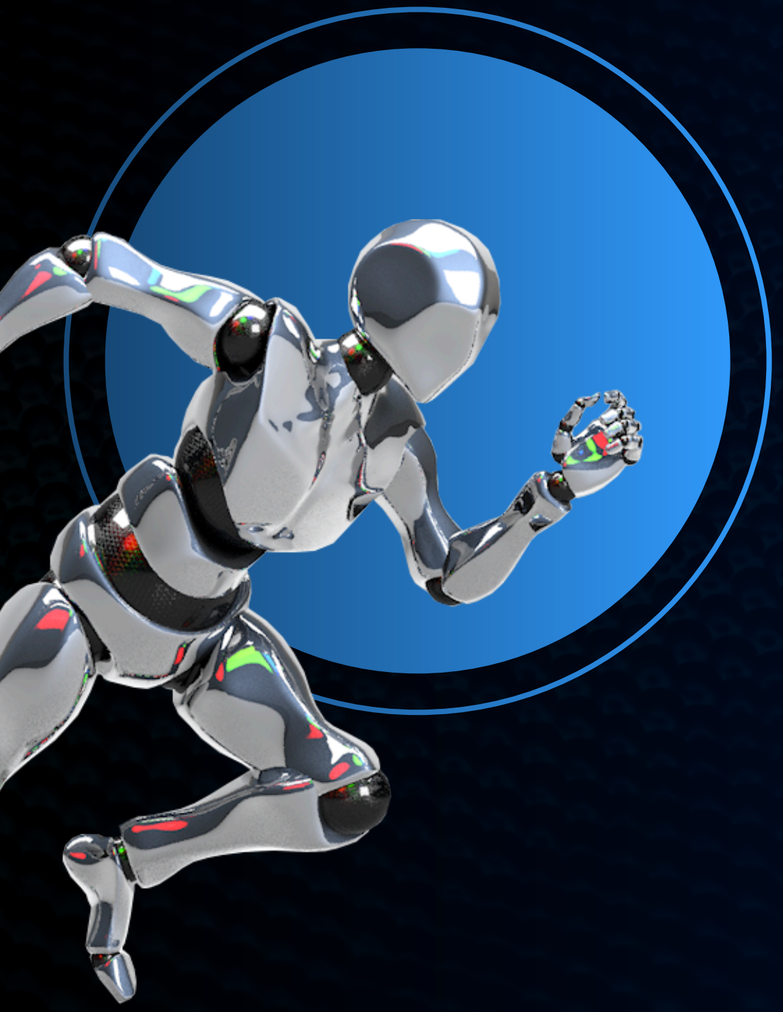


Title

Shortest Path Finding in Dynamic Graphs using Q-learning and Deep Q-Networks

Team

Pratik Rana, Krishna Sharma, Rahul Kumar

Problem Being Solved

The Challenge of Dynamic Shortest Paths

Traditional Problem: Find path $P = (v_0, v_1, ..., v_k)$ from source s to destination d that minimizes total cost $C(P) = \sum^{k-1}_{i=0} w(v_i, v_{i+1})$

Limitation of Traditional Algorithms

- Dijkstra's and A* require complete recalculation when graph changes
- D* Lite needs explicit replanning when optimal path is affected
- Computationally expensive for graphs with frequent changes

Dynamic Environment Challenge:

- Edge weights w(u,v) vary temporally (e.g., traffic congestion, network load)
- Need for adaptive solutions that don't require complete recalculation

RL Formulation

Environment & Agent:

- **Environment**: Dynamic graph G = (V, E, W(t)) with time-dependent edge weights
- **Agent**: Entity navigating the graph from start to goal

State Representations:

- Standard QL: s = (current_node, goal_node)
- HQL: s = (current_node, goal_node, region_id)
- DQN: s = concat(one_hot(current), one_hot(goal), adj(current))

Reward Functions:

- Large +ve reward for reaching the goal
- -w(current, next) for valid steps
- $-P_{invali}$ for invalid moves

Action Space:

- Standard QL/DQN: Adjacent nodes only
- HQL: Adjacent nodes (region-specific)
- Expanded QL: All nodes in the graph

Policy:

- Policy: $\pi(s) = \text{argmax}_{a \in A(s)} Q(s,a)$

Current Methods

Method	Strength	Weakness
D* lite Algorithm	Reuses prior computations for fast updates	Memory-intensive for large graphs, Struggles with frequent edge changes
LPA(Lifelong Planning A*)	Heuristic-driven, balances optimality and speed	Memory-intensive for large graphs, Struggles with frequent edge changes
Multi-Agent Reinforcement Learning (MARL)	Decentralized Adaptation, Superior Performance in Dynamic Settings	High Complexity, Communication Overhead

