

Practical

explanation of the algorithms used in the Cairo Transportation Network Optimization project, along with a comparison table and the libraries involved:

**Algorithms Used in the Project**

**1. Minimum Spanning Tree (MST) - Network Design**

- ****Purpose****: Designs an optimal road network by connecting all neighborhoods with the least total construction cost.
- ****Algorithm****: Kruskal's or Prim's algorithm
- ****Implementation****:
 - Connects all nodes (neighborhoods/facilities) with minimum total edge weight (construction cost)
 - Prioritizes roads based on population density and construction cost
- ****Use Case****: Planning new road infrastructure

**2. Dijkstra's Algorithm - Traffic Flow Optimization**

- ****Purpose****: Finds the shortest path between two locations considering traffic conditions.
- ****Algorithm****: Dijkstra's shortest path

-

Implementation:

- Weights edges based on real-time traffic data
- Optimizes for travel time during different periods (morning/evening peaks)
- **Use Case:** Route planning for commuters

3. A* Algorithm - Emergency Response Routing

- **Purpose:** Finds the fastest route for emergency vehicles.
- **Algorithm:** A* search algorithm
- **Implementation:**
 - Uses heuristic (Euclidean distance) to guide search toward destination
 - Prioritizes routes with least congestion and shortest distance
- **Use Case:** Ambulance/fire truck routing

4. Dynamic Programming (DP) - Public Transit Optimization

- **Purpose:** Optimizes public transit schedules and routes.
- **Algorithm:** Dynamic programming (Bellman equation)
- **Implementation:**
 - Solves transit scheduling as a sequence of optimal decisions
 - Minimizes waiting time and transfer penalties
- **Use Case:** Metro/bus schedule optimization

5. Greedy Algorithm - Traffic Signal Optimization

- **Purpose**: Optimizes traffic light timing at intersections.
- **Algorithm**: Greedy approach
- **Implementation**:
 - Makes locally optimal choices at each intersection
 - Adjusts signal timing based on real-time traffic flow
- **Use Case**: Reducing congestion at busy intersections

Comparison Table of Algorithms

Cairo Transportation Network Optimization Dashboard

Comparison Table of Algorithms

Algorithm	Type	Time Complexity	Use Case	Strengths
MST (Kruskal's)	Graph Theory	$O(E \log V)$	Road network design	Minimal cost, cor
Dijkstra's	Path Finding	$O(V^2)$	Shortest path routing	Guaranteed short
A*	Heuristic Search	$O(b^d)$	Emergency vehicle routing	Faster than Dijkst
Dynamic Programming	Optimization	$O(n^2)$	Transit scheduling	Optimal substruc
Greedy Algorithm	Optimization	$O(n \log n)$	Traffic signal timing	Fast, simple imple

Libraries Used

Cairo Transportation Network Optimization Dashboard

Libraries Used

Library	Purpose	Key Features Used
Streamlit	Web app framework	UI components, layout, caching
Folium	Interactive maps	Layer controls, markers, polylines
Pandas	Data manipulation	DataFrames, CSV I/O, statistics
Plotly	Interactive visualizations	Charts, graphs, statistical plots
Branca	Color mapping for Folium	Linear colormaps for road conditions
NumPy	Numerical operations	Mathematical calculations
Streamlit-Folium	Embeds Folium maps in Streamlit	<code>folium_static()</code> for map rendering

Key Insights

1. **MST** is best for infrastructure planning where cost minimization is critical.
2. **Dijkstra's/A** excel in pathfinding but trade off speed vs. optimality.
3. **DP** handles transit scheduling well but requires more computational resources.
4. **Greedy algorithms** provide quick solutions for traffic signals but may not be globally optimal.
5. The libraries work together to:
 - Streamlit for the UI
 - Folium for geospatial visualization
 - Pandas/NumPy for data processing
 - Plotly for statistical charts

This combination of algorithms and libraries creates a comprehensive transportation optimization system for Cairo.

