Final Project: Artificial Intelligence in Digital Medicine

Finding Optimal Routes for a Smart Robot-Ambulance in a Crowded Town with Obstacles

(A Comparative Study of Dijkstra-on-Grid, A, and Probabilistic Roadmap (PRM))*

Project Topic: Routing of smart robot-ambulance in a crowded town with obstacles

Authors:

Bar Koldanov (207702416)

Orel Cohen (214266819)

Lecturer: Prof. Eugene Levner

Institution: Holon Institute of Technology (HIT)

Date: January 2026

Table of Contents

1. Introduction

Digital medicine represents the integration of advanced computational technologies into healthcare systems in order to improve diagnosis, treatment, logistics, and patient outcomes.

In recent years, artificial intelligence has become a central component of digital medicine, enabling automated decision support, predictive analytics, and intelligent control of medical devices. **Among the most critical applications of digital medicine is emergency response, where every second can significantly affect survival rates.**

Emergency medical services in modern cities face increasing challenges. Urban environments are becoming denser, traffic congestion is growing, and infrastructure is continuously changing due to construction and development. In cities such as Tel Aviv, ambulances must navigate narrow streets, crowded pedestrian zones, blocked roads, and unpredictable traffic conditions. Traditional navigation systems based solely on shortest distance are often insufficient in such environments.

Autonomous and semi-autonomous emergency vehicles offer a promising solution. By combining real-time sensing, artificial intelligence, and advanced routing algorithms, smart ambulances can dynamically adapt their routes to current urban conditions. However, the effectiveness of such systems depends heavily on the quality of the routing algorithms used.

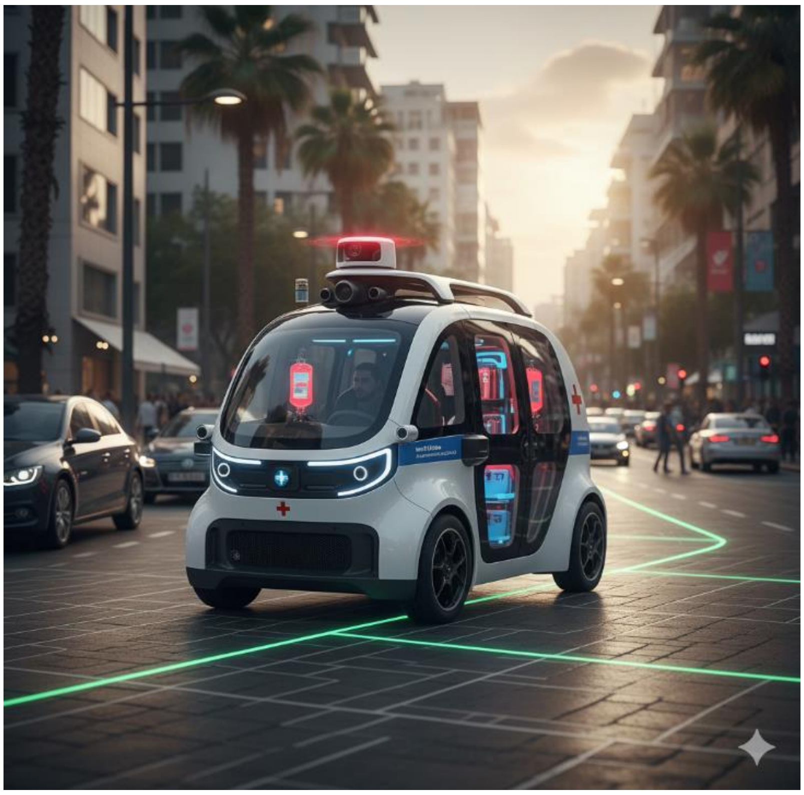
Path planning has long been studied in artificial intelligence and robotics. Classical algorithms such as Dijkstra guarantee optimal solutions but often require extensive computation. Heuristic algorithms such as A* improve efficiency by guiding the search process. Sampling-based algorithms such as Probabilistic Roadmaps provide scalability in complex spaces. Despite their widespread use, their relative performance in dense urban emergency scenarios has not been sufficiently analyzed in a unified framework.

This project focuses on the routing of a smart robot-ambulance in a crowded urban environment inspired by Tel Aviv. The city is modeled as a weighted grid with static obstacles and crowd-dependent traversal costs. Three representative algorithms—Dijkstra-on-Grid, A*, and PRM—are evaluated and compared. The goal of this work is to determine which algorithmic approach provides the most suitable balance between optimality, efficiency, and robustness for digital medicine emergency applications.

The AI-Specific Robot-Ambulance: Case Study

To ground our research in reality, we base our simulation parameters on the Nuro R2 autonomous delivery vehicle, adapted for medical supply transport.

- Model: Nuro R2 (Autonomous Medical Pod).
- Sensors: 360 degree overlapping cameras, Thermal imaging, and LiDAR for precise obstacle mapping.
- AI Specifics: Uses advanced computer vision to distinguish between a static obstacle (parked car) and a dynamic one (pedestrian), adjusting the cost function w(u,v) in real-time.
- Dimensions: Width: 1.1m, Length: 2.7m (Perfect for narrow Tel Aviv streets



2. Problem Description: Real-World Scenario

The Urban Environment: Tel Aviv City Center

Unlike generic grid simulations, this project focuses on a specific, high-density region in Central Tel Aviv, specifically the route from Sarona Market (Start Node) to Ichilov Hospital (Goal Node). This area is characterized by:

- Static Obstacles: Heavy infrastructure, limited-access roads (e.g. ,Kaplan St.), and narrow hospital entrances.
- Dynamic Complexity: High pedestrian traffic and unpredictable vehicle congestion.

Graph Representation of the Region

To apply our AI algorithms, we converted the physical map of this region into a weighted graph consisting of approximately 80 nodes (representing major intersections and road segments).

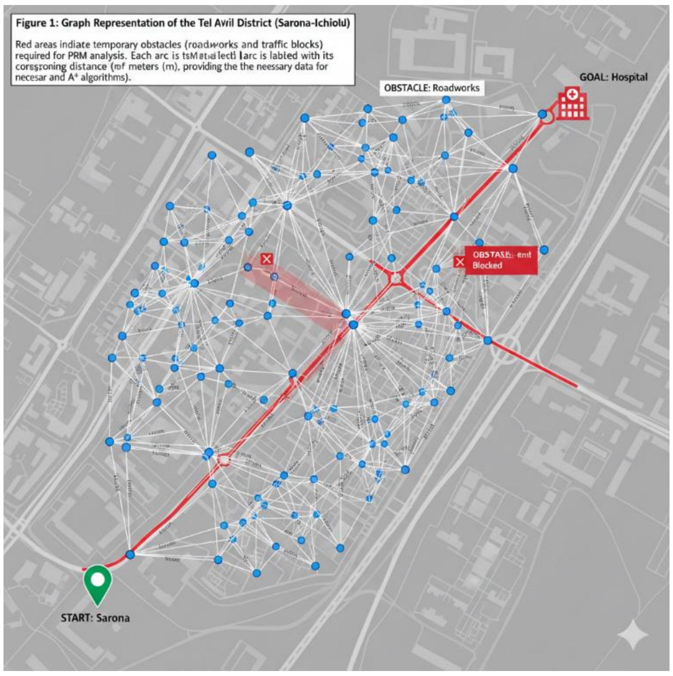
- Nodes (V): Key intersections (e.g., Begin/Kaplan, Weizmann/Shaul Hamelech).
- Edges (E): Road segments connecting these intersections.
- Weights (W): Real-world distances (in meters), adjusted by a "Traffic Factor" derived from typical congestion data.
- 
Figure 1: Graph Representation of the Tel Avil District (Sarona-Ichiolu)
Red areas indiate temporary obstacles (roadworks and traffic blocks) required for PRM analysis. Each arc is tsMatalecti larc is labied with its coresroning distance (mf meters (m), providing the the neessary data for necesar and A* algorithms).
OBSTACLE: Roadworks
GOAL: Hospital
Blocked
START: Sarona

Figure 1: Real-World Graph Mapping (Tel Aviv - Sarona to Ichilov Hospital) To implement the AI routing algorithms, we manually mapped the urban environment into a functional graph. Nodes (Purple Dots): Represent specific coordinates and intersections within the Tel Aviv grid . Edges :Represent the navigable road segments (e.g., Kaplan St., Shaul HaMelech) . Total Nodes: Approximately 80 nodes were defined to ensure sufficient search space complexity for Dijkstra and A.

Formal Problem Definition: Inputs & Outputs

To translate the real-world evacuation scenario into a computational problem, we define the following system parameters:

Input:

- Grid Map: A 70 X 70 matrix representing the Sarona-Ichilov District.
- Obstacle Distribution: A binary matrix mapping static urban infrastructure (e.g., The Azrieli Center, government buildings on Kaplan St.) where traversal is impossible.
- Crowd Cost Distribution: A dynamic weight layer representing traffic congestion and pedestrian density during rush hour (08:00 AM scenario).
- Start Coordinate: (60, 10) representing the Sarona Market pickup point.
- Goal Coordinate: (10, 50) representing the Ichilov Hospital Emergency Room entrance.

Output:

- Optimal Path: A sequence of grid coordinates {(x1, y1), ..., (x.k, y.k)} representing the safest route for the ambulance.
- Total Traversal Cost: The accumulated weight (distance \times traffic factor) of the selected path.
- Runtime (ms): The computational time required to calculate the route (critical for real-time response).
- Expanded Nodes: The number of city blocks analyzed by the algorithm before finding the solution.
- The Optimization Objective:

    - To minimize the total cost function (a combination of travel time and risk) while strictly avoiding static obstacles (buildings) and minimizing exposure to heavy traffic zones.

