mathbb{R}^H$ preserves spatial context in a form suitable for GNN processing, facilitating edge sparsification decisions based on geometry-aware learning.

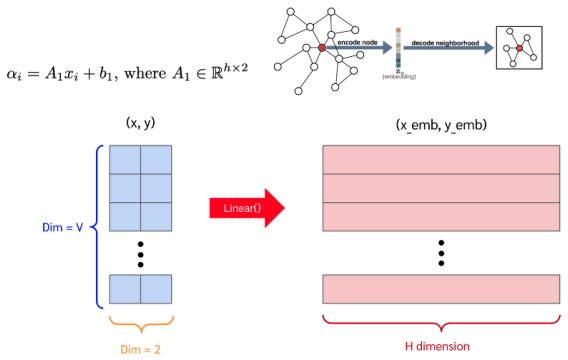

Figure 1: Embedding nodes from Euclidean space $\mathbb{R}^2$ into embedding space $\mathbb{R}^H$.

4.1.2 Edge Embedding

Edge embeddings encode the relationships between node pairs, serving as the foundation for learning graph topology in a way that supports QUBO optimization.

Distance Matrix Embeddings The distance matrix captures global spatial relationships between nodes (Figure 2). Encoding this as an edge feature allows the GNNs to reason about geometric proximity and prioritize shorter connections.

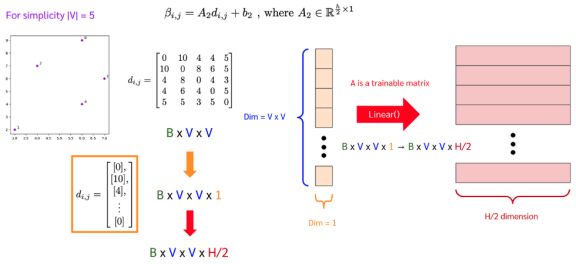

Figure 2: Embedding distance matrix information into edge features.

Neighborhood Embeddings Local connectivity is equally important. We use the k-nearest neighbors (k-NN) of each node (Figure 3) to provide the GNNs with neighborhood context, enabling it to detect local patterns and dependencies.

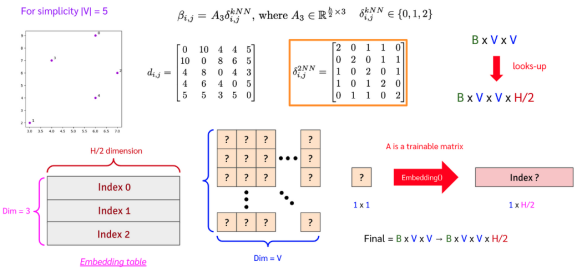

Figure 3: Embedding neighborhood information using k-nearest neighbors.

Combining Edge Embeddings By concatenating the distance matrix and k-NN features (Figure 4), we incorporate both global and local structure, providing a richer input representation for the GNNs.

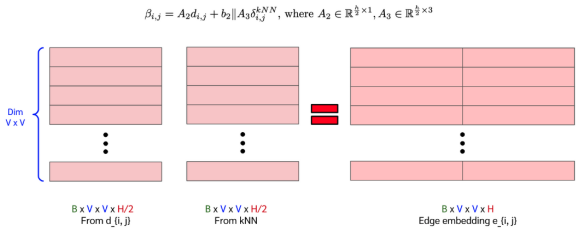

Figure 4: Combining distance and neighborhood information into a unified edge embedding.

4.1.3 Edge Features

We combine edge embeddings and node embeddings to compute edge features:

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

Here, W_1 and W_2 are learned weight matrices. The first term incorporates the edge's intrinsic representation; the second aggregates the features of the two connected nodes (Figures 5 and 6).