Code

Import Statements

```
import streamlit as st
import folium
import pandas as pd
from streamlit_folium import folium_static
import branca.colormap as cm
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
```

- **streamlit**: Creates the web app interface
- **folium**: For interactive map visualization
- **pandas**: Data manipulation and analysis
- **streamlit_folium**: Integrates folium maps with streamlit
- **branca.colormap**: For creating color scales on the map
- **numpy**: Numerical operations
- **plotly.express/go**: For creating interactive charts

Page Configuration

```
st.set_page_config(
    page_title="Cairo Transportation Network Optimization",
    page_icon="🚗",
    layout="wide",
    initial_sidebar_state="expanded"
)
```

- Sets up the web page with a title, car icon, wide layout, and expanded sidebar

Custom CSS

```
st.markdown("""
    <style>
    .main {
        padding: 0rem 1rem;
    }
    .stMetric {
```

```

        background-color: #f0f2f6;
        padding: 1rem;
        border-radius: 0.5rem;
    }
</style>
""" , unsafe_allow_html=True)

```

- Adds custom styling to the app (padding for main content and metric cards)

Title and Description

```

st.title("Cairo Transportation Network Optimization")
st.markdown("""...""")

```

- Displays the main title and a description of the dashboard's purpose

Data Loading Function

```

@st.cache_data
def load_data():
    try:
        existing_roads = pd.read_csv('existing_roads.csv')
        potential_roads = pd.read_csv('potential_roads.csv')
        # ... other CSV files

        # Check for empty DataFrames
        if existing_roads.empty:
            st.error("existing_roads.csv is empty")
            return None, None, None, None, None, None
        # ... similar checks for other files

        return existing_roads, potential_roads, public_transit, traffic_flow,
        facilities, neighborhoods
    except FileNotFoundError as e:
        st.error(f"Could not find required CSV file: {str(e)}")
        return None, None, None, None, None, None
    except Exception as e:
        st.error(f"Error loading data: {str(e)}")
        return None, None, None, None, None, None

```

- Decorated with `@st.cache_data` to cache loaded data
- Attempts to load 6 CSV files with error handling

- Checks for empty DataFrames and file existence

Main Try-Except Block

```
try:
    # Load data
    existing_roads, potential_roads, public_transit, traffic_flow, facilities
    , neighborhoods = load_data()

    if any(df is None for df in [existing_roads, potential_roads, public_transit, traffic_flow, facilities, neighborhoods]):
        st.error("Failed to load data. Please check the CSV files.")
        st.stop()

    # Debug information
    st.write("Data loaded successfully:")
    st.write(f"Existing roads: {len(existing_roads)}")
    # ... similar for other datasets
```

- Main execution block wrapped in try-except
- Shows debug info about loaded data

Map Setup

```
# Create feature groups for layer control
existing_roads_group = folium.FeatureGroup(name='Existing Roads')
# ... other feature groups

# Algorithm selection in sidebar
algorithm = st.sidebar.selectbox(...)

# Function to get coordinates with error handling
def get_coordinates(loc_id, df):
    # ... implementation

# Create map centered on Cairo
try:
    center_lat = neighborhoods['y_coordinate'].mean()
    center_lon = neighborhoods['x_coordinate'].mean()
except:
    center_lat, center_lon = 30.0444, 31.2357 # Default Cairo coordinates
```



```
m = folium.Map(location=[center_lat, center_lon], zoom_start=10)
```

- Sets up folium map layers and controls
- Creates coordinate helper function
- Centers map on Cairo coordinates

Data Visualization on Map

```
# Add existing roads to map
for _, row in existing_roads.iterrows():
    from_lat, from_lon = get_coordinates(...)
    to_lat, to_lon = get_coordinates(...)
    folium.PolyLine(...).add_to(existing_roads_group)

# Similar blocks for:
# - Potential roads
# - Facilities
# - Neighborhoods
```

- Visualizes each dataset on the map with appropriate styling

Algorithm-Specific Visualizations

```
if algorithm == "Network Design (MST)":
    # MST visualization
    for _, row in potential_roads.iterrows():
        if row['construction_cost'] <= max_cost:
            folium.PolyLine(...).add_to(optimization_group)

elif algorithm == "Traffic Flow (Dijkstra)":
    # Dijkstra's algorithm visualization
    folium.PolyLine(...).add_to(optimization_group)

# Similar blocks for other algorithms
```

- Implements different transportation algorithms with visualizations

Statistical Analysis Section

```
st.subheader("Statistical Analysis")
stat_tab1, stat_tab2, stat_tab3, stat_tab4 = st.tabs([...])
```

```

with stat_tab1: # Road Network Analysis
    # Road condition distribution chart
    fig = px.bar(...)
    st.plotly_chart(fig)

    # Road length distribution chart
    fig = px.histogram(...)
    st.plotly_chart(fig)

# Similar blocks for other analysis tabs

