**RAMDEOBABA UNIVERSITY, NAGPUR**

**Department of Computer Science and Engineering**

Session: 2025-26

Subject: Design and Analysis of Algorithms (DAA) Lab Project

III Semester

**LAB PROJECT REPORT**

**Name:** Pranav Rathi    **Roll No:** 48    **Batch:** B3    **Section:** A5

**Name:** Aditya Chapke    **Roll No:** 37    **Batch:** B3    **Section:** A5

**Name:** Hiten Borkar    **Roll No:** 15    **Batch:** B1    **Section:** A5

**RESCUE ROBO**

## 1. OBJECTIVES

### 1.1 Primary Objectives

1. To implement and visualize Greedy Best-First Search algorithm for optimal pathfinding in disaster response scenarios.
2. To apply backtracking optimization technique for solving the Traveling Salesman Problem (TSP) variant in rescue operations.
3. To develop an interactive web-based application demonstrating real-time algorithm execution.
4. To analyze time complexity of pathfinding and optimization algorithms in grid-based environments.

### 1.2 Secondary Objectives

1. To create an educational tool for understanding algorithmic concepts through emergency response simulations.
2. To provide intuitive visualization of algorithm execution phases and performance metrics.
3. To demonstrate practical applications of DAA concepts in life-critical situations.

## 2. INTRODUCTION

### 2.1 Problem Statement

In emergency rescue operations, time is the most critical factor. This project simulates an emergency response scenario where a rescue helicopter must navigate through a debris-filled disaster zone to reach multiple survivors and transport them to a hospital. The challenge is to find the optimal route that minimizes total travel distance while ensuring all survivors are rescued.

**Given:**

* A grid map representing a disaster zone
* Starting position (helicopter launch point)
* Multiple survivor locations
* Goal position (hospital/rescue center)
* Debris/obstacle positions that block paths

**Find:**

* The shortest complete path that visits all survivors and reaches the hospital
* Optimal order of survivor collection to minimize total distance
* Valid navigation routes that avoid all obstacles

### 2.2 Technology Stack

**Frontend:** React 18, TypeScript, Vite, TailwindCSS, HTML5 Canvas

**Algorithms:** Greedy Best-First Search (A\* variant), Backtracking for TSP, Manhattan Distance heuristic

### 2.3 Real-World Applications

* **Emergency Services:** Disaster relief operations, search and rescue missions
* **Logistics:** Package delivery route optimization, warehouse robot navigation
* **Robotics:** Autonomous robot navigation, inspection drones

## 3. ALGORITHMS/TECHNIQUES USED

### 3.1 Algorithm 1: Greedy Best-First Search (Pathfinding)

#### 3.1.1 Algorithm Name

Greedy Best-First Search with Manhattan Distance Heuristic

#### 3.1.2 Purpose

Find the shortest path between two points on a grid while avoiding obstacles.

#### 3.1.3 Algorithm Pseudocode

ALGORITHM: GreedyBestFirstSearch(grid, start, goal) INPUT: grid, start position, goal position OUTPUT: shortest path, distance, explored nodes BEGIN CREATE priority queue PQ CREATE set VISITED CREATE map PARENT CREATE map DISTANCE DISTANCE[start] ← 0 PQ.enqueue(start, heuristic(start, goal)) WHILE PQ is not empty DO current ← PQ.dequeue() IF current equals goal THEN RETURN RECONSTRUCT\_PATH(PARENT, start, goal) END IF IF current in VISITED THEN CONTINUE END IF ADD current to VISITED FOR EACH neighbor in GET\_NEIGHBORS(current) DO IF neighbor is walkable AND neighbor not in VISITED THEN tentative\_distance ← DISTANCE[current] + 1 IF tentative\_distance < DISTANCE[neighbor] THEN DISTANCE[neighbor] ← tentative\_distance PARENT[neighbor] ← current priority ← tentative\_distance + heuristic(neighbor, goal) PQ.enqueue(neighbor, priority) END IF END IF END FOR END WHILE RETURN empty\_path END FUNCTION heuristic(pos1, pos2) RETURN |pos1.row - pos2.row| + |pos1.col - pos2.col| END FUNCTION

#### 3.1.4 Algorithm Explanation

1. **Initialization:** Create priority queue ordered by f(n) = g(n) + h(n), where g(n) is actual distance and h(n) is heuristic (Manhattan distance).
2. **Exploration:** Extract node with lowest f(n) value, mark as visited.
3. **Goal Check:** If current node equals goal, reconstruct and return path.
4. **Neighbor Expansion:** For each unvisited neighbor, calculate tentative distance, update if shorter path found.
5. **Termination:** Success when goal reached, failure when queue empty.