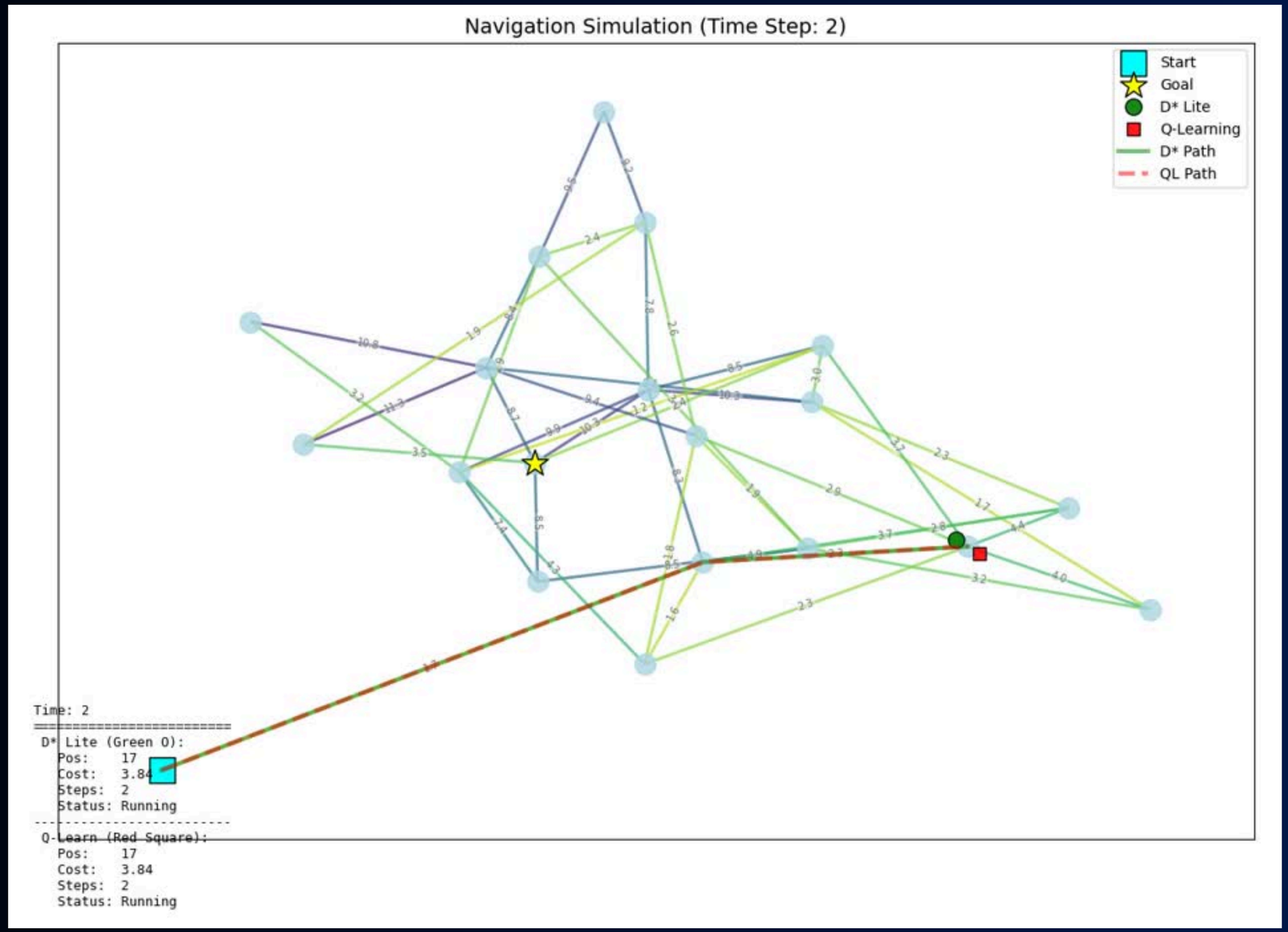
Dynamic Graph generation:

- Graph Types: Erdős–Rényi graphs with guaranteed connectivity
- Scalability: Supports small to large graphs (10–10k nodes)
- Busy Node Pattern: Deterministic selection using index intervals (e.g., every 5th node)
- Edge Weights:
  - Initialized based on node "busyness"
  - Time-varying via sinusoidal model at each time step
  - weight(t) = base + variation × sin(period × t)
- Dynamic Behavior: Simulates load changes, congestion, or traffic over time
- Reproducibility: Fully seed-controlled for deterministic experiments

Evaluation Metrics:

- Path cost, computation time, Path Steps, optimal Path

Path Finding Visualization Demo for Dynamic Graph



Our Methodology:

1. **Q-Learning Approaches:**
    a. Standard
    b. Expanded Action Space
    c. Hierarchical Q-Learning

2. **Deep Q-Network (DQN)**

Implement dynamic graph generator (Python)

Week 1 - 2

Training

Week 3 - 4

Benchmark against D*/LPA* (optimality, speed)

Week 5 - 6

Our Methodology 1 → Q-Learning Approaches

1.a: Standard Q-Learning:

- $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- ε-greedy policy with exponential decay

- s = (current_node, goal_node)
- action space = Adjacent nodes only

1.aa: Adaptive Episodes

- Nepisodes = BaseEpisodes × (N/Nmin)
- Scales training effort with graph complexity
- Enables better policy convergence for larger graphs

1.b: Expanded Action Space Q-Learning

- approach: Action space A(s) = V (all nodes)
- Larger action space (from O(d) to O(N))
- Penalty-based learning of graph connectivity
- Required significantly more training episodes

Our Methodology 1.c → Hierarchical Q-Learning

Core Idea

- Divide-and-conquer approach to handle larger graphs
- Graph automatically partitioned into k regions using BFS-based strategy
- Border nodes connecting different regions identified

Architecture

- Independent Q-Learning agent for each region
- Each agent specializes in navigating its local territory
- Collaborative goal: Find global path by efficient region transitions

HQL: s = (current_node, goal_node, region_id)

Learning Mechanism

- Regional Decision Making: Only the agent for current region selects next action
- Hierarchical Reward Structure:
  - Base: -w(current, next) for movement cost
  - Terminal: +100 for reaching final goal
  - Coordination Bonus: Small reward for crossing to a region closer to goal

Our Methodology 2 → Deep Q-Network (DQN)

Enhanced State Representation:

- s_vec = concat(one_hot(current), one_hot(goal), adj(current))
- Provides location, goal, and binary neighbour connectivity information

Network Architecture:

- Input: 3N vector
- 2 hidden layers (128 neurons each) with ReLU activation
- Output: Q-values for N possible target nodes

Training

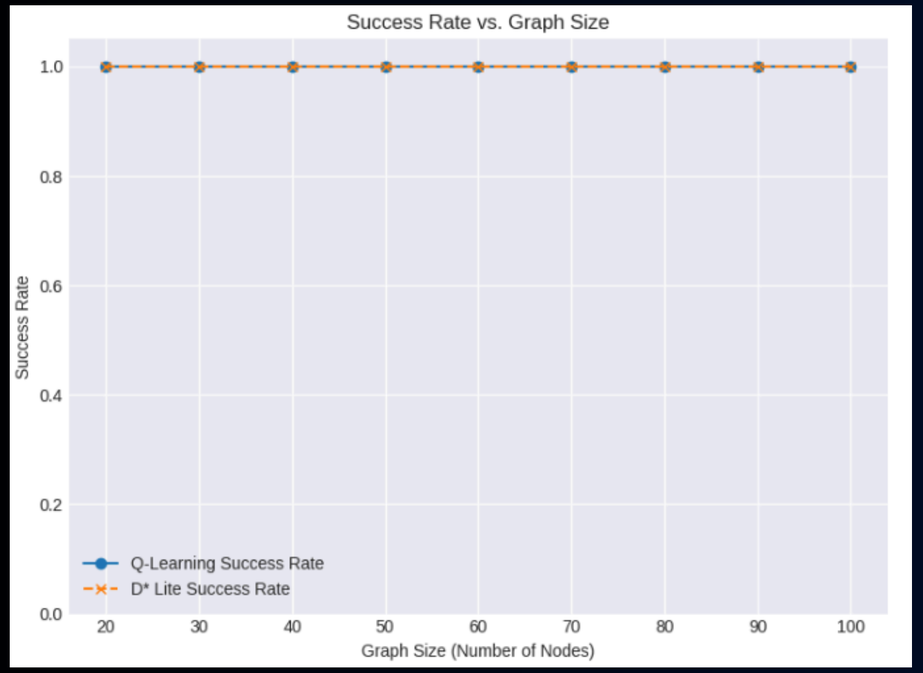
- Experience replay buffer
- Target network for stable training
- Action masking for valid neighbors only
- Smooth L1 loss function