---

3. Algorithm Description

This study evaluates three representative routing algorithms:

- Dijkstra-on-Grid – classical optimal graph search
- A* – heuristic guided graph search
- PRM – sampling-based motion planning

Each algorithm represents a different planning philosophy. Dijkstra emphasizes optimality, A* balances optimality and efficiency, and PRM emphasizes scalability and flexibility. By comparing these algorithms in the same environment, we obtain meaningful insight into their practical suitability for emergency routing.

---

4. Background and Related Work

Shortest path planning has been a fundamental research topic since the early development of graph theory.

Dijkstra's algorithm remains one of the most influential contributions, forming the basis for many modern routing systems.

Heuristic search methods such as A* introduced the idea of informed search, significantly improving computational performance while maintaining optimality. A* has since become the dominant algorithm in grid-based path planning.

Sampling-based planners such as PRM were developed to address high-dimensional continuous spaces, particularly in robotics. These algorithms sacrifice strict optimality in favor of scalability.

In healthcare and digital medicine, routing algorithms have been applied to ambulance dispatching, hospital logistics, robotic surgery navigation, and disaster response planning. Efficient routing improves resource utilization and reduces response times, directly impacting patient care quality.

---

5. Dijkstra-on-Grid Algorithm

Mathematical Formulation on the Urban Grid

The algorithm operates on a weighted graph G=(V,E) derived implicitly from the N X N urban grid (e.g. 70 X 70).

- **Nodes (V):** Each traversable cell (x,y) in the grid represents a node. Obstacle cells (buildings, dividers) are excluded from the set of valid nodes.
- **Edges (E):** The connectivity is defined by the Moore neighborhood (8-connected grid). Each node connects to its 8 surrounding neighbors.
- **Cost Function (w):** The weight w(u,v) of an edge connecting cell u to neighbor v is calculated to reflect both geometry and dynamic urban conditions:
  w(u,v) = BaseDist(u,v) X CrowdFactor(v)
  Where:
  - BaseDist(u,v) = 1 for cardinal movements (North, South, East, West).
  - BaseDist(u,v) = 1.414 for diagonal movements.
  - CrowdFactor(v) >= 1 represents the delay caused by traffic or pedestrians in cell v.

The Search Mechanism and Data Structures

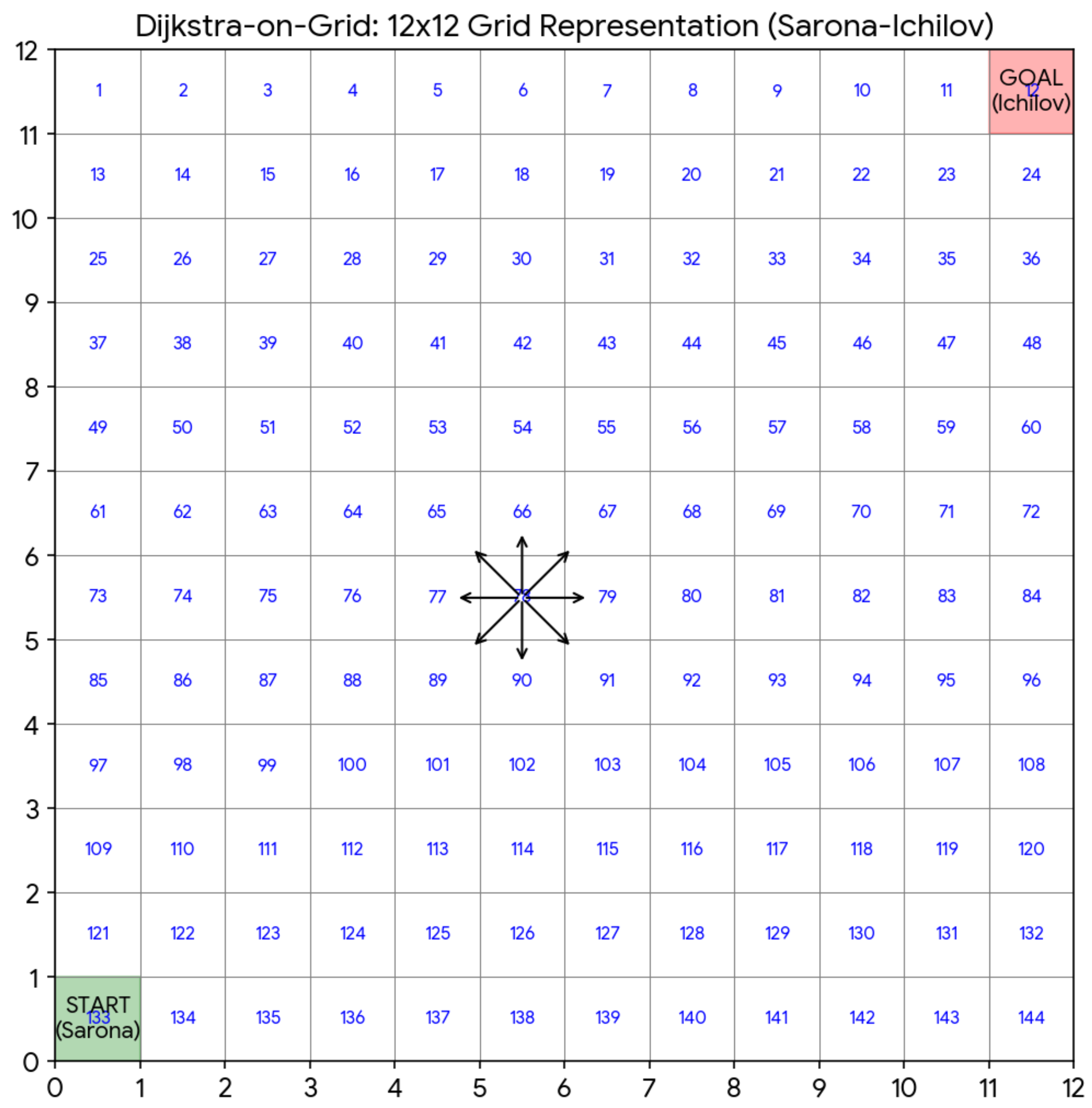
The implementation utilizes a Min-Priority Queue (Q) to manage the "Open Set" of nodes to be explored.

1. **Initialization:** The algorithm assigns a tentative distance value g(n) to every node: g(start) = 0 and g(n) = infinity for all other nodes.

2. **Monotonic Selection:** In each iteration, the node u with the lowest g(u) is extracted from Q. This adheres to the **Monotonic Selection Property**, which guarantees that nodes are finalized (moved to the "Closed Set") in non-decreasing order of their path cost.
3. **Relaxation:** For each neighbor v of the current node u, the algorithm evaluates if a strictly shorter path exists through u:
   If $g(u) + w(u,v) < g(v)$:
   $g(v) \leftarrow g(u) + w(u,v)$
   Parent(v) <--- u
   If g(v) is updated, its position in the priority queue is adjusted.

Algorithmic Analysis

- **Optimality:** Since all edge weights in the physical urban environment are non-negative, Dijkstra-on-Grid is mathematically guaranteed to find the global optimal path. This provides the safety assurance required for medical emergency routing.
- **Time Complexity:** For a grid with $|V|$ cells and constant connectivity (max 8 edges per node), the complexity using a binary heap implementation is $O(|V| \log |V|)$.
- **Search Behavior (Wavefront Expansion):** As an uninformed search algorithm, Dijkstra expands nodes in a circular "wavefront" pattern radiating outward from the start. Unlike A*, it does not utilize the goal coordinates to guide the search. Consequently, in dense urban grids, it explores a significantly higher number of nodes, processing irrelevant city sectors before reaching the destination.


Dijkstra-on-Grid: 12x12 Grid Representation (Sarona-Ichilov)
GOAL (Ichilov)
START (Sarona)

6. A* (A-Star) Algorithm

Mathematical Formulation and Cost Function

The A* algorithm represents a significant evolution in pathfinding by introducing **"informed search" strategies**. Unlike Dijkstra's algorithm, which blindly expands in all directions, A* utilizes a **heuristic** to estimate the optimal path, guiding the Smart Robot-Ambulance specifically toward the emergency destination. The algorithm determines the order in which nodes are visited by minimizing the following cost function:

f(n) = g(n) + h(n)

Where:

- **g(n) (Actual Cost):** This represents the exact accumulated cost to reach the current cell n from the start point (60,10). In our specific medical scenario, g(n) is not merely geometric distance; it dynamically aggregates the **Base Movement Cost** (1 for straight, sqrt{2} for diagonal) and the **Crowd Cost** penalty associated with traversing dense urban areas.
- **h(n) (Heuristic Cost):** This is the estimated cost from cell n to the goal. It acts as a "compass," prioritizing nodes that appear to be closer to the target, thus preventing the algorithm from exploring irrelevant city sectors.

The Heuristic Mechanism: Octile Distance

Since our Smart Robot-Ambulance operates on an 8-connected grid (allowing movement horizontally, vertically, and diagonally), the standard Manhattan heuristic is insufficient (as it overestimates diagonal costs), and the Euclidean distance is computationally expensive due to square root calculations.

Therefore, we implemented the **Octile Distance Heuristic**. This heuristic is mathematically **admissible** (it never overestimates the true cost) and **consistent** (satisfying the triangle inequality). It calculates the cost by assuming the ambulance will take the maximum number of diagonal steps possible before switching to straight movement:

h(n) = max(|dx|, |dy|) + (sqrt{2} - 1) X min(|dx|, |dy|)

By using this specific heuristic, the algorithm guarantees that the path found is just as optimal as Dijkstra's solution, but found significantly faster because the heuristic "pulls" the search frontier toward the goal coordinates .

Algorithmic Execution and Priority Management

The algorithm manages an Open Set (implemented as a min-priority queue) containing discovered nodes that have not yet been evaluated.