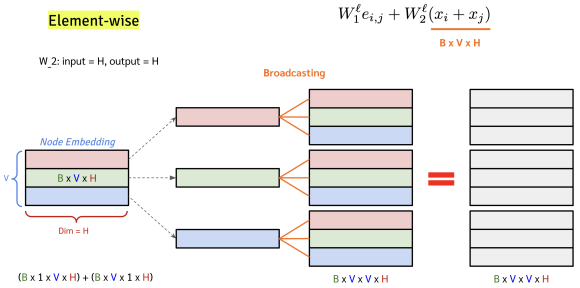

Figure 5: Element-wise broadcasting sum in PyTorch for node feature combination.

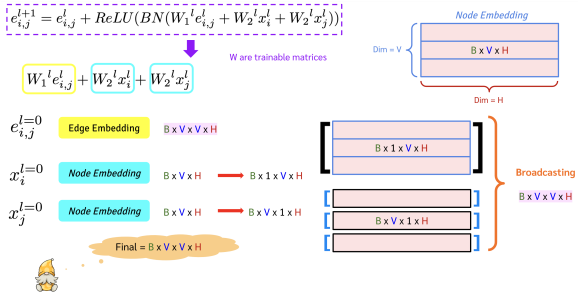

Figure 6: Edge features combine node embeddings and edge embeddings.

4.1.4 Node Features

Node features are updated using:

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

The first term transforms the node's own embedding; the second aggregates features from neighboring nodes (Figures 7 and 8).

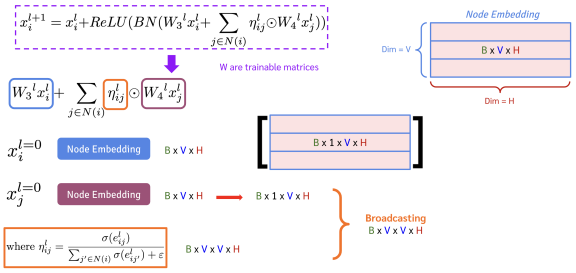

Figure 7: Node features: self-information vs. neighbor aggregation.

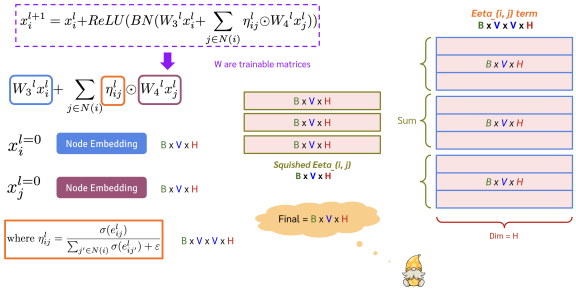

Figure 8: Aggregating neighboring information into node features.

4.1.5 Graph Convolutional Layers

GCN layers iteratively update node and edge features, starting with precomputed embeddings:

$$e_{i,j}^0 = \beta_{i,j}, \quad x_i^0 = \alpha_i$$

Each layer uses batch normalization, ReLU activations, and residual connections to refine features. The final edge features are passed to an MLP for predicting edge inclusion probabilities (Figures 9 and 10).

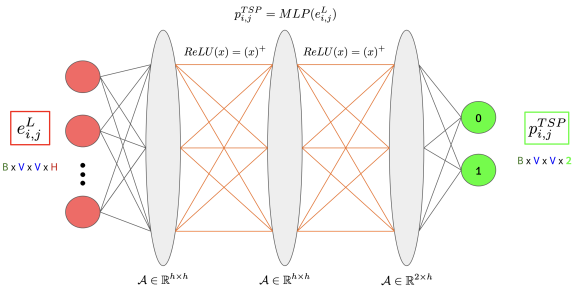

Figure 9: MLP layers for final edge probability prediction.

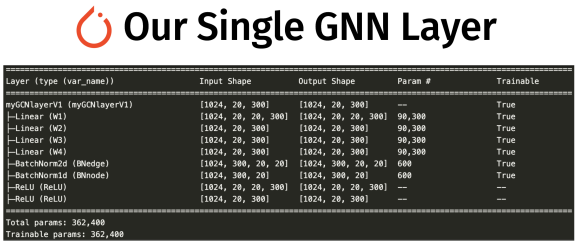

Figure 10: Torch summary of a single GNN layer.