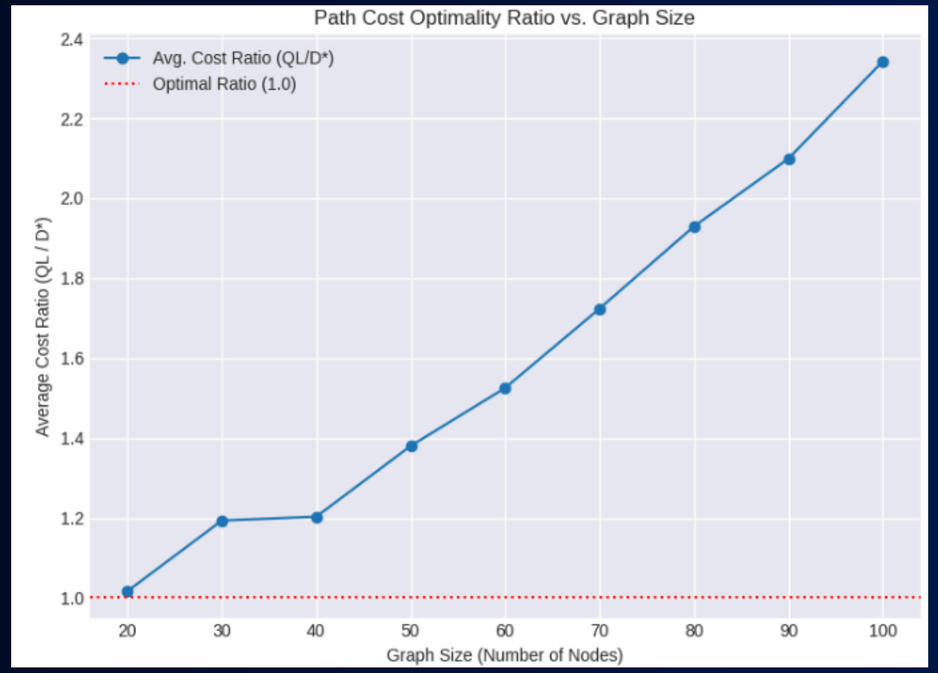
Action Selection & Reward Shaping

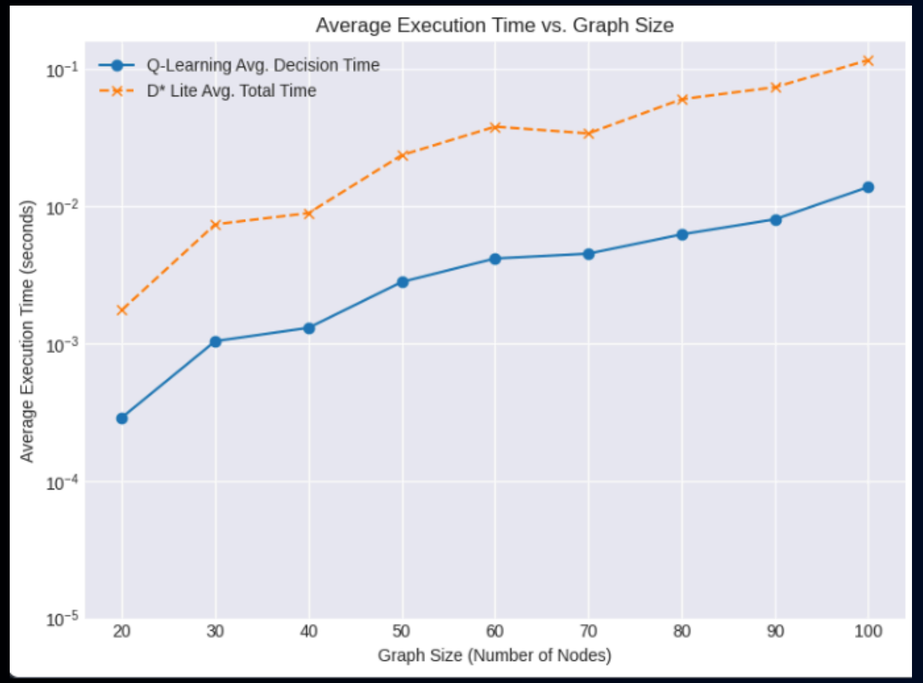
- Agent selects next hop from valid neighbors.
- Epsilon-greedy (train) / Greedy (eval) over masked Q-values.
- Scaled Edge Cost, +Goal Bonus, -Loop/Step Penalties