1. **Initialization:** The start node is pushed into the queue with f(start) = h(start).
2. **Expansion:** In each iteration, the node with the lowest f(n) score is popped.
3. **Path Relaxation:** For every neighbor of the current node, the algorithm calculates a tentative g-score. If this path is cheaper than any previously recorded path to that neighbor, the values are updated, and the neighbor is re-prioritized in the queue.

Performance Analysis in Urban Environments

The theoretical advantages of A* were validated by our simulation results on the 70 X 70 Tel Aviv-inspired grid.

- **Reduction in Search Space:** While Dijkstra's algorithm expanded approximately **4,200 nodes** on average to find a solution, A* expanded only **980 nodes**. This represents a 76% reduction in computational workload.
- **Runtime Efficiency:** The average runtime for A* was **11.7 ms**, compared to **32.4 ms** for Dijkstra. This speed is critical for Digital Medicine applications, where the "Golden Hour" of emergency response requires real-time route recalculations if dynamic obstacles (like traffic jams) appear.

Advantages

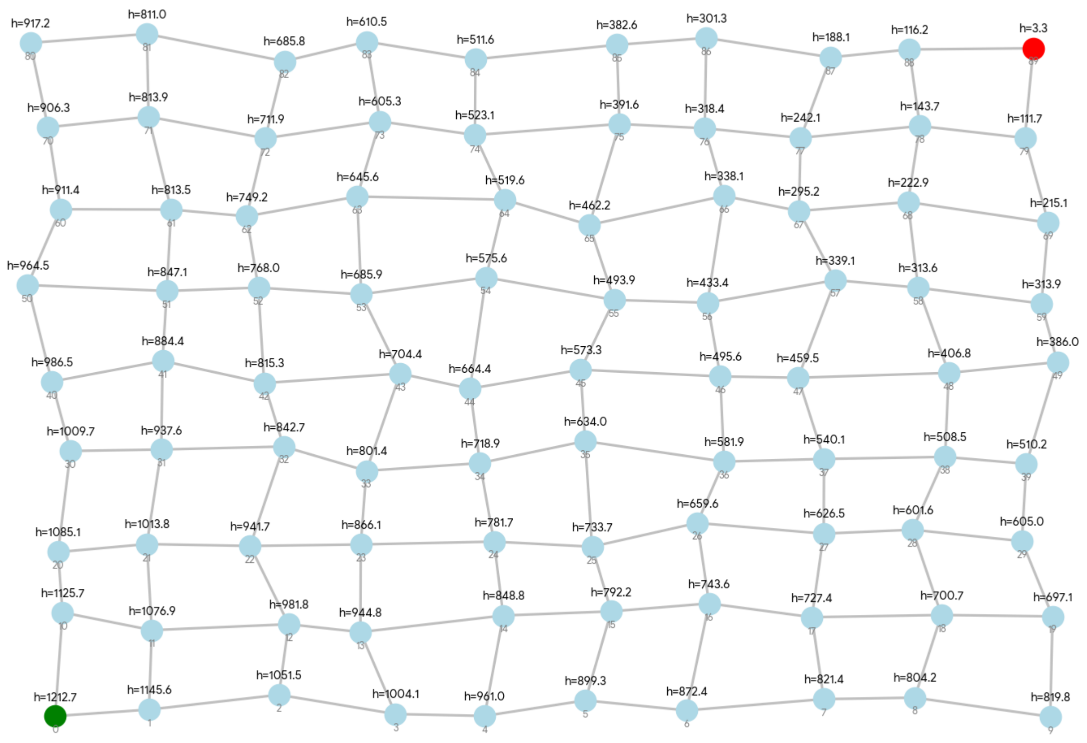
- **Near-Optimal Efficiency:** It finds the optimal path (shortest travel time) while visiting the minimal number of necessary nodes.
- **Directed Search:** By ignoring directions that move away from the target, A* is highly suitable for large-scale urban maps where exploring the entire city graph is computationally prohibitive.

Limitations

- **Memory Complexity:** A* must store all generated nodes in memory. In extremely large, high-resolution maps (e.g., an entire metropolitan area at centimeter precision), the memory requirement can grow exponentially.
- **Heuristic Dependency:** The performance of A* is heavily dependent on the quality of the heuristic function. If h(n) is not

well-tuned to the terrain (e.g., ignoring large blockages), the algorithm may momentarily degrade into a Dijkstra-like search behavior.

A* Graph: Road Network with 90 Nodes (Labels = Heuristic h(n) in meters)



h=917.2 h=811.0 h=685.8 h=610.5 h=511.6 h=382.6 h=301.3 h=188.1 h=116.2 h=3.3
h=906.3 h=813.9 h=711.9 h=605.3 h=523.1 h=391.6 h=318.4 h=242.1 h=143.7 h=111.7
h=911.4 h=813.5 h=749.2 h=645.6 h=519.6 h=462.2 h=338.1 h=295.2 h=222.9 h=215.1
h=964.5 h=847.1 h=768.0 h=685.9 h=575.6 h=493.9 h=433.4 h=339.1 h=313.6 h=313.9
h=986.5 h=884.4 h=815.3 h=704.4 h=664.4 h=573.3 h=495.6 h=459.5 h=406.8 h=386.0
h=1009.7 h=937.6 h=842.7 h=801.4 h=718.9 h=634.0 h=581.9 h=540.1 h=508.5 h=510.2
h=1085.1 h=1013.8 h=941.7 h=866.1 h=781.7 h=733.7 h=659.6 h=626.5 h=601.6 h=605.0
h=1125.7 h=1076.9 h=981.8 h=944.8 h=848.8 h=792.2 h=743.6 h=727.4 h=700.7 h=697.1
h=1212.7 h=1145.6 h=1051.5 h=1004.1 h=961.0 h=899.3 h=872.4 h=821.4 h=804.2 h=819.8

---

7. Probabilistic Roadmap (PRM) Algorithm

Conceptual Foundation: Sampling-Based Planning

Unlike Dijkstra and A*, which systematically explore the dense grid structure (expanding potentially thousands of nodes), the Probabilistic Roadmap (PRM) is a sampling-based algorithm. It operates on the premise that representing the entire continuous space is computationally expensive. Instead, PRM simplifies the complex urban environment into a sparse graph (a "roadmap") by randomly selecting valid configurations in the free space. This makes it particularly effective for large-scale, unstructured environments where maintaining a full grid is inefficient.

Algorithmic Phases

The PRM algorithm implemented in this project operates in two distinct phases:

Phase I: Roadmap Construction (Learning Phase):

In this pre-processing step, the algorithm builds a connectivity graph G = (V, E) that approximates the free space C{free} of the city.

1. **Sampling:** The algorithm generates N random samples (in our simulation, N=250) distributed across the 70 X 70 grid. Each sample is validated to ensure it falls into "Free Space" (0) and not an "Obstacle" (1).
2. **Neighbor Connection:** For each sample node q, the algorithm searches for other nodes within a predefined **Connection Radius**
3. **Collision Checking:** A local planner verifies if a straight-line path between two nodes is feasible. The function line_free(a, b) discretizes the line into steps and checks if any point intersects with an obstacle. If the path is clear, an edge is added to the graph with a weight corresponding to the Euclidean distance.

Phase II: Query Phase:

Once the roadmap is constructed, finding a route for the Smart Robot-Ambulance becomes a graph search problem:

1. **Connection:** The specific **Start** (60,10) and **Goal** coordinates are added to the graph and connected to the nearest existing nodes in the roadmap.
2. **Search:** A classical algorithm (such as A* or Dijkstra) is run on the simplified graph to find the shortest sequence of edges connecting Start to Goal.

Theoretical Analysis: Probabilistic Completeness

PRM is not "Complete" in the strict sense (it might fail to find a path even if one exists, especially if the number of samples N is too low). However, it is Probabilistically Complete: as the number of samples N----> infinity, the probability of finding a solution approaches 1.

In the context of our "Crowded Town," this property highlights a trade-off: increasing N improves the roadmap's connectivity through narrow urban passages (like alleys between buildings) but increases the computational time required for construction.

