Assignment 2: Optimization in Practice

Management Science

Assignment Overview

- Due: Start of Lecture 10
- Weight: 30% of final grade
- Expected Time: 5-7 hours
- Work: Groups

Your consulting firm has landed a major contract with CityExpress, a rapidly growing local delivery company. The CEO is concerned about rising operational costs and worker turnover. Your team has been tasked with:

1. Route Optimization: Reduce fuel costs by finding better delivery routes
2. Workforce Satisfaction: Improve worker retention by creating shift schedules that respect preferences

Consultants

Who is part of your group?

```
"""
YOUR ANSWER HERE:
"""
```

Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import hashlib

# Sets a random seed for reproducibility (CHANGE THIS HERE!!!)
group_name = "[THE NAMES OF YOUR GROUP MEMBERS HERE]"
# Convert string to integer for seeding (deterministic across sessions)
seed = int(hashlib.md5(group_name.encode()).hexdigest(), 16) % (2**31)
np.random.seed(seed)
```

> ⚠ Warning
>
> Fill in the names of all the members of your group in `group_name` instead of `[THE NAMES OF YOUR GROUP MEMBERS HERE]`. If you don't do so, you will maximally receive half of the points.

Part A: Route Optimization (50%)

The Challenge

CityExpress currently plans routes manually. A driver mentioned: "I feel like I'm driving in circles sometimes." The operations manager suspects they're wasting 15-20% on inefficient routing.

Tomorrow's schedule has 20 customer deliveries. Your job: build a route that minimizes total distance, starting and ending at the depot.

Impact: If successful, this approach will be rolled out to all 50 daily routes.

The Data

> ℹ Note
>
> Unique Data Per Group: The customer locations below are randomly generated based on your group's seed. Each group will get different customer coordinates within a 20km × 20km service area.

```
# DON'T CHANGE ANYTHING BELOW!
# Generate customer locations based on your group's seed
# Each group gets unique locations within a 20km × 20km area

# Depot is always at the center
locations = {0: (10, 10)}

# Generate 20 customer locations randomly
for i in range(1, 21):
    x = np.random.uniform(2, 18)  # Between 2-18 km
    y = np.random.uniform(2, 18)  # Between 2-18 km
    locations[i] = (x, y)

print("=" * 60)
print("YOUR GROUP'S UNIQUE CUSTOMER LOCATIONS")
print("=" * 60)
print(f"Depot: {locations[0]}")
print(f"\nCustomer locations (x, y in km):")
for i in range(1, 21):
    print(f"  Customer {i:2d}: ({locations[i][0]:5.2f}, {locations[i]
[1]:5.2f})")
```

```
print("=" * 60)
# DON'T CHANGE ANYTHING ABOVE!
```

```
============================================================
YOUR GROUP'S UNIQUE CUSTOMER LOCATIONS
============================================================
Depot: (10, 10)

Customer locations (x, y in km):
  Customer  1: ( 7.10,  3.29)
  Customer  2: ( 9.45, 17.06)
  Customer  3: (12.59, 16.53)
  Customer  4: (15.31, 15.74)
  Customer  5: (17.82,  9.38)
  Customer  6: (11.85, 17.08)
  Customer  7: (10.93, 14.50)
  Customer  8: (15.53,  4.74)
  Customer  9: ( 5.03,  9.69)
  Customer 10: ( 7.84,  2.82)
  Customer 11: ( 4.03, 11.35)
  Customer 12: ( 5.08, 13.78)
  Customer 13: ( 9.10,  9.82)
  Customer 14: ( 3.72,  2.27)
  Customer 15: (14.84,  2.26)
  Customer 16: ( 9.61, 11.55)
  Customer 17: (12.06,  6.64)
  Customer 18: (12.21,  5.29)
  Customer 19: ( 2.94, 16.30)
  Customer 20: ( 7.62,  9.48)
============================================================
```

Task 1: Build an Initial Route (20%)

Use the nearest neighbor heuristic to create a baseline route.

The approach:

- Start at the depot
- At each step, visit the nearest unvisited customer
- Return to depot when all customers are visited

Your deliverable:

- Implement functions to calculate distances and build the route
- Print the route and its total distance

> **Tip**
>
> The Euclidean distance formula is: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