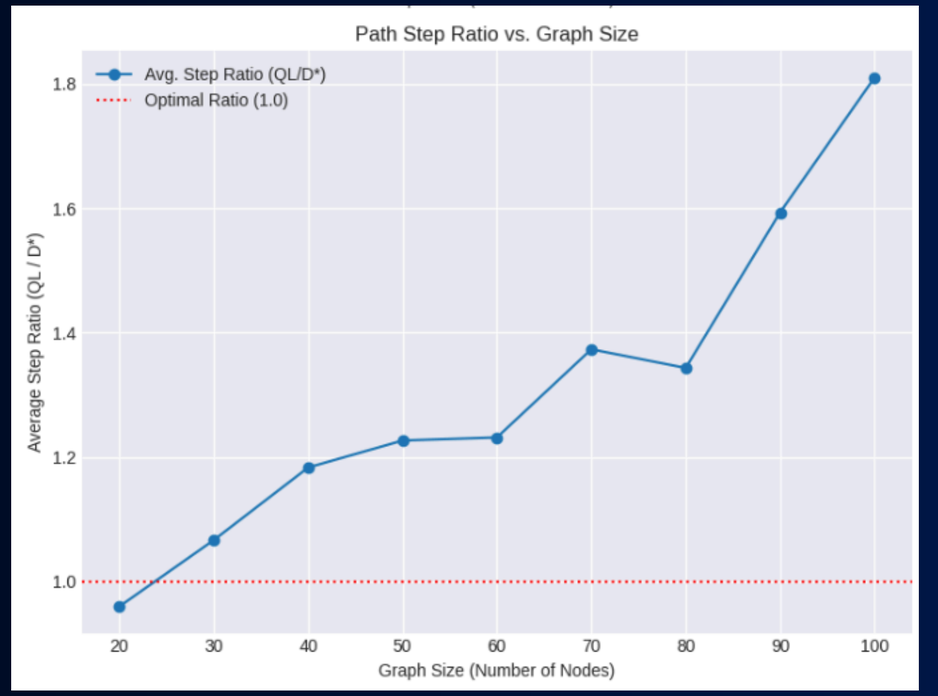
Results

Results – Q-Learning Performance





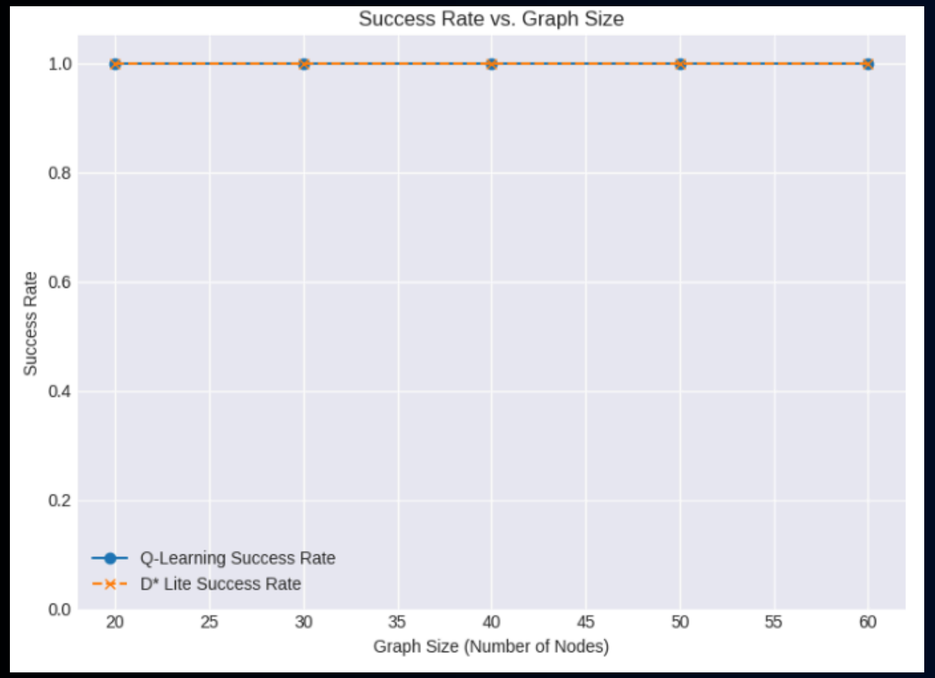


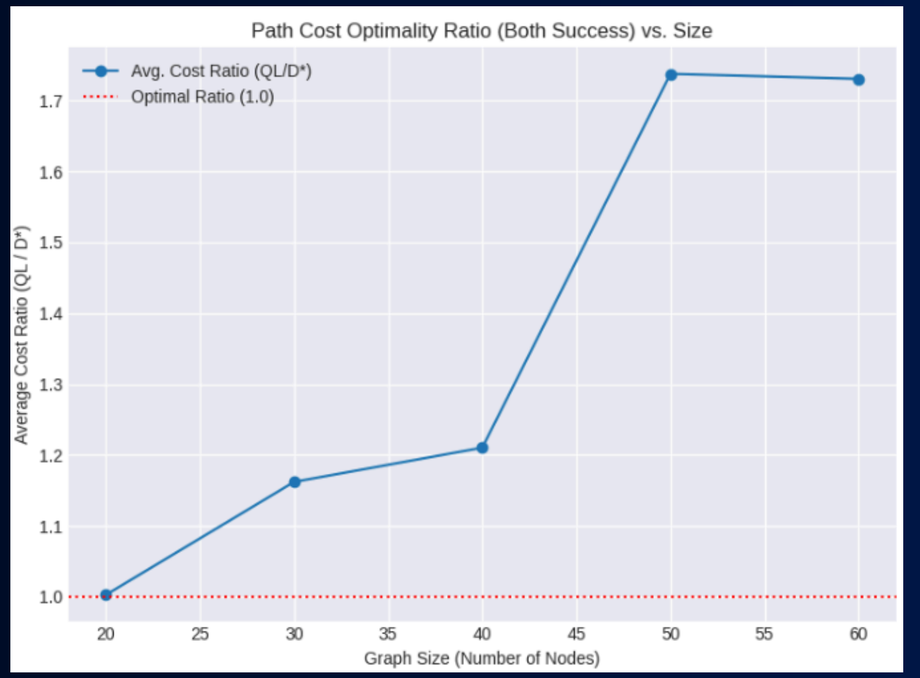


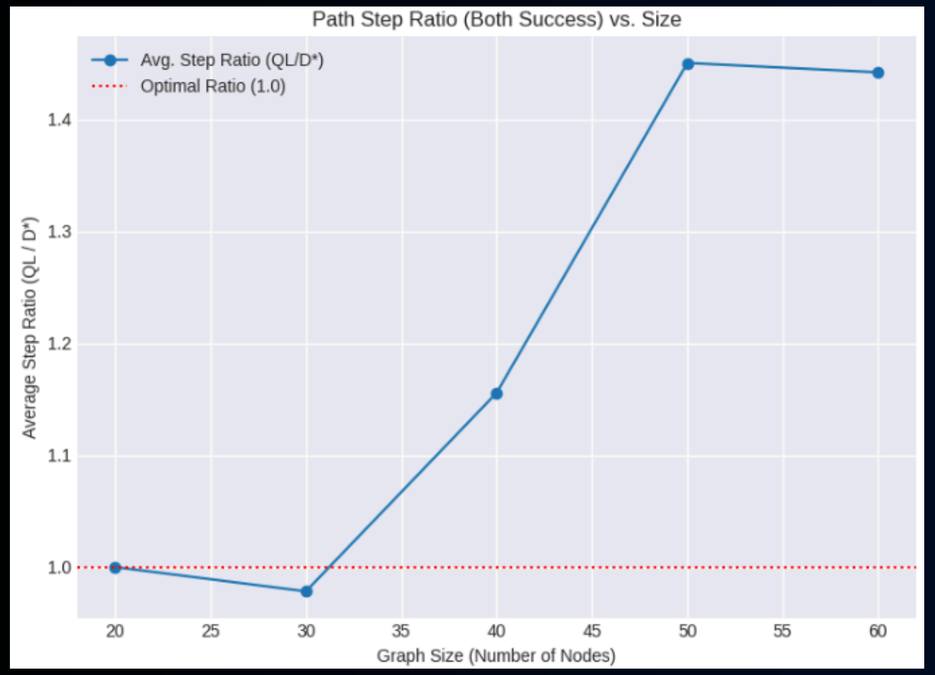
Standard Q-Learning Performance Overview