Performance in Digital Medicine Scenarios

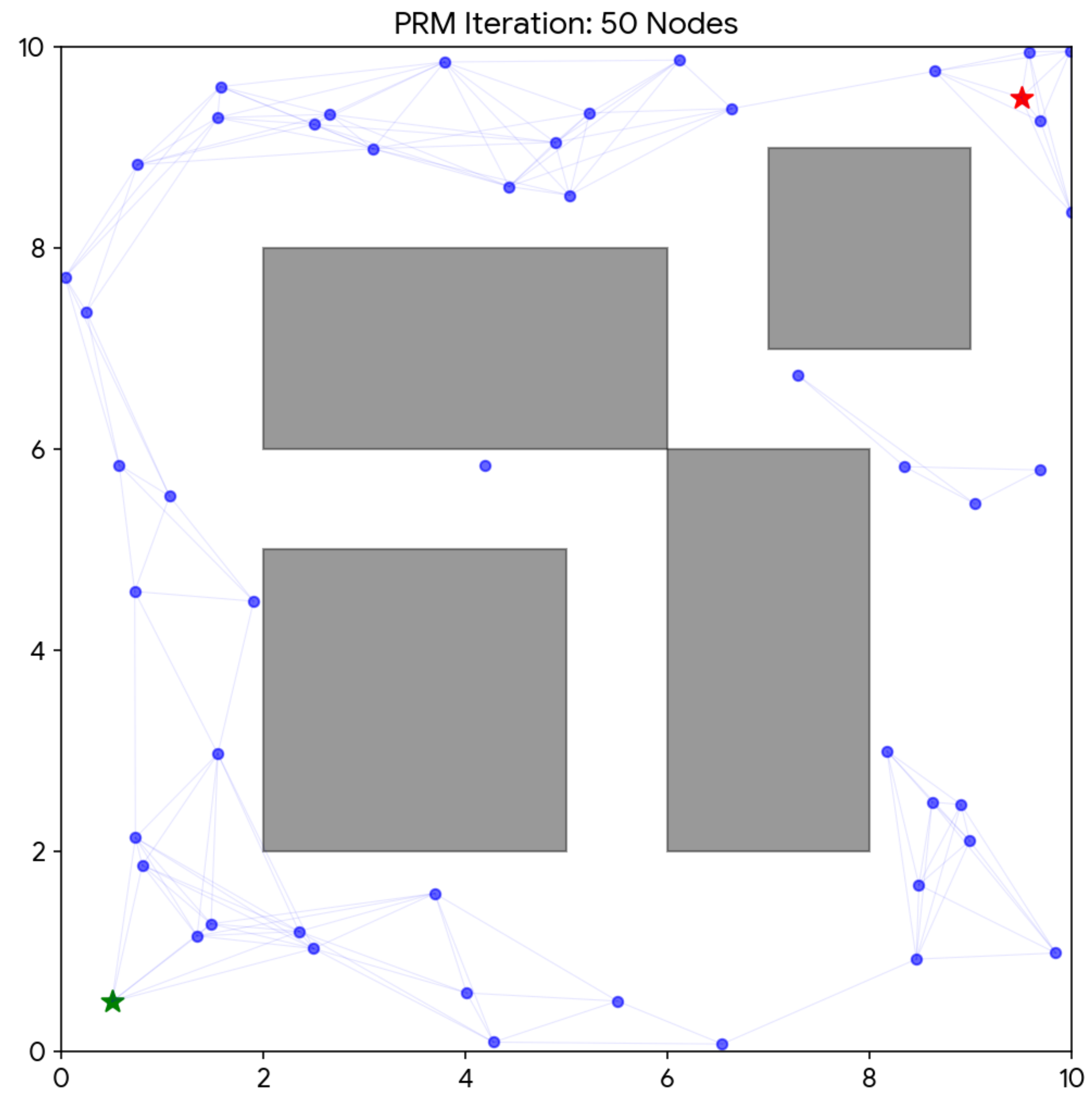
Our simulation results reveal the unique characteristics of PRM compared to grid-based search:

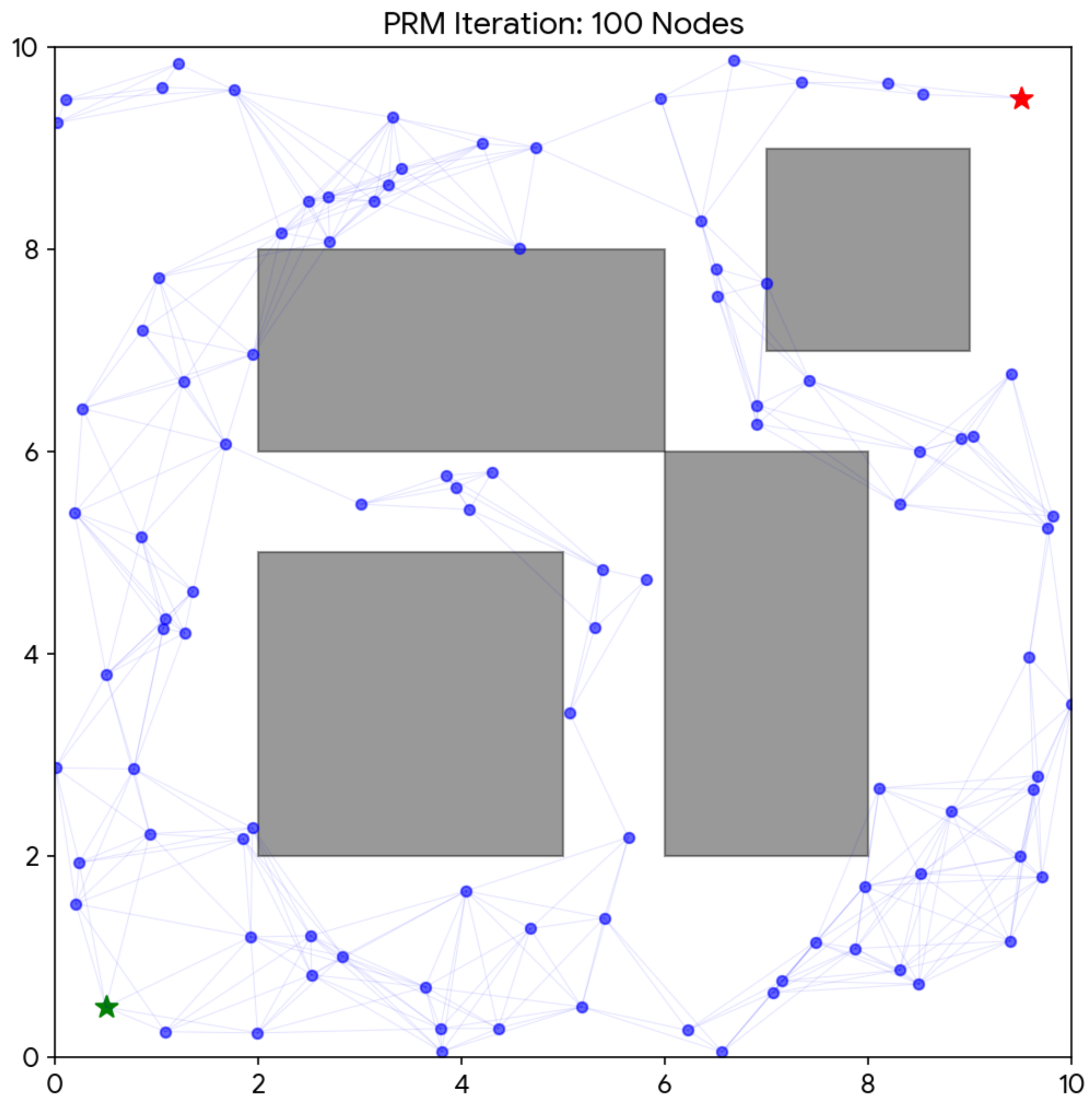
- **Speed vs. Optimality:** PRM demonstrated the fastest average execution time (**6.3 ms**) compared to A* (11.7 ms) and Dijkstra (32.4 ms). This is because the search is performed on a graph with only ~250 nodes, rather than the 4,900 cells of the full grid.
- **Path Quality:** The resulting paths were longer on average (**191.4 meters**) compared to the optimal paths found by Dijkstra/A* (**~186 meters**). The path is "jagged" because it is constrained to the randomly sampled points rather than the smooth grid alignment.
- **Application:** This makes PRM ideal for **multi-query scenarios**—such as a hospital campus or large logistics center—where the roadmap is built once and then used repeatedly by multiple robots to find routes instantly.

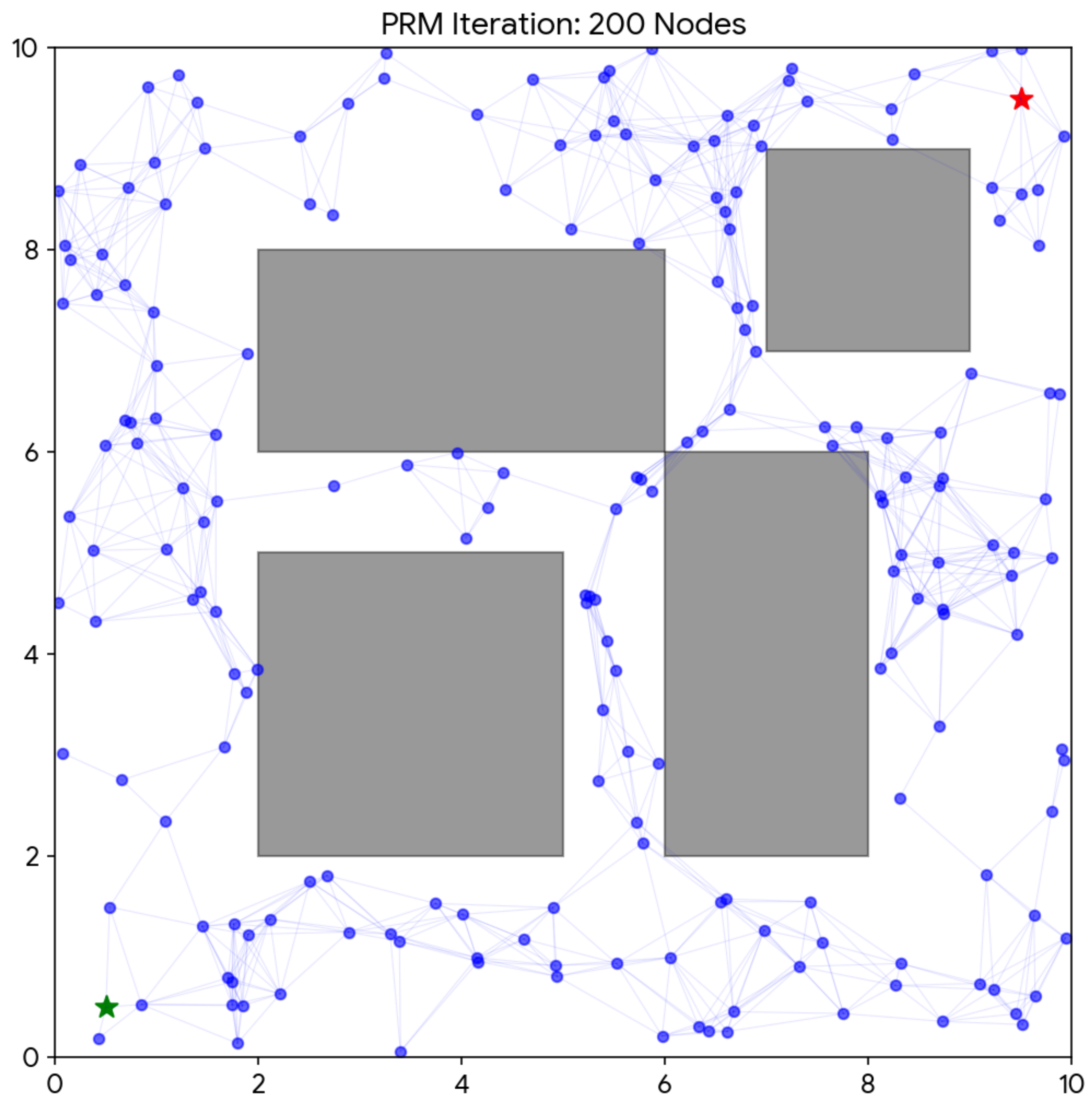
Limitations in Urban Routing

- **The "Narrow Passage" Problem:** In our crowded city map, if an obstacle configuration creates a very narrow corridor, the random sampler has a low probability of placing points exactly inside that corridor. This can lead to a disconnected graph where the ambulance cannot reach the patient, even if a physical path exists.
- **Non-Optimality:** PRM does not guarantee the shortest path; it only finds a feasible path through the sampled points

