NYC Subway Challenge: A Hierarchical RL Approach to Minimum Spanning Walks

Mason duBoef Sivaraaman Balakrishnan

1 Introduction

The New York Subway Challenge invites participants to explore every subway station as fast as possible. We investigate a simplified version of the problem, modeling it as a reinforcement learning (RL) domain.

The core computational challenge stems from the exponential state space inherent to this problem. If we represent the state as a tuple containing the current station and a binary vector indicating which stations have been visited, we obtain $n \times 2^n$ possible states, where n is the number of stations. With 499 stations in the NYC subway system, this state space is computationally intractable for direct approaches. Our hierarchical decomposition addresses this challenge by partitioning the problem into manageable sub-problems that can be solved exactly.

1.1 Data

Data was compiled from the Metro Transport Authority (MTA). We processed the name, latitude, longitude and connectivity of each station. We used this data to construct an adjacency matrix representation of the subway system. (Appendix Figure 1).

2 Graph Reduction

2.1 Global Connectivity Enforcement

Before any graph reduction, we first verify that the subway network forms a single connected component. We perform a breadth-first search (BFS) to identify the main connected component of each cluster and any disconnected components.

If the graph is disconnected, we inject walkable transfer edges from the MTA's transfers data.

2.2 Merging Nearby Walkable Stations

To reduce the effective state space we collapse pairs of stations that the MTA deemed "walkable" into single representative nodes.

2.3 Corridor Identification and Removal

We perform a further graph reduction step to decrease the problem size while preserving the essential structure. The key insight is that degree-2 nodes represent "corridor" segments that offer no meaningful decision points.

Our reduction algorithm iteratively identifies maximal chains of consecutive degree-2 nodes and replaces them with a single edge directly connecting the chain's two endpoints (Appendix Figure 2). Chains like this are very common in transit systems, with many stations only being serviced by a single line.

2.4 Edge Weighting

While all edges in the original graph have unit weight, simplified edges created during corridor removal are assigned weights equal to the number of original edges they replace. Specifically, if a corridor of k degree-2 nodes is collapsed into a single edge between two hub nodes (degree ≥ 3), that edge receives weight $k + 1$ (accounting for all the hops in the original path). This weight information is later used for the reward function in our MDP formulation, as it allows the agent to accurately account for the true traversal cost of simplified edges.

2.5 Edge Expansion Mapping

To keep track of removed degree-2 nodes we construct an edge expansion map E that enables us to reconstruct the full path after solving the reduced problem.

3 Clustering

MDPs have been applied to the Traveling Salesman Problem (TSP), a similar NP-hard traversal problem. Like in this problem setting, TSP requires that you track which nodes have been visited alongside your current station in your representation for a single state. This requirement makes the state space very large. As such, RL algorithms directly solving TSP become intractable with more than 20 nodes [1].

Even after graph reduction, solving a single MDP with $n \times 2^n$ states remains intractable for $n \approx 100$ nodes. We address this through hierarchical decomposition, partitioning the reduced graph into clusters small enough to solve exactly, then solving a higher-level MDP over cluster sequences.

3.1 Betweenness Centrality Clustering

We employ edge betweenness centrality clustering, following the Girvan-Newman algorithm, to identify natural bottlenecks in the network. Edge betweenness centrality measures how often an edge lies on shortest paths between all pairs of nodes. Edges with high betweenness represent critical bridges between densely connected regions.

The algorithm iteratively computes edge betweenness centrality for all edges, removes the edge with highest betweenness, and checks if the graph has separated into the target number of connected components.

3.2 Cluster Size Constraints

We target clusters of 6 nodes with a maximum of 8 nodes and a minimum of 3 nodes. This constraint ensures that the low-level MDP for each cluster remains tractable for value iteration. If any cluster exceeds the maximum size after initial clustering, we recursively apply edge betweenness splitting to further subdivide it. To avoid the formation of many small clusters, we enforce a minimum cluster size. Undersized clusters are merged into a neighboring cluster with the most boundary connections, provided the merge does not cause the neighboring cluster to exceed the maximum size limit.

3.3 Inter-Cluster Corridor Expansion

After clustering, inter-cluster edges with weight greater than 1 present a challenge: the intermediate corridor nodes must be visited but belong to neither cluster. We address this by expanding such edges: for each inter-cluster corridor edge, we resurrect one intermediate node from the original corridor and assign it to the smaller of the two adjacent clusters. This transforms the weighted inter-cluster edge into two edges—an intra-cluster corridor edge and a unit-weight inter-cluster edge—ensuring all inter-cluster transitions have weight 1 and corridor traversal is handled within the low-level MDPs.