#### 3.1.5 Manhattan Distance Heuristic

**Formula:** h(n) = |x₁ - x₂| + |y₁ - y₂|

**Properties:** Admissible (never overestimates), consistent (satisfies triangle inequality), computationally efficient O(1).

### 3.2 Algorithm 2: Backtracking for TSP Optimization

#### 3.2.1 Algorithm Name

Backtracking-based Traveling Salesman Problem (TSP) Solver

#### 3.2.2 Purpose

Find the optimal order to visit multiple survivors that minimizes total travel distance.

#### 3.2.3 Algorithm Pseudocode

ALGORITHM: FindOptimalSurvivorRoute(distances, start, survivors, goal) INPUT: precomputed distances, start, survivors array, goal OUTPUT: optimal order, minimum distance, permutations tested BEGIN bestDistance ← infinity bestOrder ← null permutationsTested ← 0 allPermutations ← GENERATE\_PERMUTATIONS(survivors) FOR EACH order in allPermutations DO permutationsTested ← permutationsTested + 1 distance ← CALCULATE\_ROUTE\_DISTANCE(start, order, goal, distances) IF distance < bestDistance THEN bestDistance ← distance bestOrder ← order END IF END FOR RETURN (bestOrder, bestDistance, permutationsTested) END FUNCTION CALCULATE\_ROUTE\_DISTANCE(start, survivorOrder, goal, distances) totalDistance ← 0 currentPos ← start FOR EACH survivor in survivorOrder DO totalDistance ← totalDistance + distances[currentPos][survivor] currentPos ← survivor END FOR totalDistance ← totalDistance + distances[currentPos][goal] RETURN totalDistance END

#### 3.2.4 Algorithm Explanation

1. **Preprocessing:** Compute shortest paths between all pairs using Greedy Best-First Search.
2. **Permutation Generation:** Generate all possible orderings of k survivors (k! permutations).
3. **Route Evaluation:** For each permutation, calculate total distance from precomputed matrix.
4. **Optimization Tracking:** Track and update best route found.
5. **Result Selection:** Return permutation with minimum total distance.

#### 3.2.5 Example Calculation

**Scenario:** Start S, Survivors A, B, C, Goal G

|  |  |  |  |
| --- | --- | --- | --- |
| **Route** | **Path** | **Distance Calculation** | **Total** |
| 1 | S → A → B → C → G | 5 + 4 + 5 + 4 | 18 |
| 2 | S → B → A → C → G | 3 + 4 + 6 + 4 | **17 (Optimal)** |
| 3 | S → C → B → A → G | 7 + 5 + 4 + 8 | 24 |

### 3.3 Three-Phase Algorithm Integration

#### Phase 1: PREPROCESSING

Compute all pairwise distances between {start, survivor₁, ..., survivorₖ, goal} using Greedy Best-First Search. Store in distance matrix.

#### Phase 2: OPTIMIZING

Use backtracking to test all k! permutations of survivor visit orders. Track best route with minimum total distance.

#### Phase 3: EXECUTING

Construct complete path by concatenating optimal route segments: start → survivor₁ → ... → survivorₖ → goal.

### 3.4 Diagrams

***Figure 1: Greedy Best-First Search Exploration Pattern***

*10×10 grid showing exploration wave radiating from start (helicopter) to goal (hospital)*

*- Light blue: Early explored nodes*

*- Dark blue: Late exploration*

*- Green path: Final shortest route avoiding debris obstacles*

***Figure 2: TSP Optimization Process***

*Complete permutation tree showing all 6 routes for 3 survivors*

*Bar chart comparing distances, with optimal route highlighted*

## 4. TIME COMPLEXITY ANALYSIS

### 4.1 Greedy Best-First Search Complexity

#### 4.1.1 Theoretical Analysis

Let V = number of cells in grid (V = rows × columns), E = number of edges (E ≈ 4V for 4-directional movement)

**Worst-Case Time Complexity:** O(E log V)

**Breakdown:**

* Priority Queue Operations: INSERT and EXTRACT-MIN both O(log V)
* Each node inserted/extracted once: O(V log V)
* Edge relaxation with priority queue update: O(E log V)
* For grid graphs where E = O(V): **O(V log V)**

**Space Complexity:** O(V) for priority queue, visited set, parent map, distance map

#### 4.1.2 Grid-Specific Performance

|  |  |  |  |
| --- | --- | --- | --- |
| **Grid Size** | **Cells (V)** | **Operations** | **Complexity** |
| 10 × 10 | 100 | 100 × log(100) ≈ 664 | O(100 log 100) |
| 20 × 20 | 400 | 400 × log(400) ≈ 3,460 | O(400 log 400) |
| 50 × 50 | 2,500 | 2,500 × log(2,500) ≈ 19,931 | O(2,500 log 2,500) |