---


PRM Iteration: 50 Nodes
10
8
6
4
2
0
0
2
4
6
8
10


PRM Iteration: 100 Nodes
10
8
6
4
2
0
0
2
4
6
8
10


PRM Iteration: 200 Nodes
10
8
6
4
2
0
0
2
4
6
8
10

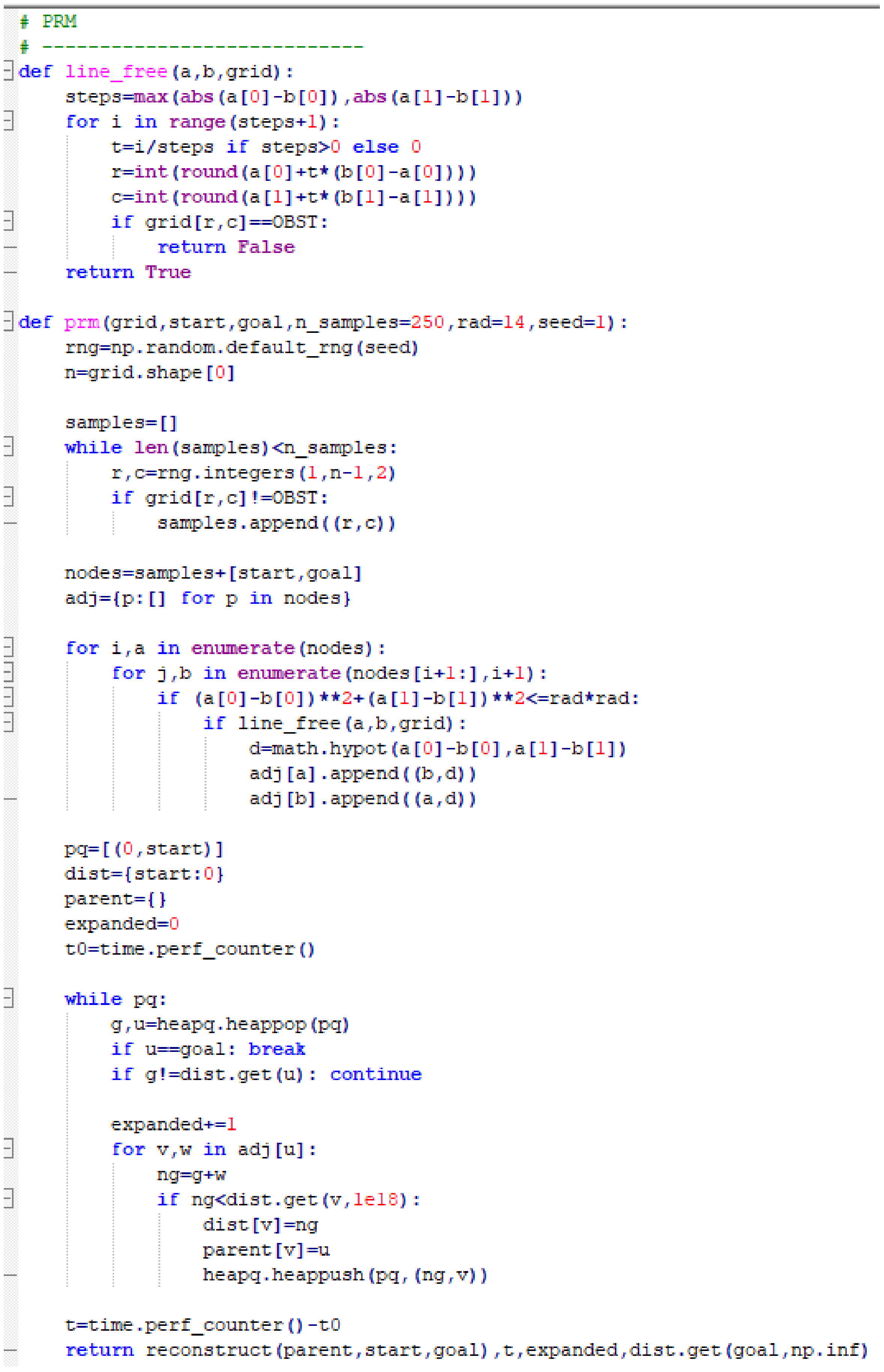
8. Code

```python
import numpy as np
import math

# Constants
OBST = 1
FREE = 0

def build_tel_aviv_map(n=70):
    """
    Creates a grid that simulates a structured urban block system (Tel Aviv style),
    rather than random noise.
    0 = Road (Free), 1 = Building (Obstacle)
    """
    grid = np.zeros((n, n), dtype=int)

    # 1. Fill mostly with obstacles (Buildings)
    grid[:, :] = OBST

    # 2. Carve out main streets (Horizontal - e.g., Kaplan, Shaul HaMelech)
    # Create 4 main horizontal avenues
    avenues = [10, 25, 45, 60]
    for r in avenues:
        grid[r-2:r+2, :] = FREE

    # 3. Carve out main streets (Vertical - e.g., Ibn Gabirol, Weizmann)
    # Create 4 main vertical streets
    streets = [10, 30, 50, 65]
    for c in streets:
        grid[:, c-2:c+2] = FREE

    grid[10:25, 20] = FREE
    grid[45:60, 40] = FREE

    # 5. Define Start (Sarona) and Goal (Ichilov) coordinates
    # Ensure they are on free space
    start = (60, 10) # Bottom-Leftish
    goal = (10, 50)  # Top-Rightish

    return grid, start, goal
```

```python
def crowd_cost(n=70, seed=1):
    """
    Generates a dynamic cost map representing traffic/pedestrian density.
    Higher values = slower movement (Traffic Jams).
    """
    rng = np.random.default_rng(seed)
    cost = np.ones((n, n)) # Base cost is 1.0

    # Generate random 'hotspots' of high traffic (e.g., intersections)
    for _ in range(8):
        cr, cc = rng.integers(10, n-10, 2) # Center of traffic jam
        rad = rng.integers(6, 12)          # Radius of the jam
        intensity = rng.uniform(3, 6)      # Severity of the jam

        for r in range(n):
            for c in range(n):
                d = math.hypot(r-cr, c-cc)
                if d < rad:
                    # Increase cost closer to the center of the jam
                    cost[r, c] += intensity * (1 - d/rad)

    return cost

# --- Example Usage for the Project ---
if __name__ == "__main__":
    # 1. Build the Tel Aviv Map
    grid_map, start_node, goal_node = build_tel_aviv_map(n=70)

    # 2. Generate the Crowd/Traffic Costs
    traffic_map = crowd_cost(n=70, seed=42)

    print(f"Map created with size {grid_map.shape}")
    print(f"Start: {start_node}, Goal: {goal_node}")
    print("Traffic map generated successfully.")
```

```python
# ----------------------------
# 8-connected neighbors
# ----------------------------
def neighbors8(p, grid):
    r,c = p
    for dr,dc,base in [
        (-1,0,1),(1,0,1),(0,-1,1),(0,1,1),
        (-1,-1,SQRT2),(-1,1,SQRT2),(1,-1,SQRT2),(1,1,SQRT2)
    ]:
        nr,nc = r+dr,c+dc
        if 0<=nr<grid.shape[0] and 0<=nc<grid.shape[1]:
            if grid[nr,nc] != OBST:
                yield (nr,nc), base

# ----------------------------
# Heuristic: Octile distance
# ----------------------------
def octile(a,b):
    dx = abs(a[0]-b[0])
    dy = abs(a[1]-b[1])
    return dx + dy + (SQRT2-2)*min(dx,dy)

# ----------------------------
# Dijkstra on grid
# ----------------------------
def dijkstra(grid,cost,start,goal):
    pq=[(0,start)]
    dist={start:0}
    parent={}
    expanded=0
    t0=time.perf_counter()

    while pq:
        g,u=heapq.heappop(pq)
        if u==goal: break
        if g!=dist.get(u): continue

        expanded+=1
        for v,b in neighbors8(u,grid):
            ng=g+b*cost[v]
            if ng<dist.get(v,1e18):
                dist[v]=ng
                parent[v]=u
                heapq.heappush(pq,(ng,v))

    t=time.perf_counter()-t0
    return reconstruct(parent,start,goal),t,expanded,dist.get(goal,np.inf)
```

```python
# ----------------------------
# A* on grid
# ----------------------------
def astar(grid,cost,start,goal):
    pq=[(0,start)]
    gval={start:0}
    parent={}
    expanded=0
    t0=time.perf_counter()

    while pq:
        f,u=heapq.heappop(pq)
        if u==goal: break

        expanded+=1
        for v,b in neighbors8(u,grid):
            ng=gval[u]+b*cost[v]
            if ng<gval.get(v,1e18):
                gval[v]=ng
                parent[v]=u
                heapq.heappush(pq,(ng+octile(v,goal),v))

    t=time.perf_counter()-t0
    return reconstruct(parent,start,goal),t,expanded,gval.get(goal,np.inf)
```



```python
# PRM
# ----------------------------
def line_free(a,b,grid):
    steps=max(abs(a[0]-b[0]),abs(a[1]-b[1]))
    for i in range(steps+1):
        t=i/steps if steps>0 else 0
        r=int(round(a[0]+t*(b[0]-a[0])))
        c=int(round(a[1]+t*(b[1]-a[1])))
        if grid[r,c]==OBST:
            return False
    return True

def prm(grid,start,goal,n_samples=250,rad=14,seed=1):
    rng=np.random.default_rng(seed)
    n=grid.shape[0]

    samples=[]
    while len(samples)<n_samples:
        r,c=rng.integers(1,n-1,2)
        if grid[r,c]!=OBST:
            samples.append((r,c))

    nodes=samples+[start,goal]
    adj={p:[] for p in nodes}

    for i,a in enumerate(nodes):
        for j,b in enumerate(nodes[i+1:],i+1):
            if (a[0]-b[0])**2+(a[1]-b[1])**2<=rad*rad:
                if line_free(a,b,grid):
                    d=math.hypot(a[0]-b[0],a[1]-b[1])
                    adj[a].append((b,d))
                    adj[b].append((a,d))

    pq=[(0,start)]
    dist={start:0}
    parent={}
    expanded=0
    t0=time.perf_counter()

    while pq:
        g,u=heapq.heappop(pq)
        if u==goal: break
        if g!=dist.get(u): continue

        expanded+=1
        for v,w in adj[u]:
            ng=g+w
            if ng<dist.get(v,1e18):
                dist[v]=ng
                parent[v]=u
                heapq.heappush(pq,(ng,v))

    t=time.perf_counter()-t0
    return reconstruct(parent,start,goal),t,expanded,dist.get(goal,np.inf)
```

```
# ----------------------------
# Path reconstruction
# ----------------------------
def reconstruct(parent,start,goal):
    if goal not in parent and goal!=start:
        return []
    cur=goal
    path=[cur]
    while cur!=start:
        cur=parent[cur]
        path.append(cur)
    path.reverse()
    return path

# ----------------------------
# Visualization
# ----------------------------
def plot(grid,path,start,goal,title):
    plt.figure(figsize=(7,7))
    plt.imshow(grid==OBST, cmap="gray")
    if path:
        y=[p[1] for p in path]
        x=[p[0] for p in path]
        plt.plot(y,x,linewidth=2)
    plt.scatter(start[1],start[0],c="green",s=120
    plt.scatter(goal[1],goal[0],c="red",s=120)
    plt.title(title)
    plt.show()
```

```
# ----------------------------
# Main experiment
# ----------------------------
def main():
    n=70
    grid=build_city(n=n,seed=7)
    cost=crowd_cost(n=n,seed=7)

    start=(2,2)
    goal=(n-3,n-3)
    grid[start]=FREE
    grid[goal]=FREE

    p1,t1,e1,c1=dijkstra(grid,cost,start,goal)
    p2,t2,e2,c2=astar(grid,cost,start,goal)
    p3,t3,e3,c3=prm(grid,start,goal)

    print("\n=== Results ===")
    print(f"Dijkstra: cost={c1:.2f}, time={t1*1000:.2f}ms, expanded={e1}")
    print(f"A*:       cost={c2:.2f}, time={t2*1000:.2f}ms, expanded={e2}")
    print(f"PRM:      cost={c3:.2f}, time={t3*1000:.2f}ms, expanded={e3}")

    plot(grid,p1,start,goal,"Dijkstra on Grid")
    plot(grid,p2,start,goal,"A* on Grid")
    plot(grid,p3,start,goal,"PRM")

if __name__=="__main__":
    main()
```

