Campus Maintenance Optimization System

Intelligent Route Planning for Facility Management

Project Overview

Our Campus Maintenance Optimization System provides intelligent route planning for cleaning and maintenance staff, ensuring efficient coverage of campus facilities while prioritizing high-traffic and important locations.

The system uses:

- Dynamic priority calculations based on multiple factors
- Modified Dijkstra's algorithm for intelligent path finding
- Historical data tracking and predictive scheduling

Key Features

- **Intelligent Route Planning**: Generates optimal daily routes based on building priorities
- **Dynamic Priority Calculation**: Considers building importance, cleanliness status, visit history, and time since last cleaning
- **Customizable Weighting**: Adjustable parameters to fine-tune the algorithm
- **Persistence**: Save and load system state from files
- **Simulation Mode**: Test and visualize different scheduling strategies

---

System Architecture

Core Components:

1. **Location Class**: Represents campus buildings with attributes
2. **Path Class**: Defines connections between locations
3. **CampusMap Class**: Manages the network of locations and paths
4. **ModifiedDijkstra Class**: Implements path-finding with custom weights
5. **MaintenanceScheduler Class**: Generates daily routes

Data Model

Location Attributes:

- ID and Name
- Importance (1-10 scale)
- Cleaning Frequency (days between cleanings)
- Visit Priority (base priority 1-10)
- Last Cleaned (days since last cleaned)
- Cleanliness Status (0-100%)

Path Attributes:

- Source and Destination
- Distance
- Travel Time
- Difficulty Factor (road condition)

Priority Calculation

Priority is calculated using a weighted formula:

```cpp
double calculateDynamicPriority(int locId) {
    // Time factor is normalized by cleaning frequency
    double timeFactorNormalized = min(1.0, static_cast<double>(loc.lastCleaned) / loc.cleaningF
    if (loc.lastCleaned < loc.cleaningFrequency) timeFactorNormalized *= 0.2;

    // Weighted priority calculation
    double priority = (loc.importance * 0.3) +
                      ((100 - loc.cleanlinessStatus) * 0.4) +
                      (loc.visitPriority * 0.1) +
                      (timeFactorNormalized * 0.2);
    return priority;
}
```

Modified Dijkstra Algorithm

Unlike standard Dijkstra, our algorithm:

- Uses weighted edge costs that consider:
  - Distance ($\alpha = 0.6$)
  - Path difficulty ($\beta = 0.3$)
  - Visit history ($\gamma = 0.1$)
  - Repeat visit penalty ($\delta = 0.2$)
- Incorporates dynamic building priorities
- Prioritizes high-importance, low-cleanliness locations

Route Planning Strategy

The MaintenanceScheduler class:

1. Updates cleanliness status daily
2. Calculates priority for all locations
3. Selects top ~1/3 of locations to visit each day
4. Uses a greedy approach to find optimal routes
5. Updates cleanliness status after visits

Example Campus Map

Our demo uses a 12-location campus with interconnected paths:

- Library (ID: 0)
- Main Building (ID: 1)
- Science Lab (ID: 2)
- Student Center (ID: 3)
- Cafeteria (ID: 4)
- Administration (ID: 5)
- Sports Complex (ID: 6)
- Research Center (ID: 7)
- Garden (ID: 8)
- Dormitory A (ID: 9)
- Dormitory B (ID: 10)
- Parking Lot (ID: 11)

Simulation Example: Day 1

Starting Status:

Location	Cleanliness	Last Cleaned	Priority
Library	100.0%	0 days	2.70
Main Building	100.0%	0 days	3.00
Science Lab	100.0%	0 days	2.40
...	...	...	...

Route: Library → Main Building → Cafeteria → Administration

Simulation Example: Day 3

Updated Status:

Location	Cleanliness	Last Cleaned	Priority
Library	87.5%	3 days	8.55
Science Lab	75.0%	3 days	14.70
Student Center	62.5%	3 days	18.13
...	...	...	...

Route: Library → Science Lab → Student Center → Garden

File Management

- **Data Persistence**: System state saved in `campus_data.txt`
- **Backup System**: Historical states in `campus_backup.txt`
- **Format**:

Copy

```
# Locations
ID,Name,Importance,CleaningFreq,Priority,Cleanliness,LastCleaned,Visits

# Paths
From,To,Distance,TravelTime,Difficulty
```

User Interface

Main Menu Options:

1. Find optimal path between two locations
2. View campus status
3. Run simulation for multiple days
4. Save current state
5. Reset to default configuration
6. Exit

Technical Considerations

- **Time Complexity**: O(E log V) for path finding
- **Space Complexity**: O(V + E) for storing the graph
- **Extensibility**: Easily add new locations or modify parameters
- **Portability**: Cross-platform C++ implementation

Future Enhancements

- Interactive visualization of routes
- Machine learning integration for adaptive priorities
- Staff scheduling and workload balancing
- Mobile application for real-time updates
- IoT integration for automatic cleanliness monitoring

Conclusion

The Campus Maintenance Optimization System demonstrates:

- Efficient resource allocation for facilities management
- Practical application of graph algorithms
- Dynamic priority-based scheduling
- Adaptable framework for different campus layouts

Q&A

Thank you for your attention!