Loss Function We use Negative Log Likelihood (NLL) loss with Adam optimization (Figure 11). NLL measures the gap between predicted edge probabilities and ground truth, while Adam adaptively adjusts learning rates for faster convergence.

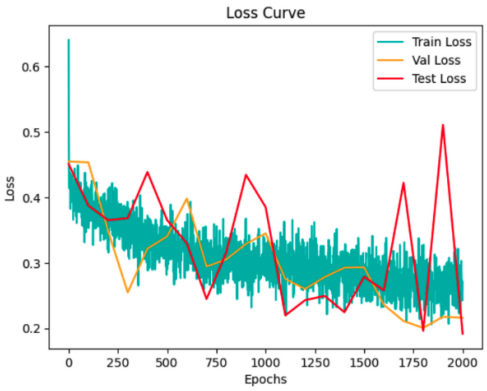

Figure 11: Initial training loss curves.

4.2 Deployment

We integrate our GNNs-based pruning strategy with the GPS QUBO formulation from [9]. The core idea is that pruning reduces the variable set from the full $N \times N \times 3$ formulation to $|E^{\text{MLP}}|_{\alpha>\tau} \times 3$, where E^{MLP} denotes the set of edges evaluated in the last-layer MLP of the GNNs, α is the predicted edge probability, and τ is the pruning confidence threshold.

As shown in Figure 12, for ST70 dataset ($|V| = 70$) from TSPLIB95, we reduced the edge set from $|E| = 4,830$ to $|E^{\text{MLP}}|_{\alpha>\tau} = 392$. This reduced the QUBO variable count from $4,900$ to $1,176$. Using the Gurobi Optimizer (version 11.0.1), solving this reduced formulation took only 12.68 seconds (Figure 13).

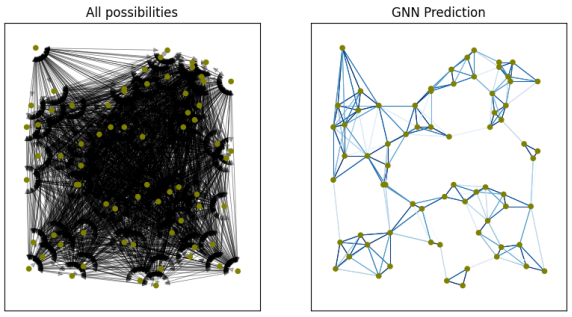

Figure 12: ST70: full graph vs. GNNs-pruned edges. The heatmap on the right shows edges retained above the confidence threshold τ. Darker edges indicate higher predicted probability α.

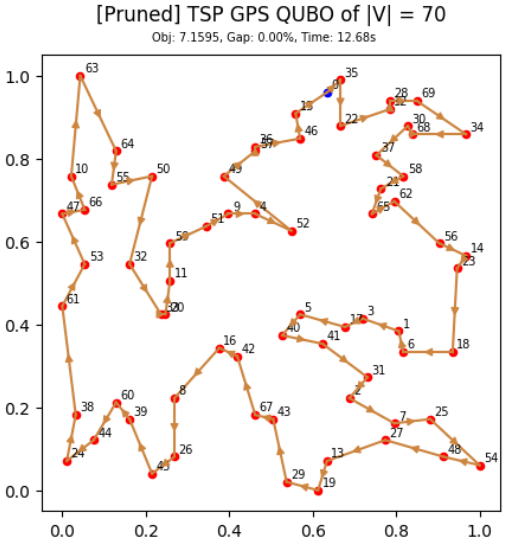

Figure 13: GNNs-pruned GPS QUBO solution for ST70, solved with Gurobi 11.0.1 in 12.68 seconds.

Compared to the optimal TSPLIB95 solution (Figure 14), our pruned solution is slightly different but still feasible for practical use, while offering a substantial computation speedup.