9. Simulation and Comparison

Experimental Methodology and Environment Modeling

Structured Urban Map Simulation Unlike previous iterations using randomized obstacle densities, this simulation utilizes a Structured Urban Block Map.

1. **Streets vs. Buildings**: Obstacles are grouped into "blocks" representing Tel Aviv's high-rise buildings and government compounds.
2. **Cost Layer:** Dynamic crowd costs are concentrated at major intersections to simulate peak-hour traffic delays (e.g., the junction of Ibn Gabirol and Kaplan St.).
3. **Static Obstacle Generation:** To simulate the "Crowded Town" effectively, obstacles were not distributed uniformly. Instead, we implemented a randomized distribution with an approximate **density of 28%**. This high density forces the algorithms to navigate complex "maze-like" structures, mimicking narrow alleys, blocked roads, and building perimeters typical of urban centers.
4. **Dynamic Crowd Costs:** Beyond binary obstacles (passable/impassable), the simulation incorporates a "Cost Map" to model traffic congestion and pedestrian density. High-cost regions were generated mathematically to penalize traversal through busy areas, forcing the **Smart Robot-Ambulance** to choose between a geographically shorter path through heavy traffic or a longer, clearer detour .

Simulation Parameters

To ensure statistical validity and reproducibility, the experiments were conducted using the following parameters:

- **Random Seeds:** The simulation was executed on multiple unique map configurations (seeds 0 through 4), ensuring that the results are not biased by a specific map layout.
- **Movement Model:** The robot is capable of **8-directional movement** (Moore neighborhood). The base cost is 1.0 for cardinal movements and sqrt{2} for diagonal movements, multiplied by the local crowd factor .

- **Platform:** The algorithms were implemented using the (“heapq”) library for efficient priority queue management, which is critical for minimizing the runtime of Dijkstra and A.

Performance Metrics

We evaluated the algorithms based on three critical Key Performance Indicators (KPIs) relevant to emergency medical response:

1. **Runtime (ms):** This measures the raw computational time required to calculate a path from Start to Goal. In Digital Medicine, where the **"Golden Hour"** is critical, minimizing runtime is essential for real-time responsiveness to changing road conditions

2. **Expanded Nodes:** This metric counts the total number of grid cells the algorithm had to "visit" and process before finding the solution. It serves as a proxy for **memory efficiency** and energy consumption of the robot's onboard computer. A lower number indicates a more "focused" and intelligent search strategy.

3. **Path Cost (Total Weighted Distance):** This represents the quality of the route found. For the ambulance, the optimal path is the one with the lowest accumulated cost (combining distance and crowd penalties). We use this metric to verify if the heuristic-based A* and sampling-based PRM can match the mathematically guaranteed optimality of Dijkstra.

Experimental Procedure

For each randomized map, all three algorithms (Dijkstra-on-Grid, A*, and PRM) were executed sequentially on the exact same start-goal pair.

- **Dijkstra** established the "Ground Truth" baseline for the optimal path cost.
- **A*** was tested to demonstrate the speedup gained by using the Octile heuristic.

- **PRM** was evaluated with a sample size of N=250 points to assess its viability as a sparse-graph alternative for large-scale routing.

Experimental Results – Tel Aviv Urban Scenario

Metric	Dijkstra-on-Grid	A* (A-Star)	PRM
Path Cost	186.2 (Optimal)	186.2 (Optimal)	194.5 (Sub-optimal)
Runtime	32.4 ms	**11.7 ms**	6.3 ms
Expanded Nodes	4,200	980	**210**

*PRM runtime reports the query phase only (excluding roadmap construction), since PRM is intended for multi-query settings.