### 4.2 Backtracking TSP Complexity

#### 4.2.1 Theoretical Analysis

Let k = number of survivors

**Worst-Case Time Complexity:** O(k!)

**Explanation:**

* Number of permutations of k survivors = k!
* For each permutation: Calculate route distance in O(k) time
* Total: O(k! × k) ≈ O(k!) for large k

**Space Complexity:** O(k) for recursive call stack

#### 4.2.2 Factorial Growth Impact

|  |  |  |
| --- | --- | --- |
| **Survivors (k)** | **Permutations (k!)** | **Time (1μs/check)** |
| 3 | 6 | 6 μs (Instant) |
| 5 | 120 | 120 μs (Instant) |
| 7 | 5,040 | 5.04 ms (Fast) |
| 8 | 40,320 | 40.32 ms (Noticeable) |
| 10 | 3,628,800 | 3.63 s (Slow) |

**Note:** This project limits survivors to ≤ 8 for practical performance.

### 4.3 Complete Algorithm Complexity

**Combined Time Complexity:**

Total = Preprocessing + Optimization + Execution

= **O(k² × V log V + k!)**

where:

* k² pairwise distance computations, each O(V log V)
* k! permutations tested in optimization phase
* For small grids and k ≤ 8: k! dominates when k ≥ 7

## 5. RESULTS

### 5.1 Test Scenarios

#### Test Case 1: Simple Rescue (3 Survivors)

**Configuration:**

* Grid Size: 10×10
* Helicopter: (1, 1)
* Survivors: (3, 3), (5, 5), (7, 7)
* Hospital: (9, 9)
* Debris: Moderate obstacles

**Results:**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Total Distance | 16 grid units |
| Nodes Explored | 247 |
| Routes Tested | 6 (3!) |
| Execution Time | ~1.8 seconds |

***Screenshot 1: Editor View - Simple Scenario***

*10×10 grid with 3 survivors (🆘), helicopter (🚁), hospital (🏥), and debris (🧱)*

***Screenshot 2: Simulation View - Exploration Phase***

*Purple/blue gradient showing explored nodes, cyan path forming*

***Screenshot 3: Results View - Optimal Path***

*Final path highlighted with metrics panel showing distance: 16, survivors: 3, routes tested: 6*

#### Test Case 2: Complex Rescue (6 Survivors)

**Configuration:**

* Grid Size: 10×10
* Survivors: 6 scattered positions
* Dense debris creating maze-like structure

**Results:**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Total Distance | 34 grid units |
| Nodes Explored | 1,842 |
| Routes Tested | 720 (6!) |
| Execution Time | ~4.2 seconds |

#### Test Case 3: Maximum Survivors (8 Survivors)

**Results:**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Total Distance | 52 grid units |
| Nodes Explored | 3,456 |
| Routes Tested | 40,320 (8!) |
| Execution Time | ~12.6 seconds |

### 5.2 Performance Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Survivors** | **Distance** | **Nodes** | **Routes** | **Time (s)** |
| 1 | 3 | 16 | 247 | 6 | 1.8 |
| 2 | 6 | 34 | 1,842 | 720 | 4.2 |
| 3 | 8 | 52 | 3,456 | 40,320 | 12.6 |

**Key Observations:**

* Execution time scales exponentially with survivor count due to k! growth
* Algorithm remains responsive up to 8 survivors on modern hardware
* Nodes explored increases with grid complexity and obstacle density
* Optimal path successfully found in all test cases

### 5.3 User Interface Screenshots

***Screenshot 4: Home Page***

*Emergency alert banner with dark theme, "Start Designing" and "Learn More" buttons*

***Screenshot 5: Map Editor - Tool Palette***

*Tool selection panel showing: Helicopter (🚁), Hospital (🏥), Survivor (🆘), Debris (🧱), Erase (🗑️)*

*Grid with light indigo background, placed elements visible*

***Screenshot 6: Live Simulation - Phase Indicators***

*Four phases shown: 🔍 Preprocessing, ⚙️ Optimizing, ▶️ Executing, ✨ Complete*

*Metrics panel displaying: Distance, Survivors, Time, Current Phase*

***Screenshot 7: Results Dashboard***

*Journey summary with survivor collection order (Survivor 1, 2, 3...)*

*Performance metrics cards showing final statistics*

*Optimal rescue sequence visualization*

### 5.4 Discussion

**Effectiveness of Greedy Best-First Search:**

The algorithm successfully found shortest paths in all test cases, efficiently navigating around debris obstacles. The Manhattan distance heuristic proved effective for grid-based navigation, significantly reducing search space compared to uninformed search.