```
# Build your nearest neighbor route here
# You'll need functions to:
# 1. Calculate distance between two locations
# 2. Calculate total route distance
# 3. Build the route using nearest neighbor logic
# YOUR CODE HERE
```

Task 2: Improve the Route (20%)

The nearest neighbor heuristic is fast but rarely optimal. Use 2-opt local search to improve your route.

How 2-opt works: The algorithm removes two edges from your route and reconnects them in a different way, potentially "uncrossing" the route.

```
Before:  Depot → A → B → C → D → Depot
After reversing segment between positions 1 and 3:
After:   Depot → C → B → A → D → Depot
```

The approach:

1. Try all possible ways to reverse segments of your route
2. If a reversal improves the distance, keep it
3. Repeat until no improvement is possible

Your deliverable:

- Implement 2-opt improvement
- Show the improved route and distance
- Report how many iterations it took to converge

> ! Important
>
> Keep the depot fixed at the start and end! Only reverse segments between customers.

```
# Implement 2-opt improvement here
# YOUR CODE HERE
```

Task 3: Visualize and Quantify Impact (10%)

The CEO wants to see the difference visually and understand the cost savings.

Your deliverable:

1. Visualization: Plot both routes (before and after optimization) showing:
   - Customer locations
   - Route paths
   - Clear labels for depot and customers

2. Business metrics:
   - Original distance vs. improved distance
   - Percentage improvement
   - Cost savings per delivery (assume €2/km fuel cost)
   - Projected monthly savings if applied to all 50 daily routes

```
# Create visualizations and calculate business impact
# YOUR CODE HERE
```

Business Question: Investment Recommendation

Based on your analysis, prepare a formal recommendation to the CEO about investing in route optimization software.

```
"""
RECOMMENDATION TO CEO

To: CityExpress CEO
From: [Your Consulting Team]
Re: Route Optimization Software Investment Recommendation

Investment Decision: [Should we invest in route optimization software? Yes/
No]

Cost-Benefit Analysis: [2-3 sentences quantifying the monthly and annual
savings
potential based on your results. Include specific numbers from your
analysis.]

Implementation Recommendations: [2-3 sentences on how to roll out this
solution. Consider pilot testing, training needs, and timeline.]

Limitations & Risk Mitigation: [2-3 sentences acknowledging limitations of
the
2-opt approach and suggesting how to address them]

Expected ROI: [1-2 sentences on payback period and long-term value,
considering
software costs versus fuel savings.]
"""
```

Part B: Worker Shift Scheduling (50%)

The Challenge

CityExpress warehouse operates with 6 workers across 6 shifts per week. Recently, worker turnover has been high. Exit interviews reveal: “They never give me the shifts I want.”

The warehouse manager manually assigns shifts each week, often just assigning workers in order without considering preferences. Your job: create a better assignment system that maximizes worker satisfaction.

Impact: Higher satisfaction could reduce the 30% annual turnover rate, saving thousands in recruiting and training costs.

The Data

> **Note**
>
> Unique Data Per Group: Worker shift preferences below are randomly generated based on your group's seed. Each group will get different preference patterns while maintaining similar complexity (varying numbers of preferences per worker, overlapping demands).

