Better Routing

Lecture 7 - Management Science

Dr. Tobias Vlćek

Introduction

Client Briefing: Artisan Bakery

. . .

Master Baker's Morning Dilemma:

"Every morning at 5 AM, our delivery van leaves with fresh bread for 12 cafés across the city. Our driver takes 3 hours for what should be a 2-hour route. We're burning fuel and arriving late. Can you optimize our morning delivery?"

The Delivery Challenge

Artisan Bakery's daily logistics puzzle:

- 12 Cafés: Each expecting fresh bread by 8 AM
- One Van: Limited capacity, must visit all locations
- Time Windows: 3 cafés open early (6:30 AM) and need priority
- Current Problem: Driver uses "gut feeling" for routing
- Cost Impact: Extra hour = €50 fuel + €30 labor + unhappy customers

. . .

> ! Important
>
> The Stakes: Poor routing costs €80+ daily, or €29,200 annually. Plus reputation damage from late deliveries!

Quick Recap: Greedy Decisions

Last week we learned greedy algorithms for scheduling:

- SPT: Process shortest jobs first
- EDD: Process by earliest due date
- Fast & Simple: Made quick decisions, no looking back

. . .

Question: Can we use the same greedy approach for routing?

. . .

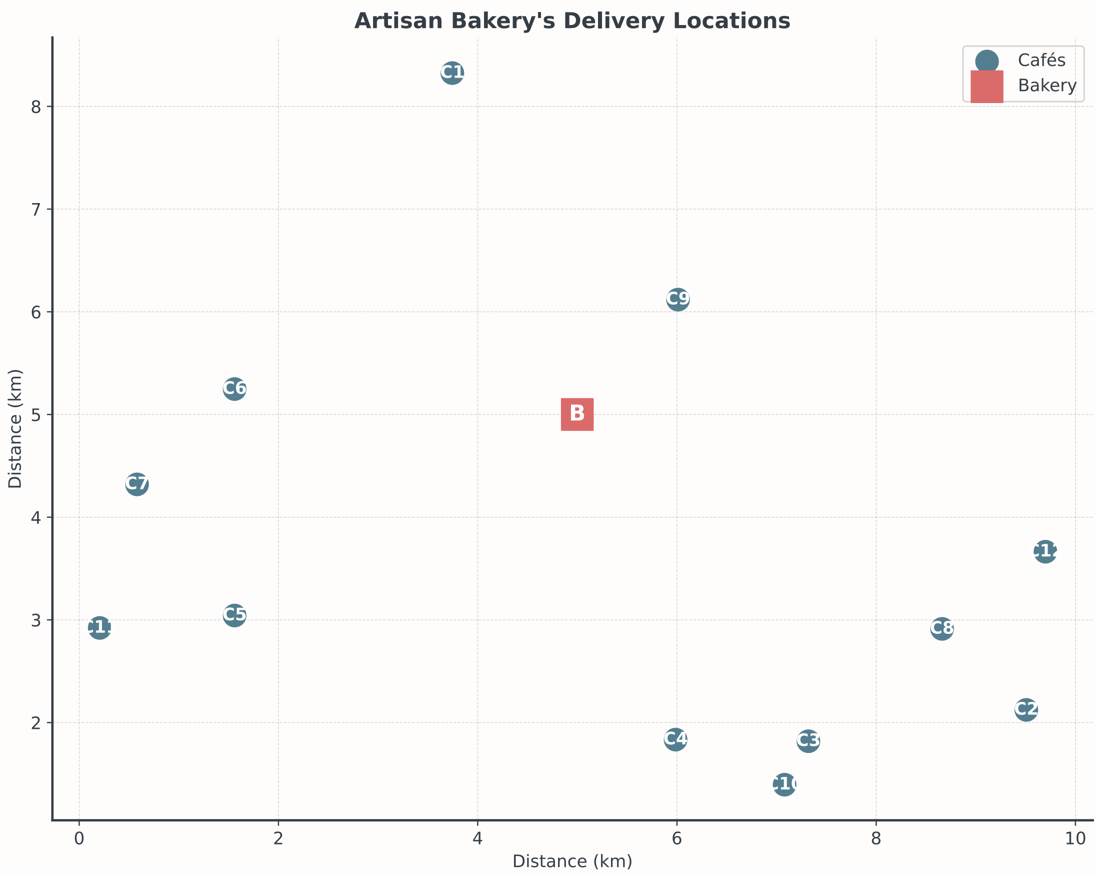
> ℹ Note
>
> Today's Journey: We'll start greedy (nearest neighbor), then learn how to improve solutions with local search!