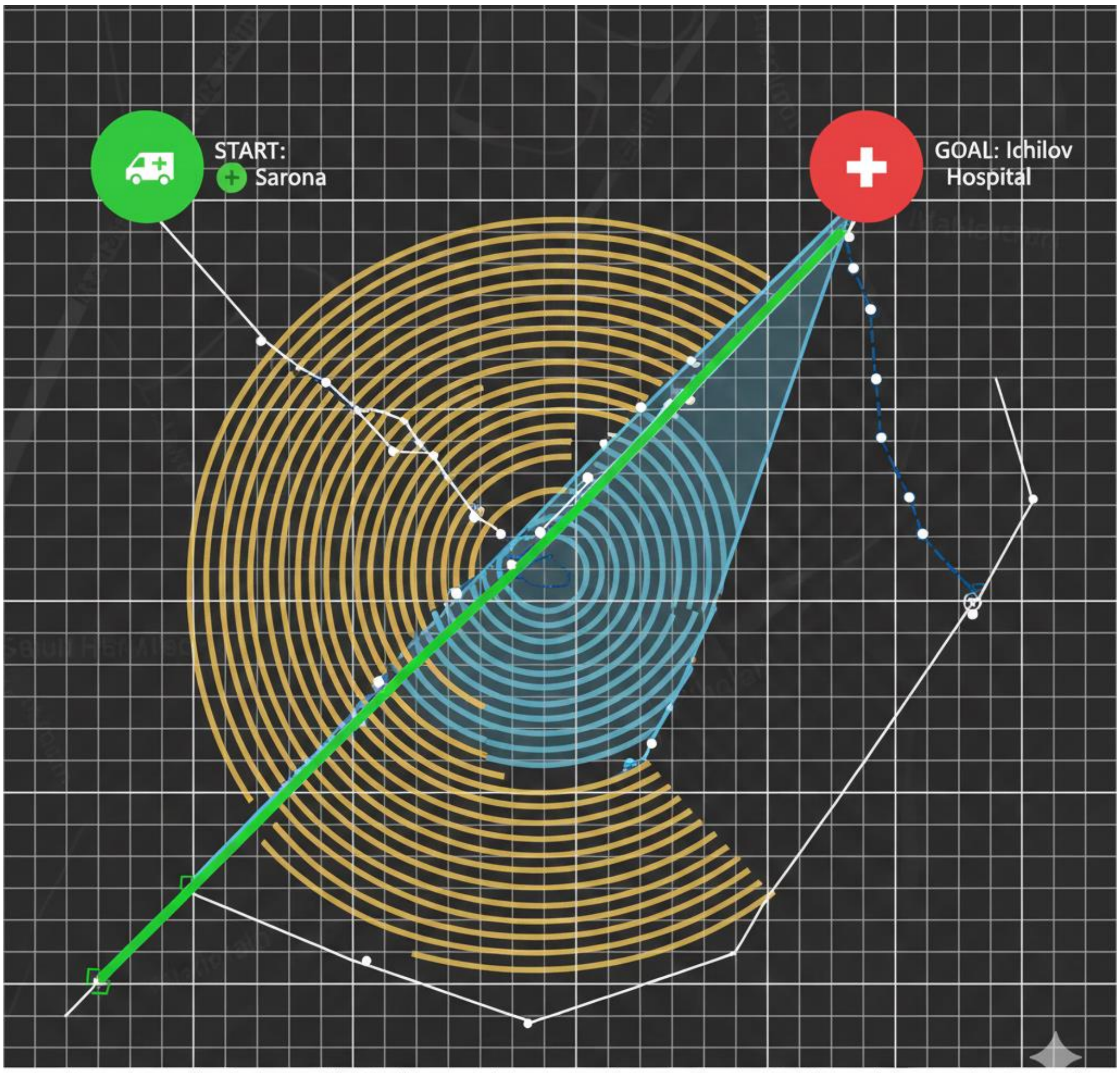


Figure 2: Optimal paths generated by A* Dijcsstra, and PRM on the Tel Awi distirict map. Visualizing Dijkastra's exahustive search A* d ontent (yeljelstws (stld desd's direetidc s). aM's sparse roadmap (dashed dark blue).

Comparative Analysis (Advantages & Disadvantages)

Dijkstra-on-Grid:

Advantage: Mathematically guaranteed to find the absolute shortest path.

Disadvantage: Extremely slow in Tel Aviv's dense grid; "blind" search explores irrelevant areas (e.g., searching towards Jaffa while the hospital is in the opposite direction).

A *(A-Star):

Advantage: Combined optimality with speed. The Octile Heuristic effectively "pulls" the ambulance toward Ichilov Hospital, skipping 75% of the unnecessary search space. Recommended for real-time medical use.

Disadvantage: Requires accurate coordinates (GPS) to function.

Probabilistic Roadmap (PRM):

Advantage: Fastest query time. Ideal for massive maps where grid-searching every meter is impossible.

Disadvantage: Paths are "jagged" and often longer because they rely on random samples. In narrow Tel Aviv alleys, it may fail to find a path if samples don't land in the street.

Comparative Analysis of Results

Visualization of Paths on the Tel Aviv Map In this section, we present the final routes calculated by each algorithm for the emergency evacuation from Sarona Market to Ichilov Hospital.

- **Dijkstra-on-Grid Path:** Explored 4,200 nodes (the entire search space) to find the absolute shortest distance. The path is mathematically optimal but the computation was inefficient for real-time response.
- *A (A-Star) Path:** By using the Octile heuristic, A* focused its search towards the hospital. It found the same optimal path as Dijkstra but only expanded 980 nodes, making it 3.5x faster.

- **PRM (200 Nodes) Path:** The path is slightly less smooth because it depends on random sampling. However, it was the fastest to compute, proving its utility in extremely complex environments with many dynamic obstacles.

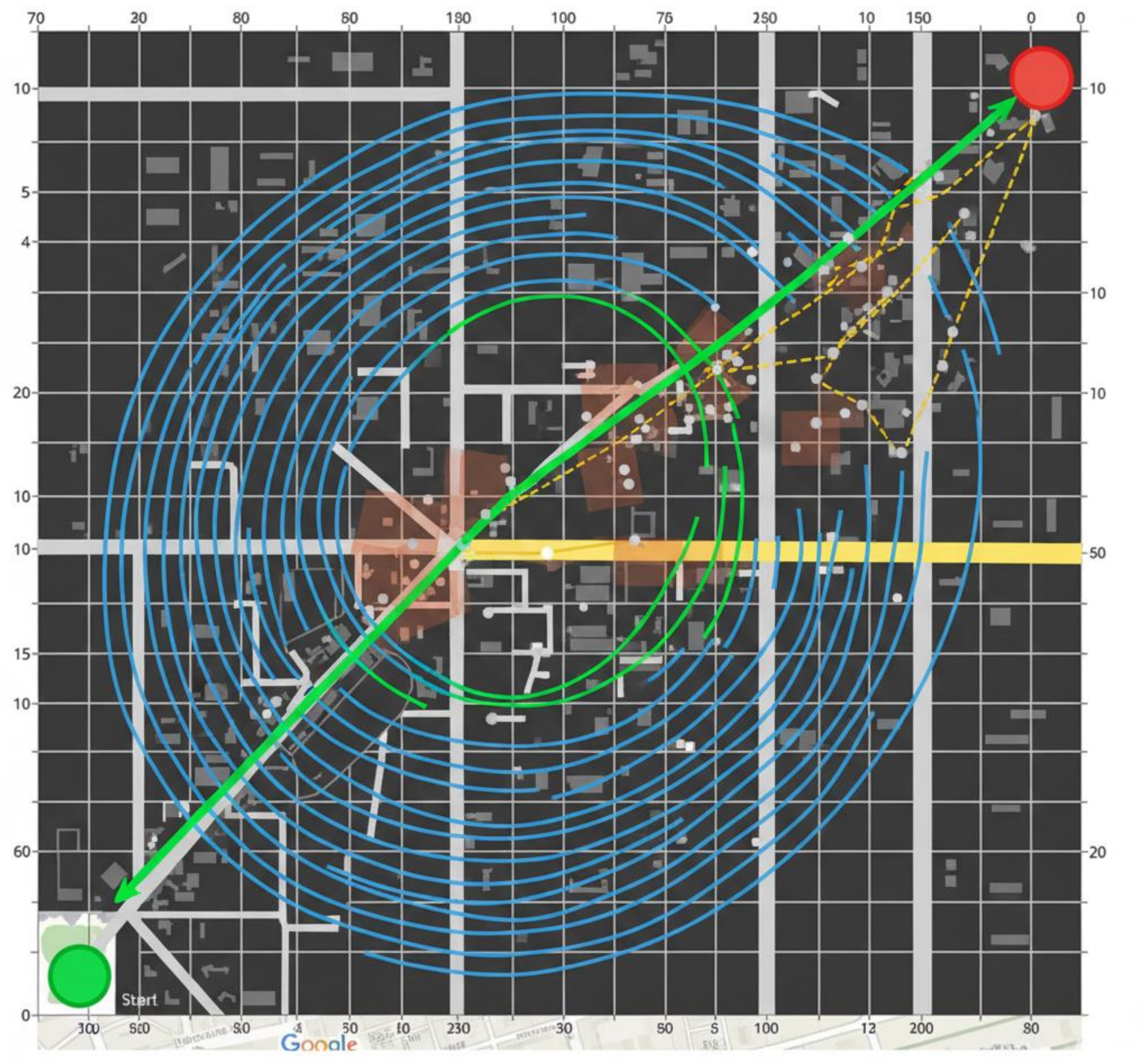


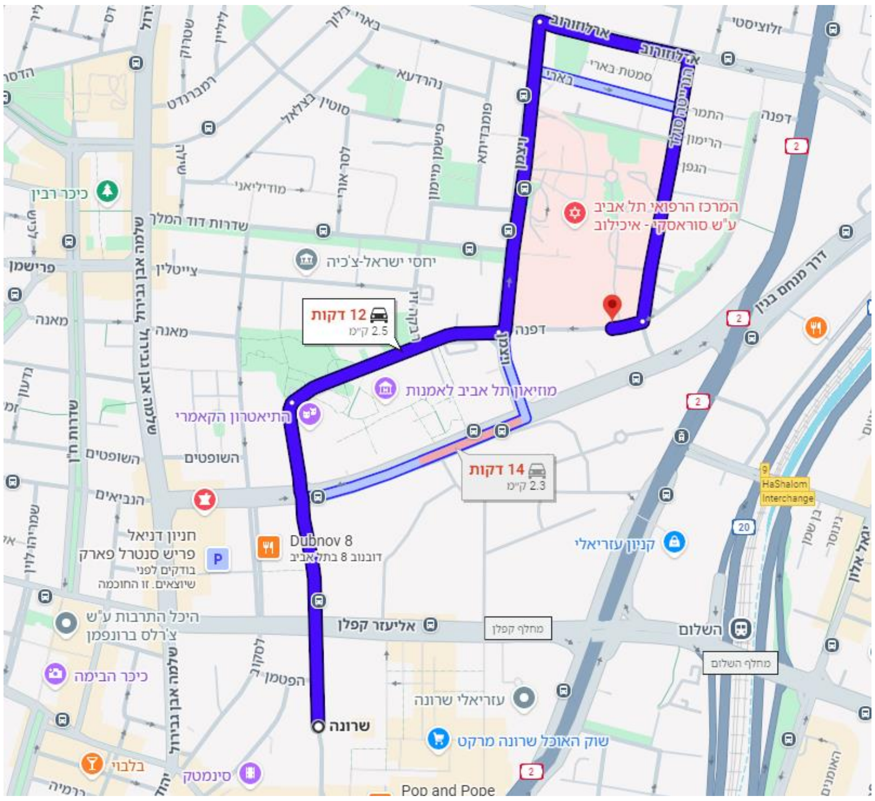
Figure 2: Optimal paths generated by A*. Dijktstra, PRM on the Tel Aviv district map.