```
# DON'T CHANGE ANYTHING BELOW!
# Available shifts for the week (14 shifts across 7 days, 2 per day)
shifts = [
    'Monday-Morning', 'Monday-Evening',
    'Tuesday-Morning', 'Tuesday-Evening',
    'Wednesday-Morning', 'Wednesday-Evening',
    'Thursday-Morning', 'Thursday-Evening',
    'Friday-Morning', 'Friday-Evening',
    'Saturday-Morning', 'Saturday-Evening',
    'Sunday-Morning', 'Sunday-Evening'
]

# Generate worker preferences randomly for your group
worker_preferences = {}
for worker_id in range(14):
    # Each worker gets 2-4 preferred shifts
    num_prefs = np.random.randint(2, 5)
    # Randomly select preferred shifts (without replacement)
    preferred_indices = np.random.choice(14, size=num_prefs, replace=False)
    worker_preferences[worker_id] = [shifts[i] for i in
sorted(preferred_indices)]

print("=" * 70)
print("YOUR GROUP'S UNIQUE WORKER SHIFT PREFERENCES")
print("=" * 70)
for worker_id, prefs in worker_preferences.items():
    print(f"Worker {worker_id:2d}: {prefs}")
print(f"\nTotal preferences across all workers: {sum(len(p) for p in
worker_preferences.values())}")
print(f"Average preferences per worker: {sum(len(p) for p in
worker_preferences.values()) / 14:.1f}")
print("=" * 70)

# Each worker works exactly ONE shift
```

```
# Each shift must have exactly ONE worker
# DON'T CHANGE ANYTHING ABOVE!
```

```
======================================================================
YOUR GROUP'S UNIQUE WORKER SHIFT PREFERENCES
======================================================================
Worker  0: ['Tuesday-Evening', 'Thursday-Evening', 'Friday-Evening',
'Sunday-Morning']
Worker  1: ['Tuesday-Morning', 'Wednesday-Morning', 'Wednesday-Evening',
'Friday-Evening']
Worker  2: ['Monday-Morning', 'Friday-Evening']
Worker  3: ['Friday-Morning', 'Friday-Evening']
Worker  4: ['Tuesday-Morning', 'Friday-Evening', 'Sunday-Evening']
Worker  5: ['Wednesday-Evening', 'Thursday-Morning']
Worker  6: ['Tuesday-Evening', 'Thursday-Morning', 'Saturday-Evening',
'Sunday-Evening']
Worker  7: ['Tuesday-Evening', 'Wednesday-Evening', 'Saturday-Morning']
Worker  8: ['Monday-Evening', 'Wednesday-Evening', 'Friday-Morning',
'Sunday-Morning']
Worker  9: ['Thursday-Morning', 'Saturday-Morning', 'Saturday-Evening']
Worker 10: ['Tuesday-Evening', 'Saturday-Evening']
Worker 11: ['Monday-Morning', 'Saturday-Evening']
Worker 12: ['Monday-Evening', 'Thursday-Morning', 'Thursday-Evening',
'Friday-Morning']
Worker 13: ['Tuesday-Morning', 'Thursday-Morning', 'Thursday-Evening']

Total preferences across all workers: 42
Average preferences per worker: 3.0
======================================================================
```

> **Note**
>
> Assignment representation: Use a list where `assignment[worker_id] = shift_index`
>
> Example: `[0, 3, 1, 13, 2, 5, ...]` means:
>
> - Worker 0 → shift 0 (Monday-Morning)
> - Worker 1 → shift 3 (Tuesday-Evening)
> - Worker 2 → shift 1 (Monday-Evening)
> - Worker 3 → shift 13 (Sunday-Evening)
> - etc.

Task 1: Design a Greedy Assignment Strategy (15%)

Remember greedy heuristics from Lecture 6 (SPT, EDD)? Design your own greedy rule for shift assignment.

Think about:

- SPT prioritized shortest jobs
- EDD prioritized earliest deadlines

- What should YOU prioritize? Worker flexibility? Number of preferences? Something else?

Your deliverable:

1. Implement a function that creates a valid assignment using YOUR greedy strategy
2. Print the assignment (who gets which shift)
3. Write 3-4 sentences explaining:
   - What attribute you chose to prioritize
   - Why this makes sense
   - How it relates to scheduling heuristics from Lecture 6