For comparison, applying the conventional TSP QUBO formulation ($x_{i,t}$ variables) without pruning (Figure 15) yielded poor-quality solutions within the same time budget, confirming the value of GNNs-based pruning for accelerating QUBO solving.

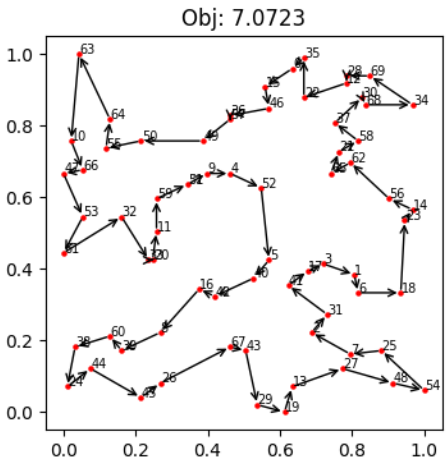

Figure 14: Optimal tour for ST70 from TSPLIB95.

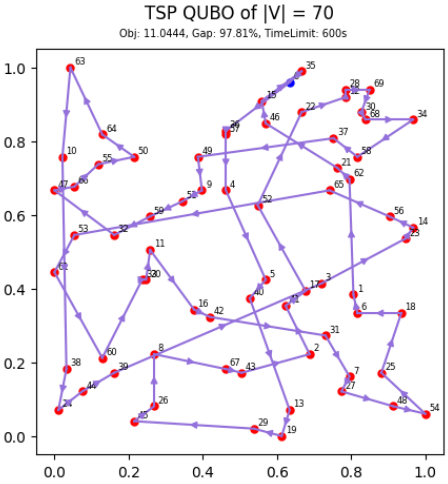

Figure 15: Conventional TSP QUBO result with 600-second limit (poor quality route), demonstrating the benefit of GNNs pruning.

5 Proposed Method

We propose using Graph Neural Networks (GNNs) to reduce the search space of the TSP by pruning edges with low predicted probability before formulating the problem as a QUBO. The objective is to remove edges unlikely to appear in the optimal tour, thereby decreasing the number of binary variables and reducing solver workload.

In this work, we adopt the GPS formulation (Section 3.2) instead of the conventional node-at-time TSP QUBO. The conventional approach, with variables $x_{i,t}$ in $\mathbb{R}^2$, declares $N \times N$ binary variables regardless of graph sparsity. While edge likelihoods can influence the solution, they cannot reduce the declared variable count. In contrast, the GPS formulation defines variables $x_{i,j,r}$ in $\mathbb{R}^3$ on directed edges (i, j) and order index $r \in \{0, 1, 2\}$, requiring $|E| \times 3$ variables. When GNNs prune the edge set to $E^{\text{MLP}}|_{\alpha>\tau}$, the total variable count becomes $|E^{\text{MLP}}|_{\alpha>\tau} \times 3$, where α is the predicted edge probability and τ is the pruning threshold. This directly reduces 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 prior to optimization can deliver both computational and operational benefits.

Building on the promising results in Section 4, we extended our approach to anonymized real business destination data, and stress-tested it by increasing the number of destinations per driver. This allowed us to evaluate both pruning quality and solver performance on the reduced QUBO.

5.1 Model Modifications

5.1.1 GraphSAGE with LSTM

We replaced the original pure trainable-matrix node embeddings with an LSTM layer to improve expressiveness. This allows the model to capture sequential and contextual relationships in logistics graphs, particularly when node order or travel path influences the solution (Figure 16).

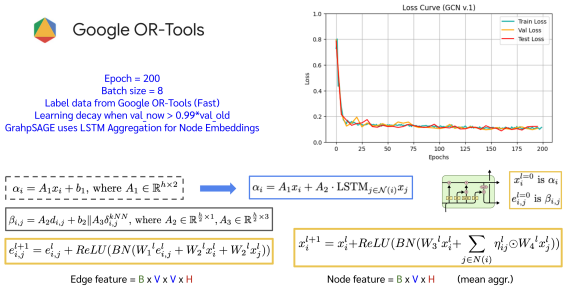

Figure 16: Replacing pure trainable-matrix embeddings with an LSTM layer for improved expressiveness.

