MediFlow: Healthcare Staff Optimization

Web-Based Intelligent Scheduling Platform

Yash Sakhare

MSL304 Assignment

November 16, 2025

Outline

Healthcare Staffing Challenge

The Problem

Healthcare facilities face critical challenges in staff scheduling:

- **Patient demand** varies throughout the week
- **Staff availability** is constrained
- **Cost optimization** while maintaining service quality
- **Bottleneck identification** in patient flow

Solution Approach

Integrated web platform combining:

- Integer Linear Programming for staff optimization
- M/M/c Queue Theory for bottleneck analysis
- Modern web interface for real-time configuration

Integer Linear Programming Formulation

Decision Variables

$x_{is} \in \{0, 1\}$: Staff member i works shift s

Objective Function

$$\text{Minimize:} \quad \sum_{i \in \text{Staff}} \sum_{s \in \text{Shifts}} c_i \cdot x_{is}$$

where c_i is the cost per shift for staff member i

Key Constraints

- Shift requirements: $\sum_i x_{is} \geq R_s$ for each shift s
- Max hours: $\sum_s x_{is} \leq H_i$ for each staff i
- Availability: $x_{is} = 0$ if staff i unavailable for shift s

Queue Theory Integration

M/M/c Queue Model

- λ : Patient arrival rate (patients/hour)
- μ : Service rate per staff (patients/hour)
- c : Number of servers (staff members)
- $\rho = \frac{\lambda}{\mu \cdot c}$: Traffic intensity

Bottleneck Detection Criteria

- Critical: $\rho \geq 0.95$ (System overload)
- Warning: $0.85 \leq \rho < 0.95$ (High congestion)
- Caution: $0.75 \leq \rho < 0.85$ (Moderate load)
- Healthy: $\rho < 0.75$ (Normal operation)

Technology Stack

Backend

- **Flask**: REST API server
- **PuLP**: LP optimization
- **SimPy**: Queue simulation
- **Python 3.11**: Core logic

Frontend

- **Bootstrap 5**: UI framework
- **JavaScript**: Interactive logic
- **CSS3**: Modern animations
- **JSON**: Data exchange

Three-Tier Architecture

Presentation Layer $\rightarrow$ Business Logic $\rightarrow$ Data Layer

Feature 1: Patient Flow Simulator

Functionality

Real-time simulation of patient flow with bottleneck detection

Inputs

- Arrival rates by department
- Service times per stage
- Number of staff per station
- Simulation duration

Outputs

- Queue lengths
- Wait times
- Utilization (ρ)
- Bottlenecks

```
# Traffic intensity calculation
rho = arrival_rate / (service_rate * num_staff)
bottleneck = "Critical" if rho >= 0.95 else "Healthy"
```

Feature 2: Staff Rota Optimizer

Functionality

Automated staff scheduling using Integer Linear Programming

Interactive Configuration

- **Checkbox interface**: Select available shifts (Mon-Fri, AM/PM)
- **Cost input**: Set cost per shift for each staff member
- **Max hours**: Define weekly hour limits
- **Shift requirements**: Specify minimum staff per shift

Intelligent Features

- **Infeasibility detection**: Identifies impossible schedules
- **Cost minimization**: Finds optimal assignment
- **Constraint validation**: Real-time feasibility check

Feature 3: Checkbox-Based Input System

Problem Solved

Previous text-based input caused parsing errors and typos

Modern Solution

Each staff member gets a card with:

- 10 checkboxes (5 days $\times$ 2 shifts)
- Select All / Clear All buttons
- Cost and max hours inputs
- Visual feedback on selection

```
// Collect configuration from checkboxes
const availability = [];
document.querySelectorAll('.shift-checkbox:checked')
    .forEach(cb => availability.push(cb.value));
config.staff[name].availability = availability;
```

Feature 4: Infeasibility Analysis

Intelligent Error Handling

When optimization fails, system provides detailed analysis

4 Types of Issues Detected

1. **Insufficient Staff**: Not enough people available
2. **No Flexibility**: Staff can't cover required shifts
3. **Overworked Staff**: Hours exceed maximum limits
4. **Capacity Shortfall**: Total capacity below demand

Actionable Suggestions

- "Increase Tech_D's max_hours from 40 to 56"
- "Add availability for Nurse_A on Mon_AM, Tue_AM"
- "Hire 1 additional staff member"

Feature 5: Real-Time Configuration Testing

Test Without Saving

Users can validate configurations before committing changes

Workflow

1. Modify staff availability
2. Click "Test Configuration"
3. View feasibility results
4. Adjust if infeasible
5. Save when satisfied

Benefits

- No data corruption
- Instant feedback
- Iterative refinement
- Risk-free exploration

Feature 6: Modern UI/UX Design

Visual Design Elements

- **Gradient backgrounds**: Purple-blue color scheme
- **Glass morphism**: Translucent navbar effects
- **Smooth animations**: 60 FPS transitions
- **Toast notifications**: Non-intrusive feedback
- **Color-coded results**: Red (critical), Yellow (caution), Green (healthy)

Responsive Design

- Tab-based navigation (Simulator — Optimizer — Configuration)
- Card-based layouts for staff management
- Mobile-friendly Bootstrap grid system
- Accessible form controls with labels

