Detailed Usage Instructions

```markdown
# AI Search Algorithms Lab - Usage Instructions

## Table of Contents
1. [Quick Start](#quick-start)
2. [Installation](#installation)
3. [Running the Program](#running-the-program)
4. [Step-by-Step Guide](#step-by-step-guide)
5. [Graph Types](#graph-types)
6. [Algorithms](#algorithms)
7. [Heuristics](#heuristics)
8. [Output Metrics](#output-metrics)
9. [Troubleshooting](#troubleshooting)

## Quick Start

### For Immediate Use:
```bash
# 1. Install requirements
pip install networkx matplotlib pandas numpy seaborn

# 2. Run the program
python src/main.py

# 3. Follow these steps in the program:
#    - Choose option 1 (Select Graph)
#    - Choose option 1 (Kansas cities)
#    - Choose option 2 (Single Algorithm)
#    - Start: Wichita
#    - Goal: Topeka
#    - Algorithm: A*
```

Installation

Prerequisites
- Python 3.8 or higher
- pip (Python package manager)

Step 1: Install Required Packages
```bash
pip install networkx matplotlib pandas numpy seaborn

```

Or use the requirements file:
```bash
pip install -r requirements.txt
```

Step 2: Verify Installation
```python
python -c "import networkx, matplotlib, pandas, numpy, seaborn; print('All packages installed successfully!')"
```

Running the Program

Basic Command
```bash
python src/main.py
```

Program Structure
```
ai_search_lab/
├── src/
│   ├── main.py            # Main program
│   ├── algorithms.py      # BFS, DFS, IDDFS, Greedy, A*
│   ├── graph_loader.py    # Load Kansas cities data
│   ├── graph_generator.py # Create random graphs & grids
│   ├── heuristics.py      # Distance calculation methods
│   ├── benchmark.py       # Performance comparison
│   └── visualization.py   # Graph drawing
├── data/
│   ├── Adjacencies.txt    # City connections
│   └── coordinates.csv    # City locations
└── README.md
```

Step-by-Step Guide

Step 1: Launch the Program
```bash
python src/main.py
```
```

You'll see the main menu:
```
🎯 AI Search Algorithm Lab
   Implement, Visualize, and Compare Search Algorithms

============================================================
MAIN MENU
============================================================
1. Select/Change Graph
   Current: No graph selected
2. Single Algorithm Search (with visualization)
3. Batch Algorithm Comparison (benchmarking)
4. Exit
============================================================
Choose option (1-4):
```

Step 2: Select a Graph (Option 1)

Option 1.1: Kansas Cities (Recommended for starters)
```
GRAPH SELECTION
==================================================
1. Use preset graph (Kansas cities)
2. Generate random graph
3. Generate grid world
4. Use current graph
5. Back to main menu
Choose option (1-5): 1
```
- Loads real geographic data of 46 Kansas cities
- Edge weights = actual Euclidean distances
- Great for testing and demonstration

Option 1.2: Random Graph
```
Choose option (1-5): 2
Number of nodes (default 20): 30
Branching factor (default 1.5): 2.0
Random seed (default 42): 123
```
- Creates connected graphs with random connections
- Customizable size and complexity
- Reproducible with seeds

Option 1.3: Grid World
```
Choose option (1-5): 3
Grid size (default 10): 15
Obstacle density (0-1, default 0.2): 0.3
Connectivity (4/8, default 4): 4
Weighted edges? (y/n, default n): n
Random seed (default 42): 456
```
- Creates maze-like environments
- Adjustable obstacle density
- 4-connected (up/down/left/right) or 8-connected (with diagonals)

Step 3: Choose Search Mode

Option 2: Single Algorithm Search
Best for understanding how one algorithm works.

Example Session:
```
SINGLE ALGORITHM MODE
==================================================
Start node: Wichita
Goal node: Topeka

Available algorithms:
1. BFS (Breadth-First Search)
2. DFS (Depth-First Search)
3. IDDFS (Iterative Deepening DFS)
4. Greedy Best-First Search
5. A* Search
Choose algorithm (1-5, default 5): 5

Available heuristics:
1. Euclidean distance
2. Manhattan distance
3. Chebyshev distance
4. Zero heuristic (uniform cost)
Choose heuristic (1-4, default 1): 1
```

Output:
```
```

RESULTS: A* from Wichita to Topeka
==
Path found: Yes
Path: Wichita -> Andover -> Towanda -> El_Dorado -> Hillsboro -> Marion -> Abilene ->
Junction_City -> Manhattan -> Topeka
Path length: 10 nodes
Path cost: 2.8463
Nodes expanded: 10
Solution depth: 9
Runtime: 0.000723 seconds
Peak memory: 145,632 bytes
Max frontier size: 8
```

#### Option 3: Batch Algorithm Comparison
Best for comparing performance across all algorithms.

**Example Session:**
```
BATCH COMPARISON MODE
==
Start node: Node_0
Goal node: Node_25
Number of runs per algorithm (default 5): 5

Available heuristics for informed searches:
1. Euclidean distance
2. Manhattan distance
3. Chebyshev distance
4. Zero heuristic (uniform cost)
Choose heuristic (1-4, default 1): 1
```

**Output includes:**
- Comparison table with mean ± standard deviation
- Success rates for each algorithm
- Runtime, memory, and node expansion statistics
- Visual charts comparing performance
- Option to save results to CSV

## Graph Types Detailed

### 1. Kansas Cities Graph
- **Nodes**: 46 actual cities in southern Kansas
```

- **Edges**: Real road connections from the dataset
- **Weights**: Euclidean distance between coordinates
- **Use Case**: Testing with real-world geographic data

Sample Cities: Wichita, Topeka, Manhattan, Salina, Hutchinson, Emporia, Newton, McPherson, El_Dorado, Abilene