5.1.2 Dropout

Dropout layers were inserted after each linear layer in the GCN architecture to reduce overfitting by randomly deactivating neurons during training (Figure 17).

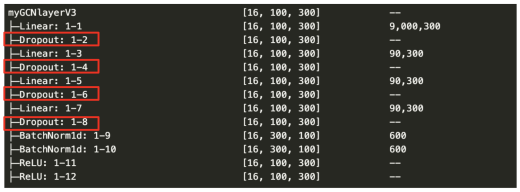

Figure 17: Dropout layers added after each linear layer to improve generalization.

5.1.3 Custom Regularization

The loss function was changed from Negative Log-Likelihood (NLL) to Weighted Cross-Entropy (WCE) to address class imbalance, with an additional penalty term to emphasize false-negative reduction. This encourages higher recall of true positive edges without sacrificing precision (Figure 18).

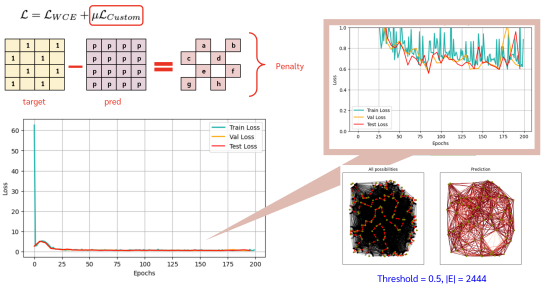

Figure 18: Loss function changed from NLL to WCE with a custom penalty to reduce false negatives.

5.2 Performance Observations

The GNNs maintained efficient pruning performance even for larger instances (Figure 19). Since the model was pre-trained, we measured just only *business logic time* from client request to solution (excluding training time). If the coordinate distribution differed significantly from training data, pruning accuracy could drop, slightly increasing preprocessing time as more edges were reintroduced. In logistics settings where destination sets are relatively stable, training on the full set and applying to daily subgraphs mitigates this issue.

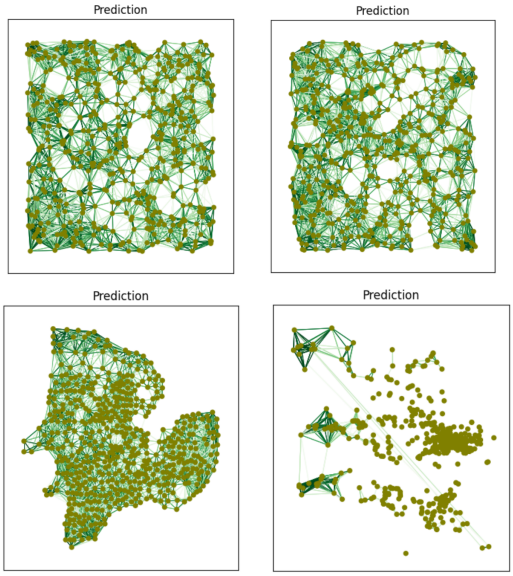

Figure 19: Subgraph of 500 nodes tested with our GNNs. Pruning took seconds; QUBO generation still took ∼3 hours (hardware limitation).

5.3 Subtour Challenges and Solutions

Applying GPS to a sparsified graph occasionally produced multiple subtours, as GPS constraints assume a fully connected graph. Missing edges can lead to disconnected, yet constraint-satisfying, solutions (Figure 20).

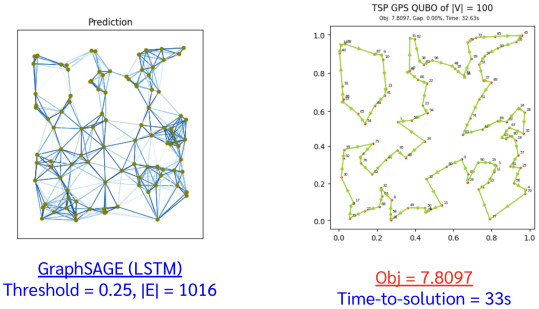

Figure 20: Example of multiple subtours after pruning.