Dynamic Configuration Reload

Problem

Configuration changes weren't reflected in optimization results

Solution

`get_current_config()` function reloads from file each time

```
def get_current_config():
    """Reload config from file to get latest changes"""
    with open('config.json', 'r') as f:
        return json.load(f)

def run_optimisation():
    config = get_current_config()  # Fresh data every time
    # ... optimization logic ...
```

Impact

Users can modify configurations and see immediate results

Infeasibility Analysis Engine

```python
def analyze_infeasibility(config):
    issues = []
    suggestions = []

    # Check 1: Insufficient staff
    available_staff = len(config['staff'])
    max_requirement = max(config['shift_requirements'].values())
    if available_staff < max_requirement:
        issues.append("Insufficient staff available")
        suggestions.append(f"Hire {max_requirement - available_staff} more staff")

    # Check 2: No flexibility (staff can't cover shifts)
    for shift in config['shift_requirements']:
        available_for_shift = [s for s, data in config['staff'].items()
                               if shift in data['availability']]
        if len(available_for_shift) < config['shift_requirements'][shift]:
            issues.append(f"Not enough staff available for {shift}")

    # ... more checks ...
    return {"issues": issues, "suggestions": suggestions}
```

REST API Endpoints

Core API Routes

POST /api/simulate Run patient flow simulation

POST /api/optimize Execute staff optimization

POST /api/config/test Test configuration feasibility

GET /api/config Retrieve current configuration

PUT /api/config Update and save configuration

```
@app.route('/api/optimize', methods=['POST'])
def optimize():
    result = run_optimisation()
    if result['status'] == 'infeasible':
        analysis = result['analysis']
        return jsonify({"status": "infeasible",
                        "issues": analysis['issues'],
                        "suggestions": analysis['suggestions']})
    return jsonify({"status": "success",
                    "assignments": result['assignments']})
```

Technical Innovations

1. **Scientific Bottleneck Detection**
   - Replaced arbitrary thresholds with M/M/c traffic intensity
   - Little's Law validation: $L = \lambda \cdot W$
   - Four-level classification system
2. **Error-Proof Input Interface**
   - Checkbox-based selection eliminates parsing errors
   - Visual feedback with select all/clear all
   - Zero typing required for shift selection
3. **Intelligent Infeasibility Handling**
   - Automatic constraint conflict detection
   - Specific, actionable suggestions with exact values
   - Real-time feasibility testing

User Experience Enhancements

Workflow Improvements

- **Automatic tab switching**: Results appear in correct tab
- **Null safety**: All DOM operations validated
- **Toast notifications**: Non-blocking status updates
- **Loading indicators**: Clear feedback during processing
- **Color coding**: Instant visual status recognition

Developer Experience

- **Clean separation**: Frontend/Backend clearly divided
- **RESTful design**: Standard HTTP methods and JSON
- **Modular architecture**: Easy to extend and maintain
- **Comprehensive testing**: 10 test cases documented

Example: Feasible Schedule

Input Configuration

- 4 staff members (2 Nurses, 2 Technicians)
- Each available Mon-Fri, both AM/PM shifts
- Max hours: 40 per week
- Shift requirements: 2 staff per shift

Optimization Result

- **Status: FEASIBLE**
- Total cost: $2,400
- All shifts covered
- No staff exceeds 40 hours
- Balanced workload distribution

Example: Infeasible Schedule

Input Configuration

- Nurse_A availability: Only Wed-Fri (reduced from Mon-Fri)
- Other staff: Unchanged
- Same shift requirements: 2 staff per shift

System Response

- **Status: INFEASIBLE**
- **Issue**: Tech_D needs 56 hours but max is 40
- **Suggestion 1**: Increase Tech_D's max_hours to 56
- **Suggestion 2**: Add Nurse_A availability for Mon/Tue
- **Suggestion 3**: Hire additional staff member

Validation Through Testing

Test Suite Coverage

10 comprehensive test cases covering:

- **Feasible scenarios**: Standard, minimal, full, distributed (6 cases)
- **Infeasible scenarios**: Understaffed, overworked, no flexibility (3 cases)
- **Edge cases**: Last-minute changes (1 case)

Validation Results

- All feasible cases produce valid schedules
- All infeasible cases correctly detected with explanations
- Configuration changes properly reflected in results
- No JavaScript errors in browser console
- Checkbox interface prevents all input errors

System Impact

Operational Benefits

- **Cost reduction**: Automated optimization saves planning time
- **Quality assurance**: Bottleneck detection prevents service degradation
- **Flexibility**: Easy configuration changes for dynamic needs
- **Transparency**: Clear explanations for infeasible schedules

Technical Achievements

- Integration of optimization and simulation
- User-friendly web interface with modern design
- Robust error handling and validation
- Production-ready with comprehensive testing

Future Enhancements

Short-term

- Multi-week scheduling
- Staff preference weights
- Export to PDF/Excel
- Email notifications

Long-term

- Machine learning predictions
- Mobile application
- Database backend
- Multi-facility support

Scalability

Architecture supports extension to:

- Larger healthcare networks
- Multiple departments simultaneously
- Historical data analysis and trend prediction

Thank You

MediFlow: Healthcare Staff Optimization

Yash Sakhare
MSL304 Assignment

Project Repository

GitHub: MSL304-Assignment

Questions?