3.4 Boundary Nodes

After clustering, we identify *boundary nodes*—nodes that have edges connecting to nodes in different clusters. These boundary nodes serve as entry and exit points for cluster traversal and are essential for the high-level MDP formulation.

3.5 Cluster Connectivity Validation

After clustering and boundary identification, we verify that each cluster forms a connected subgraph. We run BFS from every node in a cluster to identify disconnected components. Disconnected components are then reassigned to a neighboring cluster.

4 Low-Level MDP Formulation

Each cluster is modeled as an independent MDP that can be solved exactly using value iteration. The goal within each cluster is to compute the minimal spanning walk from every boundary node to every other boundary node.

4.1 State Space ($\mathcal{S}$)

A state represents the current station, a record of previously visited stations, and a record of which corridor edges have been traversed:

$$s = (\text{currentNode}, \text{visitedMask}, \text{corridorEdgesMask}) \tag{1}$$

where currentNode $\in \{0, 1, \ldots, n_c - 1\}$ indexes nodes within the cluster, visitedMask $\in \{0,1\}^{n_c}$ indicates which nodes have been visited, and corridorEdgesMask $\in \{0,1\}^{e_c}$ indicates which corridor edges (edges with weight > 1) have been traversed, where e_c is the number of such edges in the cluster.

The state space size for a cluster with n_c nodes and e_c corridor edges is $n_c \times 2^{n_c} \times 2^{e_c}$. This extended state space ensures the agent must traverse all corridor edges to reach a terminal state.

4.2 Action Space ($\mathcal{A}$)

At each station within the cluster, the agent may travel to any neighboring station within the same cluster:

$$\mathcal{A}(s) = \{v' : (v, v') \in E_c\} \tag{2}$$

where v is the current node and E_c is the set of edges in the cluster subgraph.

4.3 Transition Dynamics ($P(s'|s,a)$)

Transitions are deterministic. Taking action a (moving to neighbor v') from state $s = (v, \text{visitedMask}, \text{corridorMask})$ updates both the visited mask and, if the traversed edge is a corridor edge, the corridor edges mask.

4.4 Reward Function ($R(s, a, s')$)

The reward function encourages efficient exploration while penalizing unnecessary steps:

$$R(s, a, s') = R_{\text{step}}(s, a) + R_{\text{visit}}(s, a) + R_{\text{corridor}}(s, a) + R_{\text{complete}}(s') \quad (3)$$

where:

- $R_{\text{step}}(s, a) = -1 \times w(v, a)$: A per-step penalty scaled by the edge weight $w(v, a)$.
- $R_{\text{visit}}(s, a) = +2 \times w(v, a)$ if node a was previously unvisited, 0 otherwise.
- $R_{\text{corridor}}(s, a)$: If traversing a corridor edge for the first time and the destination was already visited, gives $+2 \times (w - 1)$ for the intermediate nodes.
- $R_{\text{complete}}(s') = +100$ if s' is a terminal state.

We do not discount rewards over time ($\gamma = 1.0$) since our reward structure already incentivizes faster/shorter routes.

4.5 Terminal States

A state is terminal if and only if all nodes in the cluster have been visited, all corridor edges have been traversed, and the agent is currently at a designated boundary node. When solving for a specific target exit boundary node b_{target}, the terminal condition additionally requires currentNode $= b_{\text{target}}$.

4.6 Value Iteration for Low-Level MDPs

We solve each cluster MDP using value iteration with threshold $\theta = 10^{-6}$. The optimal policy is extracted as:

$$\pi(s) = \arg\max_{a \in \mathcal{A}(s)} \left[R(s, a, s') + \gamma V(s') \right] \quad (4)$$

We do not employ multiple algorithms. Given that we understand the rewards and transition dynamics of our environment, dynamic programming methods are the ideal course of action. Other algorithms like Q-learning, SARSA or even TD-methods which generate trajectories are not a good fit for this setting. We discussed this with Prof. da Silva in class. He told us that applying value iteration to this problem with novel state abstraction, would be sufficient for the project.

4.7 Pre-computing Cluster Policies

For each cluster c with boundary nodes B_c, we solve the MDP once for each target exit boundary node $b_{\text{exit}} \in B_c$. For each solution, we extract optimal paths from every entry boundary node $b_{\text{entry}} \in B_c$ to the target exit. These pre-computed policies serve as "macro-actions" for the high-level MDP.

5 High-Level MDP Formulation

The high-level MDP operates over clusters rather than individual stations, using the pre-computed cluster policies to determine transition costs.