```
# Implement your greedy assignment strategy
# Must produce a valid assignment (all workers assigned, no duplicate
shifts)
# YOUR CODE HERE
```

```
"""
YOUR EXPLANATION HERE:
My greedy strategy: [what did you prioritize?]
Reasoning: [why does this make sense?]
Connection to Lecture 6: [how does this relate to SPT/EDD/other
heuristics?]
"""
```

Task 2: Measure Assignment Quality (15%)

You need a way to measure how "good" an assignment is. Design a satisfaction scoring system.

Design questions:

- Should 1st choice be worth more than 2nd choice? How much more?
- What if a worker gets a shift they didn't list?
- Should some workers count more than others?

Your deliverable:

1. Implement a satisfaction calculation function
2. Calculate the satisfaction score for your greedy assignment
3. Write 2-3 sentences explaining:
   - Your scoring system design
   - Why you chose this approach

```
# Implement satisfaction calculation
# YOUR CODE HERE
```

```
"""
YOUR EXPLANATION HERE:
```

```
My scoring system: [how do you calculate satisfaction?]
Reasoning: [why this approach?]
"""
```

Task 3: Improve with Local Search (15%)

Your greedy solution is probably not optimal. Use local search (like 2-opt from Part A) to improve it.

The approach:

- Try swapping shifts between pairs of workers
- If a swap improves satisfaction, keep it
- Repeat until no improvement is found

Your deliverable:

1. Implement local search improvement
2. Show before/after satisfaction scores
3. Create a visualization comparing the solutions (table or chart)
4. Write 3-4 sentences explaining:
   - How much you improved the greedy solution
   - Whether you think you found the optimal solution
   - What limitations local search might have

```
# Implement local search improvement
# YOUR CODE HERE
```

```
# Visualize the improvement
# YOUR CODE HERE
```

```
"""
YOUR EXPLANATION HERE:
Results: [how much improvement?]
Optimality: [did you reach the best possible solution? how do you know?]
Limitations: [what are the downsides of local search?]
"""
```

Task 4: Business Recommendations & Contingency Planning (5%)

Business Question 1: Emergency Coverage Protocol

A critical situation has emerged: Worker 2 calls in sick 10 minutes before their shift. Prepare an emergency response to the warehouse manager.

```
"""
EMERGENCY COVERAGE

To: Warehouse Manager
```

```
From: [Your Consulting Team]
Re: Emergency Shift Coverage Protocol - Worker Absence

Immediate Action Plan: [2-3 sentences]

Decision Framework: [2-3 sentences]

Future System Adaptation: [2-3 sentences]

"""
```

Business Question 2: Strategic Workforce Management Analysis

The warehouse manager is skeptical: “This is too complicated. I just need bodies in shifts. I don’t care about preferences.” Prepare a strategic analysis comparing approaches.

```
"""
STRATEGIC ANALYSIS

To: Warehouse Manager
From: [Your Consulting Team]
Re: Cost-Benefit Analysis of Worker Preference System

Current Approach (Random Assignment):
- Advantages: [2-3 sentences]
- Hidden Costs: [2-3 sentences]

Proposed Approach (Preference-Based):
- Implementation Costs: [2-3 sentences]
- Expected Benefits: [2-3 sentences]

ROI Calculation: [2-3 sentences]

Recommendation: [1-2 sentences with your final recommendation based on your
quantitative analysis above.]
"""
```

Submission Checklist

- ☐ All code cells run without errors
- ☐ Part A: Routes visualized and business impact calculated
- ☐ Part B: All three methods implemented (greedy, evaluation, local search)
- ☐ All written explanations completed with business context
- ☐ Code is clean and understandable
- ☐ Group member names listed at top

Tips

- Use AI tools to help you understand concepts and write code, but make sure YOU understand the logic
- Start simple - get a basic version working before adding complexity
- Test incrementally - don't write all functions at once
- Experiment - there's no single "correct" greedy strategy or scoring system
- Think like a consultant - your audience is business stakeholders, not just programmers

Bibliography