**TSP Optimization Performance:**

Backtracking exhaustively tested all permutations, guaranteeing optimal survivor collection order. However, factorial growth became apparent at 8 survivors (40,320 routes), taking ~12 seconds. This demonstrates the importance of algorithm choice for scalability.

**Visualization Impact:**

Real-time visualization of exploration phases helped users understand algorithm behavior. The color-coded exploration wave and live metrics panel made abstract concepts tangible and engaging.

## 6. CONCLUSION AND FUTURE SCOPE

### 6.1 Conclusion

This project successfully implemented and visualized advanced pathfinding and optimization algorithms in an emergency rescue context. Key achievements include:

1. **Algorithm Implementation:** Successfully implemented Greedy Best-First Search achieving O(V log V) pathfinding performance and backtracking TSP solver with O(k!) complexity.
2. **Educational Value:** Created intuitive visualizations demonstrating algorithm decision-making processes, making abstract concepts accessible through real-world scenarios.
3. **Performance Analysis:** Demonstrated practical impact of time complexity through empirical testing, showing exponential growth of factorial algorithms.
4. **Technical Excellence:** Built responsive web application using modern technologies (React 18, TypeScript) with type-safe implementation.

**Learning Outcomes:**

* Deep understanding of graph algorithms and heuristic search techniques
* Practical experience with complexity analysis (time and space)
* Recognition of trade-offs between optimality and performance
* Hands-on implementation of data structures (priority queues, hash maps)

The project reinforces that while brute-force approaches guarantee optimal solutions, their exponential growth limits practical scalability. For k≤8 survivors, our implementation performs admirably, but larger instances require more sophisticated techniques.

### 6.2 Future Scope

#### 6.2.1 Algorithm Enhancements

**1. Advanced TSP Solvers:**

* **Held-Karp Dynamic Programming:** O(2^k × k²) complexity, enabling k≤20 survivors while maintaining optimality
* **Branch-and-Bound Pruning:** Eliminate suboptimal branches early, reducing effective k! factor
* **Approximation Algorithms:** Christofides algorithm (1.5-approximation) for larger instances

**2. Heuristic Methods:**

* Genetic Algorithms for population-based optimization
* Simulated Annealing for probabilistic search
* Ant Colony Optimization for distributed problem solving

**3. Pathfinding Variants:**

* A\* with consistent heuristic for comparison
* Bidirectional search for large grids
* Jump Point Search for performance optimization

#### 6.2.2 Feature Additions

**1. Enhanced Map Editor:**

* Different terrain types with varying movement costs
* Weighted survivors (priority levels: critical, standard, stable)
* Map import/export functionality (JSON, preset scenarios)

**2. Advanced Simulation:**

* Multiple rescue vehicles with team coordination
* Dynamic obstacles shifting during mission
* Time constraints and survivor health degradation

**3. Visualization Improvements:**

* 3D terrain rendering using Three.js
* Side-by-side algorithm comparison
* Animation export as video

#### 6.2.3 Technical Improvements

**1. Performance Optimization:**

* Web Workers for background computation
* WebAssembly compilation for critical algorithms (10-100× speedup potential)
* Memoization and caching for repeated computations

**2. Extended Applications:**

* Real-world city map integration via GIS systems
* Machine learning for route prediction
* Multi-objective optimization (distance, time, fuel consumption)

### 6.3 Closing Remarks

This project demonstrates that complex algorithms can be made accessible through thoughtful visualization and real-world contextualization. By framing pathfinding and TSP as an emergency rescue mission, we've shown how academic concepts directly impact life-critical situations.

The exponential growth of k! serves as a powerful reminder that algorithmic efficiency matters. While modern computers are incredibly fast, poor algorithm choice can make even simple problems intractable. Our project makes this abstract concept visceral—watching 40,320 routes being tested for just 8 survivors is far more impactful than reading "O(k!)" in a textbook.

We hope **Rescue Robo** serves both as an effective learning tool and a demonstration of how computer science can contribute to solving humanity's most pressing challenges—one optimized path at a time.

## 7. REFERENCES

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
2. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.
3. Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (2006). The Traveling Salesman Problem: A Computational Study. Princeton University Press.
4. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.
5. React Documentation. (2024). React 18. Retrieved from https://react.dev/
6. TypeScript Documentation. (2024). TypeScript Handbook. Retrieved from https://www.typescriptlang.org/docs/

**--- END OF REPORT ---**

Date of Submission: October 31, 2025