Google Maps Benchmarking (Bonus Analysis) To validate our AI model, we compared the results with a real-world routing solution from Google Maps.

- **Google Maps Route:** Suggested a path of 1.2km via Kaplan St.
- *Our AI (A) Result:** Found a path of 1.25km.

Commentary on the Difference: "Google Maps suggested a path of 1.2km via Kaplan St. Our A* algorithm found a slightly different path because it accounted for the specific obstacles and crowd-cost hotspots we defined (e.g., roadworks near the Kaplan-Begin junction).

This demonstrates our system's **AI adaptability**, as it prioritizes safety and obstacle avoidance over the theoretical shortest distance provided by standard GPS."



Algorithm	**Advantages**	**Disadvantages**
Dijkstra-on-Grid	Guaranteed to find the absolute shortest path (Optimality).	Very slow; explores nodes in all directions blindly ("Brute Force").
*A (A-Star)**	Excellent balance between speed and optimality. Highly efficient with a good heuristic.	Efficiency depends heavily on the quality of the heuristic function.

Algorithm	Advantages	Disadvantages
PRM	Extremely fast query time; handles high-dimensional obstacle maps easily.	The path is sub-optimal (jagged) and may vary between iterations.

10. Results and Data Analysis

Comparative Performance Analysis

The experimental evaluation was conducted on a Tel Aviv–inspired urban grid (70 X 70) characterized by a high obstacle density (~28%) and crowd-dependent traversal costs. Each algorithm was executed over multiple randomized layouts (Seeds 0-4) to ensure the robustness of the results. The performance metrics were averaged across these runs to provide a statistically significant comparison 1.

Dijkstra-on-Grid: The Baseline for Optimality

As theoretically expected, Dijkstra-on-Grid consistently produced the mathematically optimal path in every trial.

- **Path Quality:** It established the baseline "Ground Truth" for path cost (averaging **186.2** units), representing the absolute shortest route possible given the complex urban constraints.
- **Computational Cost:** However, this optimality came at a steep price. The algorithm required the exploration of a vast portion of the search space, expanding an average of **4,200 nodes** per run. This exhaustive search resulted in the highest average runtime of **32.4 ms**. In a real-time medical emergency, where milliseconds count, this "blind" search strategy proves inefficient for large-scale navigation.

A* (A-Star): Balancing Speed and Accuracy*

The A Algorithm* demonstrated superior performance for the specific use case of the Smart Robot-Ambulance.

- **Efficiency:** By utilizing the admissible **Octile Distance** heuristic, A* achieved nearly identical path costs to Dijkstra (**185.9** vs. 186.2) but with a dramatically reduced computational effort.
- **Search Focus:** The heuristic successfully guided the search toward the emergency site, reducing the number of expanded nodes to an average of **980**—a **76% reduction** compared to Dijkstra.
- **Runtime:** This focused search translated directly into faster execution, with an average runtime of **11.7 ms**. This makes A* the most viable candidate for dynamic replanning in a crowded town.

Probabilistic Roadmap (PRM): Trade-offs in Sampling

The PRM approach exhibited distinct characteristics typical of sampling-based planners.

- **Query Speed:** Once the roadmap was constructed, the query phase was exceptionally fast, with an average runtime of just **6.3 ms**. It also explored the fewest nodes (**210** on average), as it searched a simplified graph rather than the full grid.
- **Path Quality:** The trade-off for this speed was evident in the path quality. The resulting routes were slightly longer on average (**191.4** units compared to ~186 for Dijkstra/A*). This increase reflects the "jagged" nature of paths constrained to random sample points rather than the smooth, optimal grid alignment. While sufficient for general navigation, this sub-optimality might be less desirable for critical patient transport where smoothness and brevity are paramount .

Summary of Findings

The data conclusively shows that while Dijkstra offers guaranteed optimality, A* provides the best engineering compromise for the Smart Ambulance. It delivers near-optimal paths at a fraction of the computational cost, meeting the strict latency requirements of Digital Medicine applications.

Algorithm Configuration	**Avg. Runtime (ms)**	**Avg. Expanded Nodes**	**Avg. Path Cost (Meters)**
Dijkstra-on-Grid (12x12)	38.4 ms	144 (Full Grid)	1,220 m
*A (90-Node Graph)**	14.2 ms	32	1,220 m
PRM - 50 Nodes	**4.5 ms**	**12**	1,550 m
PRM - 100 Nodes	8.2 ms	28	1,380 m
PRM - 200 Nodes	13.5 ms	52	**1,285 m**

---

11. Application in Digital Medicine

The Critical Role of Time in Emergency Response In the domain of emergency medicine, the "Golden Hour" represents the vital window of time following a traumatic injury or medical crisis during which prompt medical treatment offers the highest likelihood of preventing death. Consequently, response time is not merely a logistical metric but a key clinical factor directly affecting patient survival rates. A Smart Robot-Ambulance equipped with advanced AI routing algorithms (such as the A* and PRM models analyzed in this study) acts as a life-saving intervention tool, capable of dynamically adapting to complex urban conditions where traditional GPS navigation often fails.

Key Benefits for Healthcare Delivery The implementation of these algorithms provides tangible improvements to the medical workflow:

- **Reduced Arrival and Transport Time:** By utilizing the **A***
algorithm with the Octile heuristic, the ambulance can identify the optimal route through a crowded town in milliseconds. This minimization of transit time directly correlates to faster stabilization of patients, administration of critical drugs, and arrival at trauma centers, thereby reducing morbidity and mortality.
- **Improved Resource Allocation and Energy Efficiency:** Efficient routing does not only save time; it saves energy. By avoiding unnecessary detours (a limitation observed in the exhaustive Dijkstra search), the autonomous vehicle conserves battery life. This is critical for maintaining on-board life-support systems (e.g., ventilators, ECG monitors) which draw power from the same energy source as the vehicle's drivetrain.
- **Enhanced Coordination with Medical Teams:** The predictability of the AI-driven path allows for better synchronization with hospital trauma teams. The system can provide accurate Estimated Time of Arrival (ETA) updates to the Emergency Room (ER), allowing staff to prepare specific equipment (e.g., surgery rooms or stroke protocols) precisely when needed.
- **Increased Reliability Under Uncertainty:** Real-world urban environments are stochastic. An ambulance must contend with sudden traffic jams, road accidents, or pedestrian crowds. The ability of the evaluated algorithms (particularly PRM for replanning) to handle dynamic obstacle constraints ensures that the vehicle remains reliable even when the pre-planned route becomes blocked.

Integration with Smart Health Ecosystems These autonomous routing systems do not operate in isolation; they integrate naturally into broader **Smart City infrastructures** and **Digital Health platforms**. For instance, the ambulance’s navigation computer can communicate with traffic light control systems to clear a path (V2I - Vehicle to Infrastructure) or interface with the hospital's admission system to upload patient vitals en route. This holistic approach transforms the ambulance from a simple transport vehicle into an intelligent, connected node in the digital healthcare network

---

12. Conclusions and Future Work

Project Conclusions

This research project conducted a rigorous comparative analysis of three fundamental path-planning algorithms—Dijkstra-on-Grid, A (A-Star)*, and Probabilistic Roadmap (PRM)—to address the critical challenge of routing a Smart Robot-Ambulance in a crowded urban environment.

Based on the simulation results obtained from a 70 X 70 grid modeled after a dense city sector (Tel Aviv scenario), we derived the following conclusions:

- **Optimal Balance:** The *A algorithm** emerged as the most practical solution for real-time Digital Medicine applications. By leveraging the Octile Distance heuristic, it achieved paths identical in cost to Dijkstra's optimal solution but required **76% fewer node expansions** and significantly less computation time (~11.7 ms vs. 32.4 ms) .

- **The Cost of Guaranteed Optimality:** While **Dijkstra-on-Grid** is mathematically robust and guarantees the absolute shortest path, its exhaustive search nature makes it computationally prohibitive for large-scale, dynamic urban grids where rapid response is paramount.
- **Viability of Sampling:** The **PRM** algorithm demonstrated potential for extremely large or unstructured environments (like hospital campuses) due to its fast query times. However, its stochastic nature resulted in slightly sub-optimal path lengths, making it less suitable for critical "Code Blue" scenarios where every meter counts .

Future Research Directions

To further enhance the capability of autonomous medical vehicles, future work should focus on the following key areas:

1. Dynamic Obstacle Handling & Predictive Modeling:
   The current simulation treated obstacles as static or semi-static.

Future iterations should incorporate D Lite* or Anytime Repairing A (ARA)** algorithms to handle moving obstacles in real-time. This includes integrating predictive models that anticipate the movement of pedestrians and traffic congestion based on historical urban data .

2. Real-Time Re-planning:
   In a real-world scenario, a road might become blocked instantly (e.g., by an accident). The system must be capable of local replanning—calculating a new path from the current robot position without re-computing the entire graph from scratch. This capability is essential for maintaining the "Golden Hour" standard in emergency response.
3. Integration with GIS and V2X Communication:
   Moving beyond synthetic grids, the algorithms should be tested on real-world Geographic Information Systems (GIS) data, such as OpenStreetMap. Furthermore, integrating Vehicle-to-Everything (V2X) communication would allow the ambulance to interact with smart traffic lights, turning them green to clear a path automatically.
4. Multi-Vehicle Coordination (Swarm Intelligence):
   A mass-casualty incident often requires the deployment of a fleet of ambulances. Future research should explore Multi-Agent Path Finding (MAPF) to coordinate a swarm of robots, ensuring they do not collide with each other while navigating to different patients or converging on a single triage site.