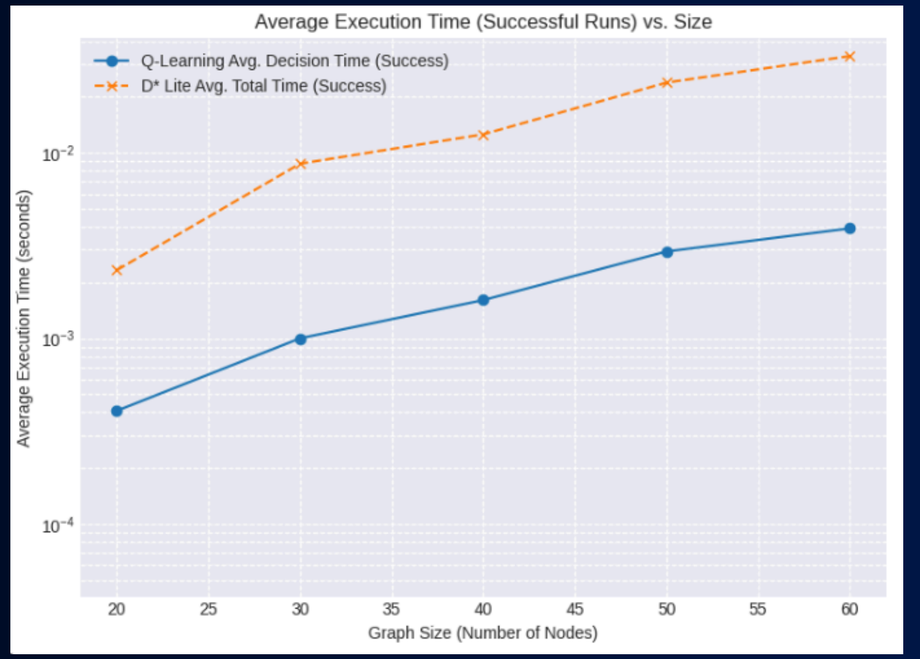
- **Success Rate:**
  - 100% across all graph sizes (20-100 nodes)
  - Matches D* Lite's reliability in finding valid paths
- **Path Optimality:**
  - Small graphs (size 20-30): Near-optimal paths (ratio ~1.0-1.2)
  - Medium graphs (size 40-60): Moderate deviation (ratio ~1.3-1.8)
  - Large graphs (size 80-100): Significant deviation (ratio ~2.0-2.5)
- **Training Strategy Comparison:**
  - Fixed Episodes: Less effective for larger graphs
  - Adaptive Episodes: Superior performance on larger graphs
    - Formula: $N_{episodes} = BaseEpisodes \times (N/N_{min})$
    - Size 100 improvements: Cost ratio ~2.34 vs ~2.50; Step ratio ~1.81 vs ~1.89
- **Key Advantage:**
  - Execution time remains consistently low (milliseconds)
  - Orders of magnitude faster than D* Lite replanning