5.1 State Space

A high-level state consists of:

$$s_H = (\text{currentCluster}, \text{currentBoundaryNode}, \text{visitedClustersMask}) \tag{5}$$

where currentCluster identifies which cluster the agent is in, currentBoundaryNode is the specific boundary node within that cluster, and visitedClustersMask $\in \{0, 1\}^K$ indicates which clusters have been fully traversed.

5.2 Action Space

Actions at the high level represent transitions to adjacent unvisited clusters with a specified target exit:

$$\mathcal{A}_H(s_H) = \{(c', b_{\text{exit}}) : c' \in \text{Adj}(c) \wedge \neg\text{visited}[c'] \wedge b_{\text{exit}} \in B_{c'}\} \tag{6}$$

This formulation allows the high-level policy to plan not just which cluster to visit next, but also which exit point to target within that cluster.

5.3 Reward Function

The reward for a high-level transition is the total reward from executing the corresponding low-level policy through the target cluster. Critically, if revisiting a cluster the agent receives a flat penalty of -20.

5.4 Terminal States

A high-level state is terminal if all clusters have been visited.

5.5 Path Extraction and Expansion

Given the high-level policy, we extract the complete route by following the policy to determine the sequence of clusters and target exits, retrieving pre-computed low-level paths for each cluster transition, and finally expanding each edge using the edge expansion map to recover the full path through all original stations.

6 Results

Our hierarchical MDP approach successfully computed a route through the NYC subway system.

Metric	**Value**
Original stations	499
Stations after walkable collapse	311
Stations after corridor removal	100
Number of clusters	16
Cluster size range	6–11 nodes
Total reward	1776.0
Expanded path length	464 steps
Unique nodes visited	311
Number of revisits	153
Revisit ratio	33.0%

Table 1: Summary of results from hierarchical MDP solution.

The full route through all 311 collapsed stations can be viewed here:
https://drive.google.com/file/d/1tvJTfiFLiSDpOXw8Q-Q1VnnNsm-iBgsn/view?usp=sharing

Seeing as we are performing value iteration to deduce an optimal policy we are not running episodes and thus do not have learning curves to display. We are iteratively calculating the optimal policy with value iteration.

7 Discussion

7.1 Interpretation of Results

This clustering and hierarchical MDP allows for enables the solving of approximate minimum spanning walks. These results would not be tractable using a direct MDP or using simply combinatorial approach. While a 33% revisit rate implies that there is room for improvement. Our formulation allows for routes solutions to be principally solved for using dynamic programming methods.

7.2 Assumptions

Our formulation relies on two simplifying assumptions: (1) **Uniform travel times within edge weights:** We assume uniform travel time per hop, though in reality travel times vary based on distance between stations and train schedules. (2) **Static graph:** The network topology is fixed; we do not account for temporary closures or service changes.

7.3 Limitations and Future Work

The 33.0% revisit ratio indicates room for improvement. While some revisits are unavoidable due to the tree-like structure of certain subway branches, optimizing the cluster sequencing and boundary node selection could potentially reduce unnecessary backtracking.

In the future, we would like to construct a more sophisticated system for state abstraction. We could create an embedding system, mapping all of the $499 \cdot 2^{499}$ to a smaller discrete state space based on common graphical features. We would likely use a Graph Neural Network (GNN) to do so. This would allow us to shrink the state space to something tractile without relying on a hierarchical structure and inexact clustering.

8 Group Member Contributions

Mason - Formulation and implementation of high and low level MDPs, value iteration, betweenness clustering, corridor station reduction. Sivaraaman - Connectivity validation of all clusters and the overall graph, data gathering and processing, merging nearby stations based on walkability, visualizations.

9 References

[1] 'A dynamic programming approach to sequencing problems', Michael Held and Richard M. Karp, Journal for the Society for Industrial and Applied Mathematics 1:10. 1962