```

- Creates interactive statistical visualizations using Plotly

Algorithm Benchmarking

```

st.subheader("Algorithm Performance Analysis")
bench_tab1, bench_tab2, bench_tab3, bench_tab4 = st.tabs([...])

with bench_tab1: # MST Analysis
    # Theoretical complexity chart
    fig = px.line(...)

    # Performance metrics chart
    fig = px.bar(...)

# Similar blocks for other algorithms

```

- Shows performance metrics for each algorithm

Data Tables

```

with st.expander("View Data Tables"):
    tab1, tab2, tab3, tab4, tab5, tab6 = st.tabs([...])

    with tab1:
        st.dataframe(existing_roads)
    # ... other tabs

```

- Displays raw data in expandable tabs

Network Statistics

```

st.subheader("Network Statistics")

```

```

coll1, coll2, coll3 = st.columns(3)

with coll1:
    st.metric("Total Roads", len(existing_roads))
    # ... other metrics

# Similar for other columns

```

- Shows key metrics in a 3-column layout

Error Handling

```

except Exception as e:
    st.error(f"An error occurred: {str(e)}")
    st.info("Please make sure all the required CSV files are present in the c
orrect format.")

```

- Catches and displays any unhandled exceptions

This code creates a comprehensive transportation optimization dashboard for Cairo with:

1. Interactive map visualization
2. Multiple algorithm implementations
3. Statistical analysis
4. Performance benchmarking
5. Data exploration capabilities

Theoretical

Here's a simplified explanation of Team 7's A* algorithm project for emergency routing in Cairo:

**Simple Explanation (Computer Science Project)**

****Project Goal:****

Find the fastest way for ambulances to reach hospitals in Cairo's busy traffic using smart computer algorithms.

****Why A*?***

It's like a smart GPS that:

1. Knows the straight-line distance to the hospital (like a crow flies)
2. Considers real traffic conditions
3. Finds the best balance between speed and accuracy

****How It Works:****

The algorithm uses this simple formula to decide which roads to take:

...

Total Cost = (Distance Traveled) + (Estimated Distance Left)

...

****Real-World Adjustments:****

1. ****Traffic Matters:**** Roads cost more during rush hour (morning/evening)
2. ****Smart Guessing:**** Uses straight-line distance to the hospital to avoid checking useless routes
3. ****Road Rules:**** All roads work both ways, and bad road conditions increase travel time