Expanded Q learning Performance





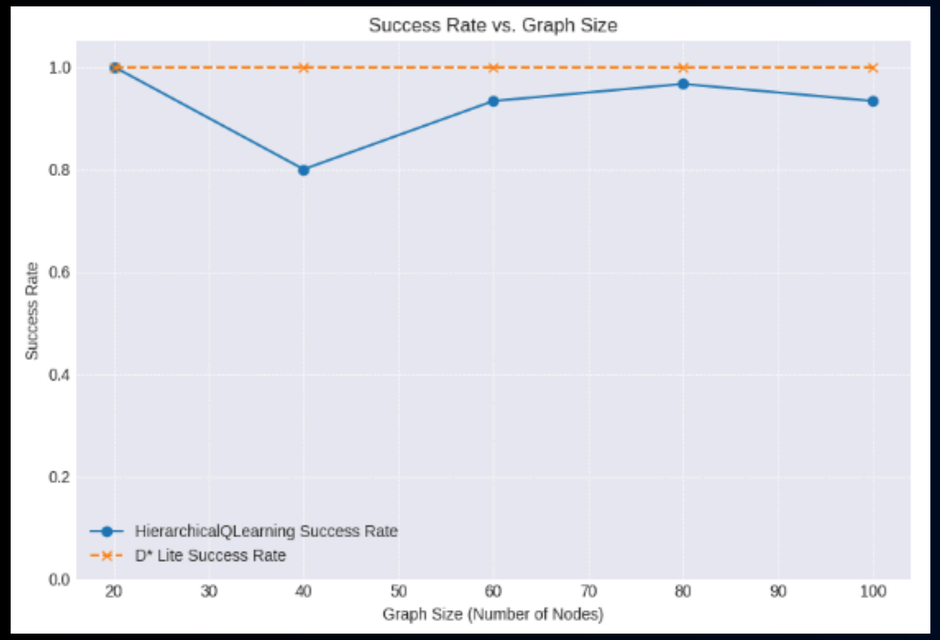


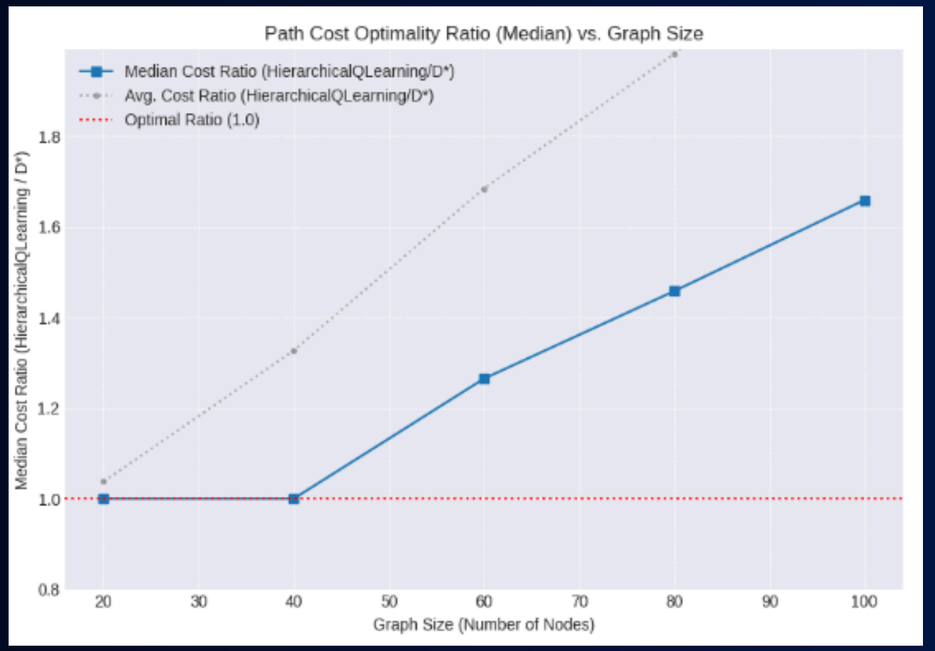


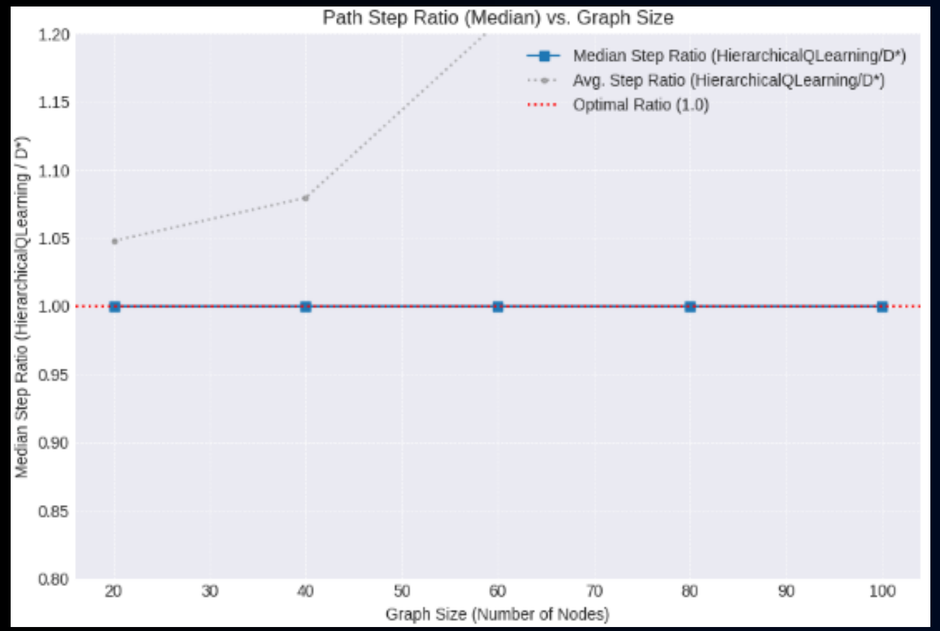
Expanded Q-Learning Performance Overview

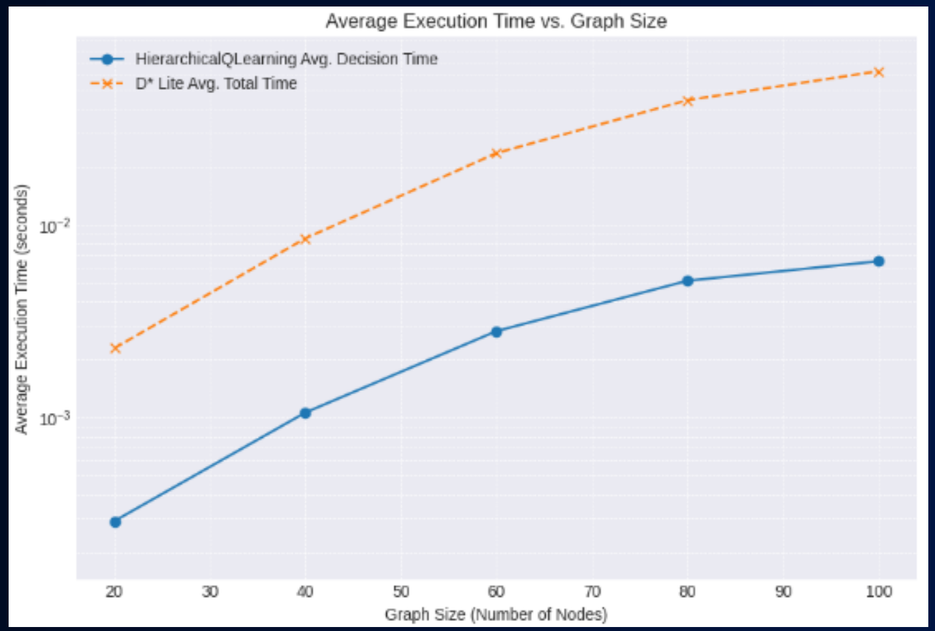
- **Core Idea:**
  - Action space = All nodes in graph (not just adjacent)
  - Learns graph connectivity implicitly through penalties
- **Success Rate:**
  - 100% across tested graph sizes (20-60 nodes)
- **Path Optimality:**
  - Consistently worse than Standard Q-Learning
  - Cost ratio increases more rapidly with graph size
  - Size 60: Cost ratio ~1.73 vs. ~1.5 for Standard QL
- **Training Challenges:**
  - Lower average rewards during training
  - Required significantly more episodes (120,000 for size 60)
  - Longer training times (~1053s for size 60)
  - Difficulty in avoiding invalid move penalties
- **Inference Performance:**
  - Decision time comparable to Standard QL
  - Still significantly faster than D* Lite