2. Random Graphs
- **Parameters**:
  - `Number of nodes`: Graph size (10-100+)
  - `Branching factor`: Average connections per node (1.0-3.0)
  - `Random seed`: For reproducible results
- **Use Case**: Testing algorithm scalability

3. Grid Worlds
- **Parameters**:
  - `Grid size`: N x N grid (5-20+)
  - `Obstacle density`: 0.0-1.0 (percentage blocked)
  - `Connectivity`: 4 (Manhattan) or 8 (Diagonal)
  - `Weighted`: Uniform or random edge costs
- **Use Case**: Pathfinding in maze-like environments

Algorithms Explained

1. BFS (Breadth-First Search)
- **Strategy**: Explore level by level
- **Completeness**: ✅ Yes
- **Optimality**: ✅ Yes (uniform cost)
- **Best for**: Finding shortest paths in unweighted graphs

2. DFS (Depth-First Search)
- **Strategy**: Go deep first, then backtrack
- **Completeness**: ❌ No (can loop infinitely)
- **Optimality**: ❌ No
- **Best for**: Memory-constrained environments

3. IDDFS (Iterative Deepening DFS)
- **Strategy**: DFS with increasing depth limits
- **Completeness**: ✅ Yes
- **Optimality**: ✅ Yes (uniform cost)
- **Best for**: When depth is unknown, memory concerns

4. Greedy Best-First Search
- **Strategy**: Always expand most promising node by heuristic

- **Completeness**: ❌ No
- **Optimality**: ❌ No
- **Best for**: Quick, good-enough solutions

5. A* Search
- **Strategy**: Combine cost-so-far and heuristic estimate
- **Completeness**: ✅ Yes
- **Optimality**: ✅ Yes (with admissible heuristic)
- **Best for**: Optimal pathfinding with good heuristics

Heuristics Available

1. Euclidean Distance
- **Formula**: $\sqrt{(\Delta x^2 + \Delta y^2)}$
- **Admissible**: ✅ Yes
- **Best for**: Geographic graphs with coordinates

2. Manhattan Distance
- **Formula**: $|\Delta x| + |\Delta y|$
- **Admissible**: ✅ Yes (for 4-connected grids)
- **Best for**: Grid-based movement

3. Chebyshev Distance
- **Formula**: $\max(|\Delta x|, |\Delta y|)$
- **Admissible**: ✅ Yes (for 8-connected grids)
- **Best for**: Grids with diagonal movement

4. Zero Heuristic
- **Formula**: $h(n) = 0$
- **Admissible**: ✅ Yes
- **Effect**: Turns A* into Uniform Cost Search

Output Metrics

Primary Metrics:
- **Path Found**: Whether a solution exists
- **Path Cost**: Total edge weight of solution path
- **Path Length**: Number of nodes in solution
- **Nodes Expanded**: Total nodes removed from frontier
- **Runtime**: Execution time in seconds
- **Memory Usage**: Peak memory consumption in bytes
- **Solution Depth**: Path length - 1
- **Frontier Size**: Maximum nodes in queue/stack at once

Benchmarking Metrics:
- **Success Rate**: Percentage of successful runs
- **Mean ± Std Dev**: Statistical performance measures
- **Comparative Charts**: Visual performance comparison

Common Usage Examples

Example 1: Classroom Demonstration
```bash
python src/main.py
# 1 → 1 (Kansas cities)
# 2 (Single algorithm)
# Start: Wichita
# Goal: Topeka
# Algorithm: Compare BFS vs A*
```

Example 2: Algorithm Research
```bash
python src/main.py
# 1 → 2 (Random graph, 50 nodes, branching=2.0)
# 3 (Batch comparison)
# Runs all 5 algorithms 10 times each
# Exports CSV for further analysis
```

Example 3: Pathfinding Testing
```bash
python src/main.py
# 1 → 3 (Grid world, 15x15, 30% obstacles)
# 2 (Single algorithm)
# Tests A* with different heuristics
```

Troubleshooting

Common Issues:

1. "Module not found" error
```bash
# Solution: Install missing packages
pip install networkx matplotlib pandas numpy seaborn
```

2. "City not found" error
```bash
# Solution: Use exact city names from the list
# Check available cities when prompted
# Use underscores for multi-word names: South_Haven
```

3. Visualization not working
```bash
# Solution: Ensure matplotlib backend is proper
# Try: python -c "import matplotlib; print(matplotlib.get_backend())"
```

4. Program crashes with large graphs
```bash
# Solution: Reduce graph size or use DFS/IDDFS
# For 100+ nodes, avoid BFS due to memory usage
```

5. No path found
```bash
# Solutions:
# - Check if graph is connected
# - Try different start/goal nodes
# - Reduce obstacle density in grid worlds
# - Use BFS or A* for guaranteed completeness
```

Getting Help:

1. **Check available nodes** when graph is loaded
2. **Start with Kansas cities** for reliable testing
3. **Use simple parameters** first, then increase complexity
4. **Save random seeds** to reproduce interesting cases

Advanced Features

Reproducible Research:
- All random generation uses seeds
- Save benchmark results to CSV
- Use same seeds to compare algorithm improvements

Performance Tuning:
- Adjust IDDFS max depth for deeper graphs

- Try different heuristics for specific domains
- Use batch mode for statistical significance

Extension Points:
- Add new algorithms in `algorithms.py`
- Create custom heuristics in `heuristics.py`
- Design new graph generators in `graph_generator.py`

Need more help? Run the program and explore - the interface provides guidance at every step!
```

These usage instructions provide everything needed to:
1. **Install and run** the program
2. **Understand all features** and options
3. **Follow step-by-step examples**
4. **Troubleshoot common issues**
5. **Use advanced features** for research
```