10 Appendix

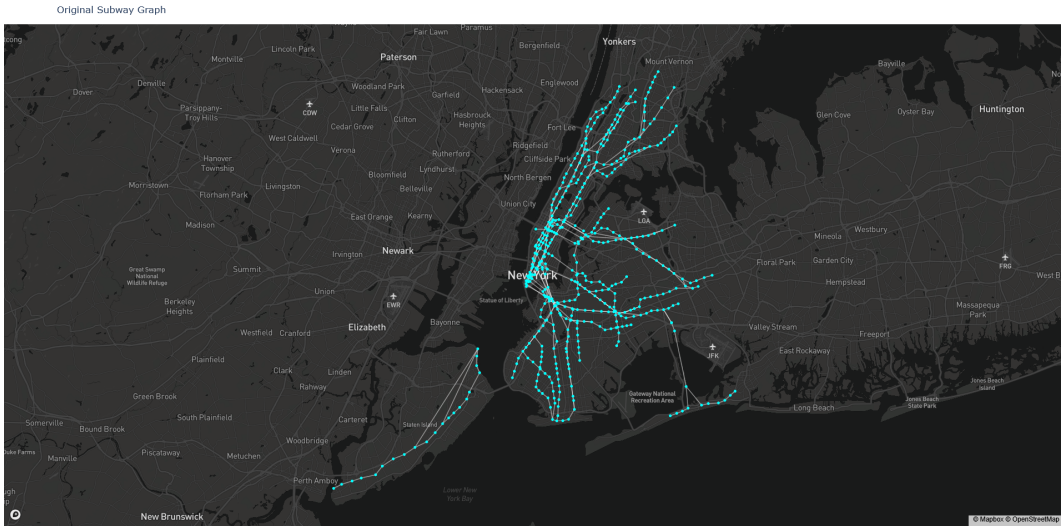

Figure 1: Original connectivity of all 499 nodes

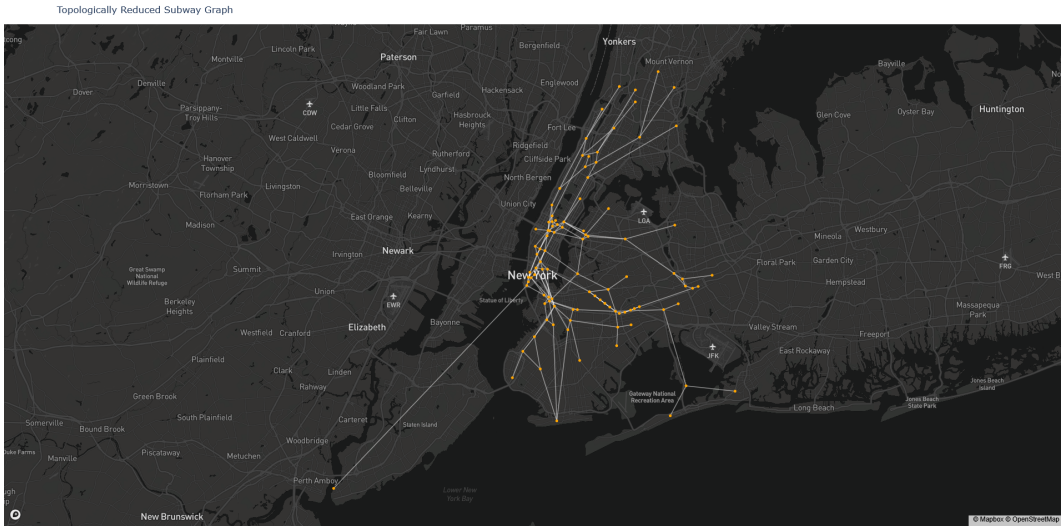

Figure 2: Reduced nodes

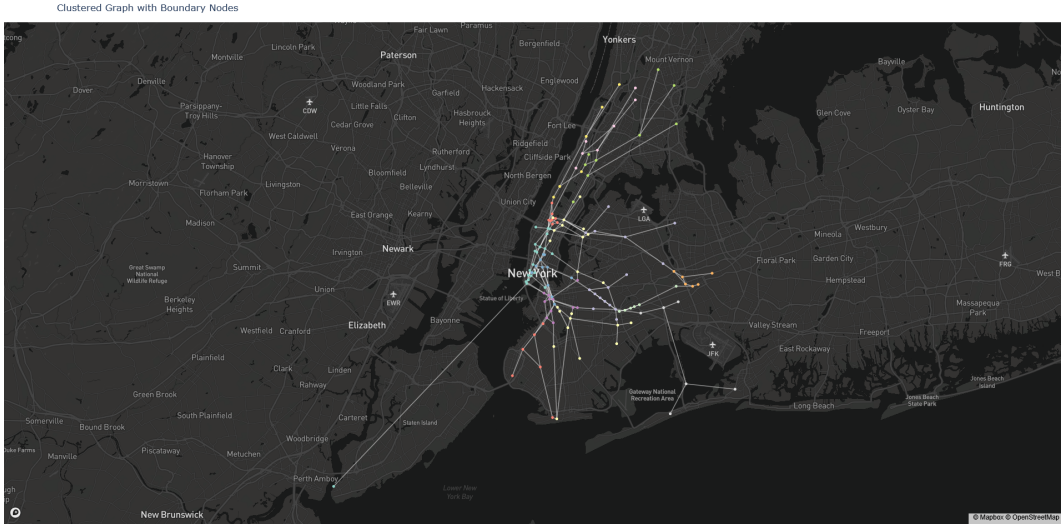

Figure 3: Reduced nodes colored by cluster