Results – Hierarchical Q-Learning Performance





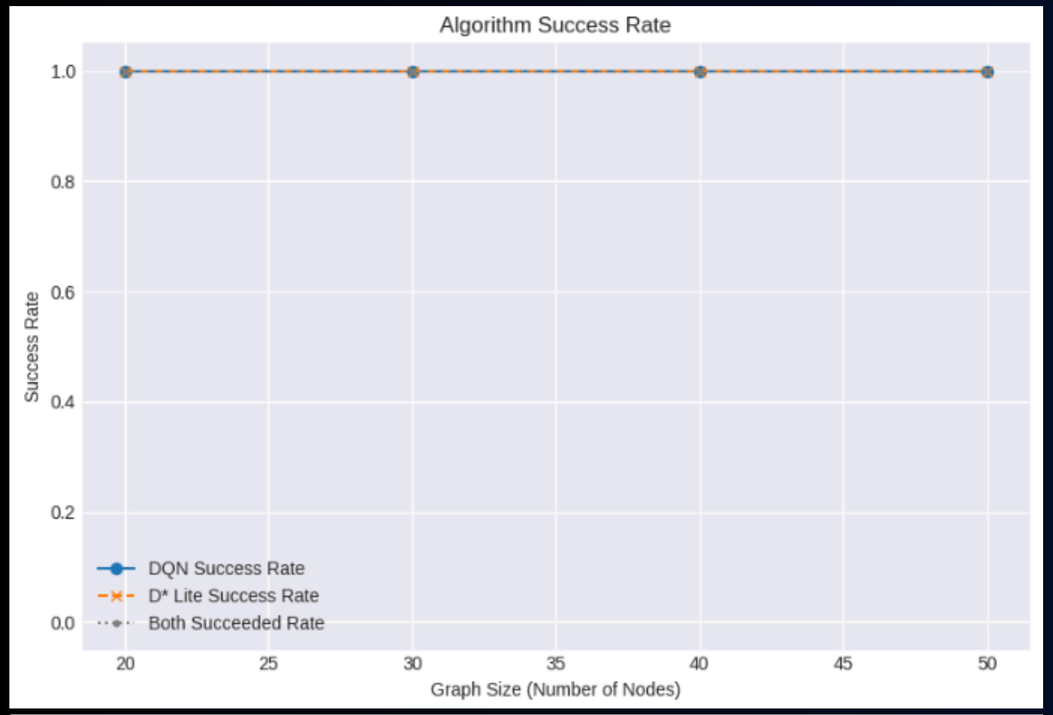


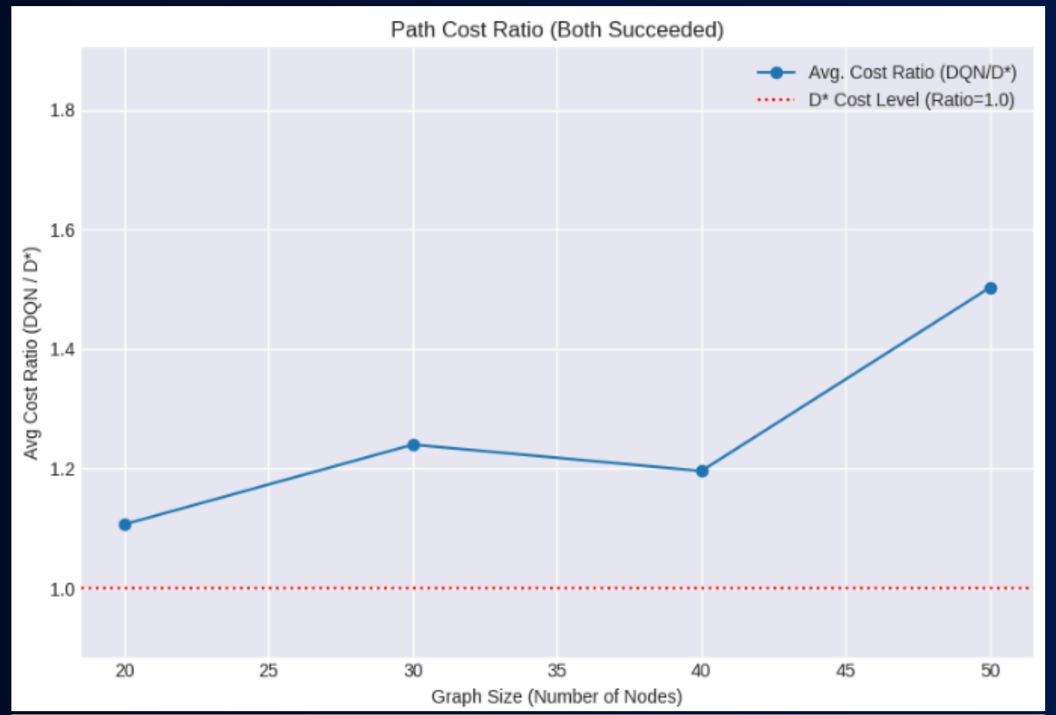


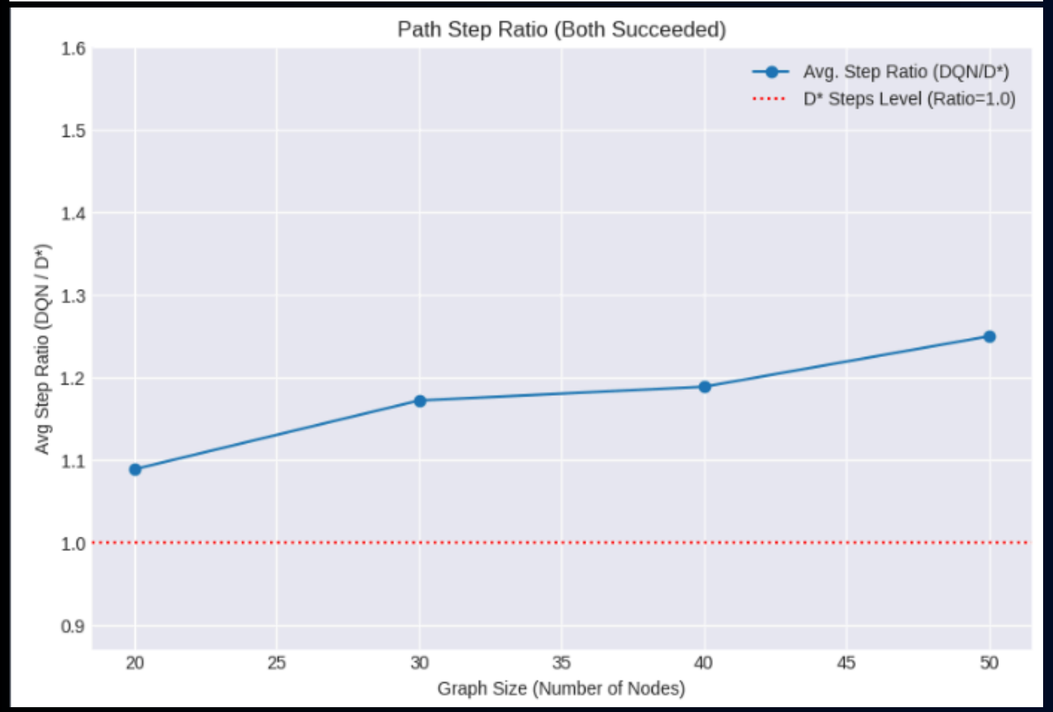
Q Learning with Graph partionting

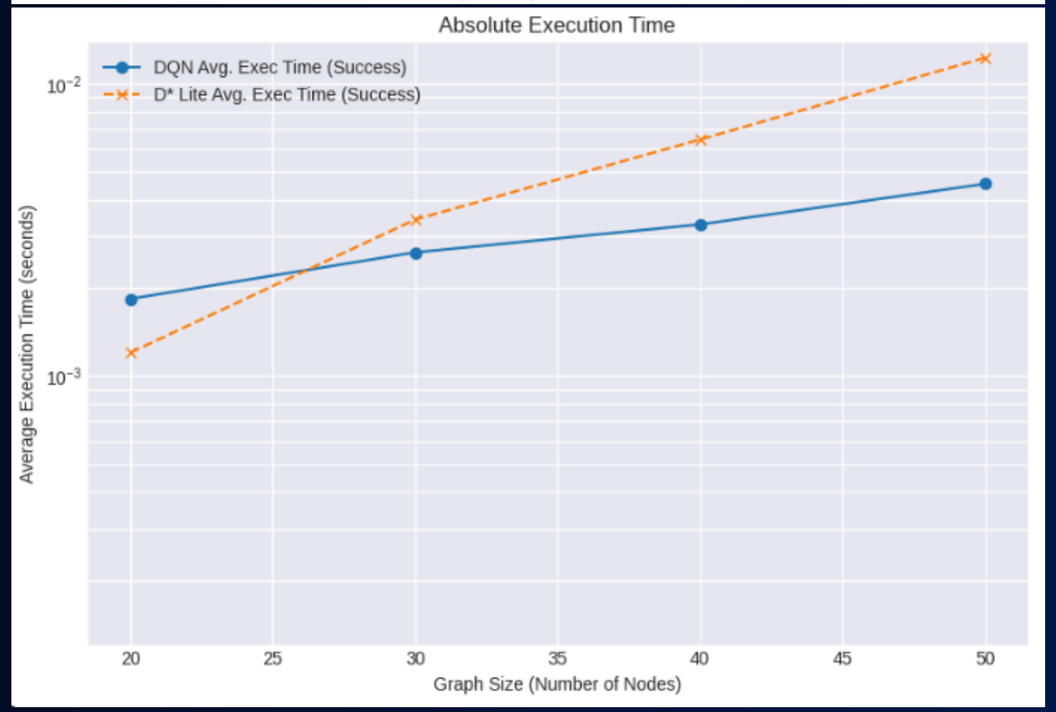
- **Success Rate:**
  - 95%+ across most graph sizes
  - Slightly lower than D* Lite's perfect 100%
  - Reliable for practical applications
- **Path Optimality:**
  - Small graphs (size 20-30): Near-optimal paths (Ratio ≈ 1.0)
  - Large graphs (size 100): Moderately suboptimal (Median Cost Ratio ≈ 1.65)
  - Path quality degrades more gracefully with size than flat Q-Learning
- **Path Structure:**
  - Step ratio near 1.0 across all sizes
  - HQL finds paths with similar number of hops as D* Lite
  - Indicates good route structure despite higher cost
- **Execution Efficiency:**
  - Major Advantage: Decision time 7-9× faster than D* Lite (Time Ratio 0.11-0.14)
  - Better scalability with increasing graph size
  - Minimal per-step decision overhead

Results – DQN Performance









DQN Performance Overview

- **Success Rate:**
  - Nearly 100% across tested sizes (20-50 nodes)
  - 100% for sizes 20, 30, 50; 96.7% for size 40
- **Path Optimality:**
  - Cost ratio (DQN/D*): Increases from ~1.2 to ~1.42 with size
  - Step ratio: Relatively stable between 1.1-1.25
  - More consistent performance with increasing graph size compared to Q-Learning
- **Path Characteristics:**
  - Path match rate decreases with graph size (57% to 37%) - not necessarily bad
  - DQN finds alternative valid paths different from D* Lite
- **Execution Efficiency:**
  - Forward pass extremely fast (2-4ms)
  - Consistent performance regardless of graph size
  - Comparable to Q-Learning table lookup
  - Significantly faster than D* Lite replanning

Metric	Standard Q-Learning	Expanded Q-Learning	Hierarchical Q-Learning	DQN	D* Lite
Success Rate	100%	100%	95%+	~100%	100%
Path Optimality	Good → Poor (1.0 → 2.5×)	Moderate → Poor (1.2 → 1.7×)	Good → Moderate (1.0 → 1.65×)	Moderate (1.2 → 1.4×)	Optimal (Upperbound)
Path Structure	Degrading with size	Similar to Std. QL	Consistent (step ratio ~1.0)	Consistent across sizes	Optimal (Upperbound)
Training Time	Increases with graph size (adaptive)	Highest (large action space)	Moderate (distributed)	Moderate, better scaling	N/A
Decision Time	Very fast (ms)	Very fast (ms)	Very fast (7-9× faster than D*)	Very fast (2-4ms)	Slow (increases with graph size)