Repair Method. We implemented a post-solve repair process:

1. Detect subtours.
2. Add selected connecting edges.
3. Re-run the solver.

This guarantees a single Hamiltonian cycle but adds overhead (Figure 21).

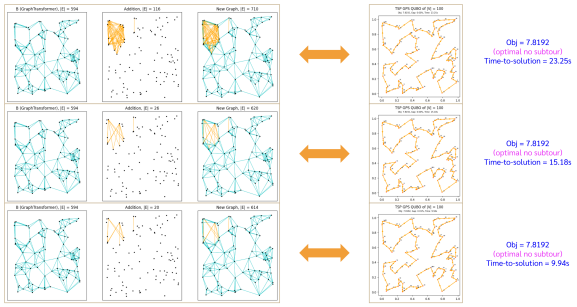

Figure 21: Solution after applying the repair method: subtours are detected, connecting edges reintroduced, and the solver re-run to produce a single Hamiltonian cycle.

Constraint-Based Fix. We introduced a *subtour-join* penalty:

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

weighted by $\gamma > 0$, which discourages edge selections that would prevent proper tour connectivity. This modification eliminated subtours in all tested sparse graphs without densifying them.

To illustrate, Figure 22 shows a proof-of-concept example on a small GPS-TSP instance with $|V| = 8$. The left plot depicts the resulting Hamiltonian cycle, while the three adjacency matrices on the right represent the $r = 0$, $r = 1$, and $r = 2$ variable states after optimization. The bottom-right equation highlights the added penalty term, ensuring that active edges $(i \to j)$ are consistent with the corresponding $x_{j,i,0}$ state.

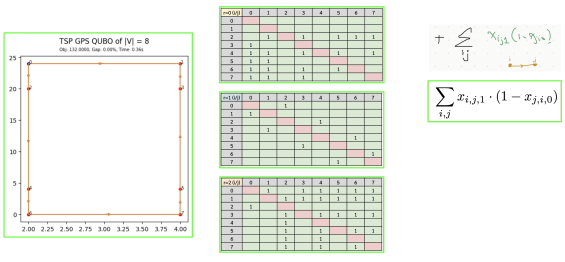

Figure 22: Proof-of-concept for the constraint-based subtour elimination on a GPS-TSP instance with $|V| = 8$. The penalty term $\mathcal{P}_{\mathrm{SJ}}$ prevents disconnected subtours by enforcing connectivity constraints directly in the QUBO.

6 Conclusion

We combined GNN-guided pruning with the GPS QUBO formulation for the TSP, aiming to improve scalability by reducing the edge set before optimization. Unlike the conventional node-at-time formulation, which always declares $N \times N$ binary variables, GPS defines variables on edges and their order. After pruning, the problem size becomes $|E^{\mathrm{MLP}}|_{\alpha>\tau} \times 3$, directly reducing both QUBO size and solver runtime.

A key challenge was the emergence of subtours after pruning. We proposed two complementary strategies:

- A **repair method** that detects and reconnects subtours post-solve.
- A **constraint-based fix** that adds a subtour-join penalty to the QUBO, preventing subtours during optimization.

Experiments showed the constraint-based fix consistently preserved sparsity while eliminating subtours, making it the preferred choice when tuning is possible. The repair method remains a practical fallback when parameter calibration is difficult.

While our approach demonstrates promising scalability and solver efficiency for large-scale routing, this work serves as a foundation rather than a final solution. Future research could explore adaptive penalty strategies, alternative QUBO formulations, and hybrid classical–quantum pipelines, with the goal of enabling real-time deployment once quantum hardware matures.

References

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

[2] 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.

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

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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.