****Code Summary (Simplified):****

```
```python
```

```
def find_emergency_route(start, hospital):
```

```
 open_roads = [start]
```

```
 best_path = {}
```

```
 while open_roads:
```

```
 current = pick_road_closest_to_hospital(open_roads)
```

```
 if current == hospital:
```

```
 return rebuild_path(best_path, hospital)
```

```
 for neighbor in all_connecting_roads:
```

```
 new_cost = current_cost + road_length + traffic_penalty
```

```
 if new_cost < best_known_cost:
```

**update\_costs\_and\_continue\_search()**

...

**\*\*Test Results:\*\***

- Morning Rush: Found a 10km detour through downtown to avoid traffic
- Afternoon: Took a longer 24.5km route when side roads were jammed
- Nighttime: Used the shortest 1.5km path when roads were empty

**\*\*Why Better Than Other Options:\*\***

| Algorithm | Problem |

|-----|-----|

| Dijkstra | Checks too many unnecessary roads |

| BFS | Ignores traffic and road lengths |

| Bellman-Ford | Too slow for big cities |

**\*\*Key Takeaways:\*\***

1. A\* works like a human - uses shortcuts and avoids traffic
2. Can handle Cairo's 22 million people and complex roads
3. Faster than other methods when every second counts

**\*\*Perfect For:\*\***


**Emergency systems where getting to hospitals fast saves lives,  
especially in crowded cities like Cairo.**

## Data set

Here's a simple breakdown of the transportation data for Cairo:


### ### **\*\*1. Neighborhoods & Districts\*\*** (15 locations)

- Each has:

 **\*\*ID, Name\*\*** (e.g., `1, Maadi`)

 **\*\*Population\*\*** (e.g., `250,000`)

 **\*\*Type\*\*** (Residential/Business/Mixed)

 **\*\*Coordinates\*\*** (X=longitude, Y=latitude)

**\*\*Example:\*\***

`3, Downtown Cairo, 100000, Business, 31.24, 30.04`

→ Small business district with 100k people at (31.24, 30.04).

---

### ### **\*\*2. Important Facilities\*\*** (10 key places)

- Includes:

 **\*\*Hospitals\*\*** (Qasr El Aini, Maadi Military)



✈️ **\*\*Airport\*\*** (Cairo International)

🚉 **\*\*Transit Hubs\*\*** (Ramses Railway Station)

🎓 **\*\*Universities\*\*** (Cairo, Al-Azhar)

**\*\*Example:\*\***

`F9, Qasr El Aini Hospital, Medical, 31.23, 30.03`

→ Major hospital at (31.23, 30.03).

---

### ### **\*\*3. Road Network\*\***

#### #### **\*\*Existing Roads\*\*** (30 routes)

- Connects neighborhoods/facilities:

📍 **\*\*FromID → ToID\*\*** (e.g., `1 → 3` = Maadi to Downtown)

📏 **\*\*Distance\*\*** (km)

🚗 **\*\*Capacity\*\*** (vehicles/hour)

⚠️ **\*\*Condition\*\*** (1-10 scale, 10=best)

**\*\*Example:\*\***

`1, 3, 8.5, 3000, 7`

→ Maadi to Downtown: 8.5km, handles 3000 cars/hour, decent condition (7/10).

### #### **\*\*Potential New Roads\*\*** (15 proposals)

- Includes:

 **\*\*Construction Cost\*\*** (Millions EGP)

 **\*\*Future Capacity\*\***

**\*\*Example:\*\***

`1, 4, 22.8, 4000, 450`

→ Proposed Maadi-to-New-Cairo road: 22.8km, costs 450M EGP.

---


### ### **\*\*4. Traffic Flow\*\***

- Shows **\*\*congestion\*\*** on existing roads by time:

 **\*\*Morning Peak\*\*** (Worst traffic)

 **\*\*Afternoon\*\***

 **\*\*Evening Peak\*\***

 **\*\*Night\*\* (Least traffic)**

**\*\*Example:\*\***

`1-3, 2800, 1500, 2600, 800`

→ Maadi-Downtown road:

- Morning: 2800 cars/hour (near max capacity!)
- Night: Only 800 cars/hour.

---

**### \*\*5. Public Transport\*\***


**#### \*\*Metro Lines\*\* (3 routes)**

-  **\*\*Stations\*\*** (Lists neighborhood IDs)

**\*\*Example:\*\***

`M1: "12,1,3,F2,11"` = Helwan → Maadi → Downtown → Ramses Station → Shubra.

**#### \*\*Bus Routes\*\* (10 routes)**

-  **\*\*Stops\*\*** (e.g., `"1,3,6,9"` = Maadi → Downtown → Zamalek → Mohandessin)

- **📊 \*\*Daily Passengers\*\*** (e.g., 35,000 for B1).

#### **#### \*\*Demand Data\*\***

- Shows popular routes (e.g., `F2 → 11` = 25,000 daily passengers from Ramses Station to Shubra).

---

#### **### \*\*Key Insights\*\***

1. **\*\*Busiest Areas\*\***: Nasr City (500k pop), Giza (550k pop).
2. **\*\*Critical Roads\*\***:
  - Downtown (ID 3) is a major hub with 8 connections.
  - Road `7-8` (6th October ↔ Giza) is long (24.5km) but well-maintained.
3. **\*\*Traffic Hotspots\*\***:
  - Morning: Roads near business districts (Downtown, Mohandessin).
  - Evening: Residential ↔ Commercial routes.
4. **\*\*Transit Gaps\*\***:
  - New Administrative Capital (ID 13) has few connections.

- Proposed road `13 ↔ 3` would cost 1.1B EGP but improve access.

---

### ### **\*\*Why This Matters\*\***

This data helps:

- 🚑 **\*\*Emergency vehicles\*\*** find fastest routes (using A\* algorithm).
- 🚦 **\*\*City planners\*\*** decide where to build new roads/metro lines.
- 🚗 **\*\*Drivers\*\*** avoid traffic jams during rush hours.

\*(Think of it like a "Google Maps database" specifically for Cairo!)\* 🌍💡

**Written by: Yusuf Yasser wardany**