Scalability	Limited by table size	Most limited	Good (distributed learning)	Supposedly good for large graphs but requires tuning	Limited by replanning cost

Key Contributions

Novel RL Architectures for Dynamic Graphs:

- Standard Q-Learning with Adaptive Episodes: Systematic scaling of training effort with graph complexity
- Expanded Action Space Q-Learning: Innovative approach allowing any node selection as action
- Hierarchical Q-Learning: Divide-and-conquer approach with regional agents and border-crossing coordination
- DQN: Leveraged function approximation for consistent path optimality and fast decisions across varying graph sizes.

Training Innovations:

- Regional Specialization: Distributed learning that scales better to complex graph structures
- Border-Node Reward System: Novel mechanism for coordinating regional policies toward global objectives

Comprehensive Evaluation Framework:

- Rigorous benchmark across multiple metrics: success rate, path cost, steps, execution time
- Analysis of scaling behavior with increasing graph complexity
- Consistent dynamic graph simulation for fair comparison

Real-World Application Insights:

- Trade-off analysis between path optimality and decision speed
- Practical guidelines for selecting appropriate method based on application constraints
- Orders-of-magnitude faster decision-making compared to traditional replanning approaches

Reflections & Learnings

Trade-offs in RL Approaches:

- **Higher offline training costs vs. faster online decisions**: Initial investment in learning yields significant runtime benefits
- **Optimality vs. Speed**: All RL methods prioritize rapid decisions over perfect optimality
- **Scalability vs. Exactness**: Function approximation and hierarchical approaches scale better but introduce approximation error

Method Specific Insights:

- **Standard Q-Learning**: A Simple tabular approach works remarkably well in constrained spaces
- **Expanded Action Space**: A Larger action space dramatically increases learning difficulty
- **DQN**: Neural representation provides more graceful performance degradation with scale
- **Hierarchical Q-Learning**: Decomposition manages complexity with minimal coordination overhead

Practical Applications

- **Real-time Navigation**: RL approaches enable millisecond decisions critical for autonomous vehicles
- **Network Routing**: Fast adaptation to changing conditions without complete recalculation
- **Resource Allocation**: Quick reallocation in dynamic environments with minimal computational burden

Thank You