The Routing Problem

The Traveling Salesman Problem

The classic optimization challenge: Visit all locations exactly once, minimize total distance.

. . .



Why Can't Computers Just Try Everything?

Let's calculate: 12 cafés = 12! = 479,001,600 routes

. . .

If your computer checks 1 million routes per second:

- 5 cafés: 0.0001 seconds ✓

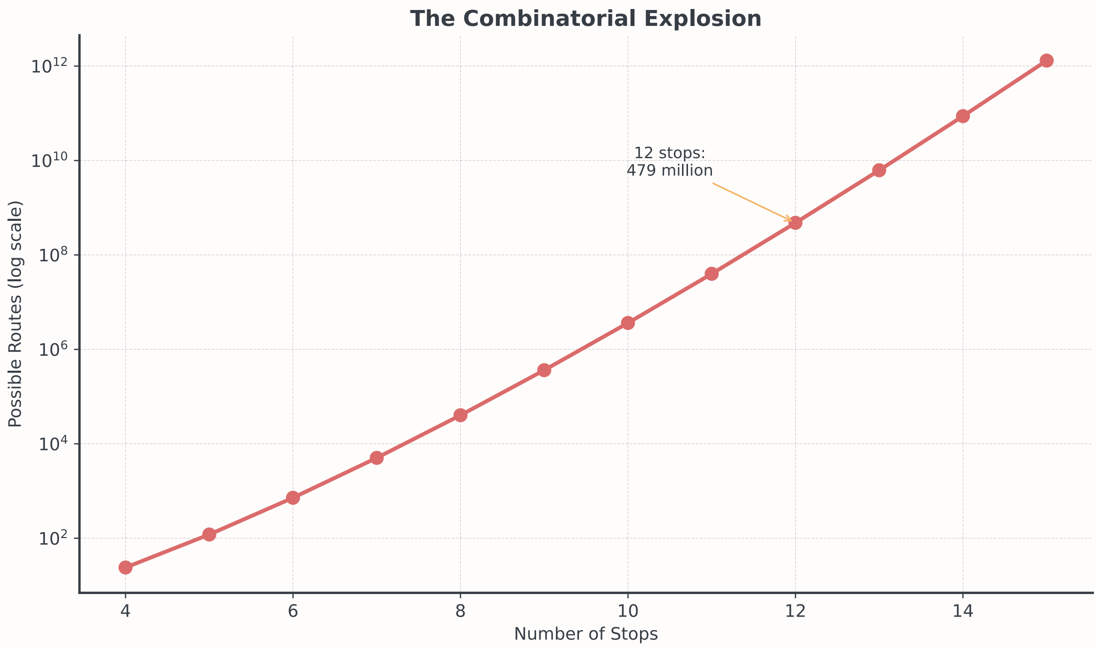
- 10 cafés: 0.18 seconds ✓
- 12 cafés: 8 minutes ⚠️
- 15 cafés: 12 hours ⚠️
- 20 cafés: 77,000 years ☠️

. . .

> ! Important
>
> The Reality: Amazon has 100,000+ delivery stops daily. Exact optimization would take longer than the universe has existed!

The Complexity Explosion

The factorial growth makes exhaustive search impossible.



Local Search Framework

The Four Pillars of Local Search

Any problem can be solved with local search by defining:

1. Search Space: All possible solutions (3.6M routes for 10 cafés!)
2. Initial Solution: Starting point (our greedy route)
3. Objective Function: How we measure quality (total distance)
4. Neighborhood Structure: How to create “nearby” solutions (2-opt swaps)

. . .

! Important

The power of local search: The same “engine” works for routing, scheduling, or any combinatorial problem - just plug in different components!

Greedy Construction

Nearest Neighbor Algorithm

Build a route by always visiting the closest unvisited location.

. . .

The Algorithm: 1. Start at the bakery 2. Find the nearest unvisited café 3. Go there 4. Repeat until all visited 5. Return to bakery

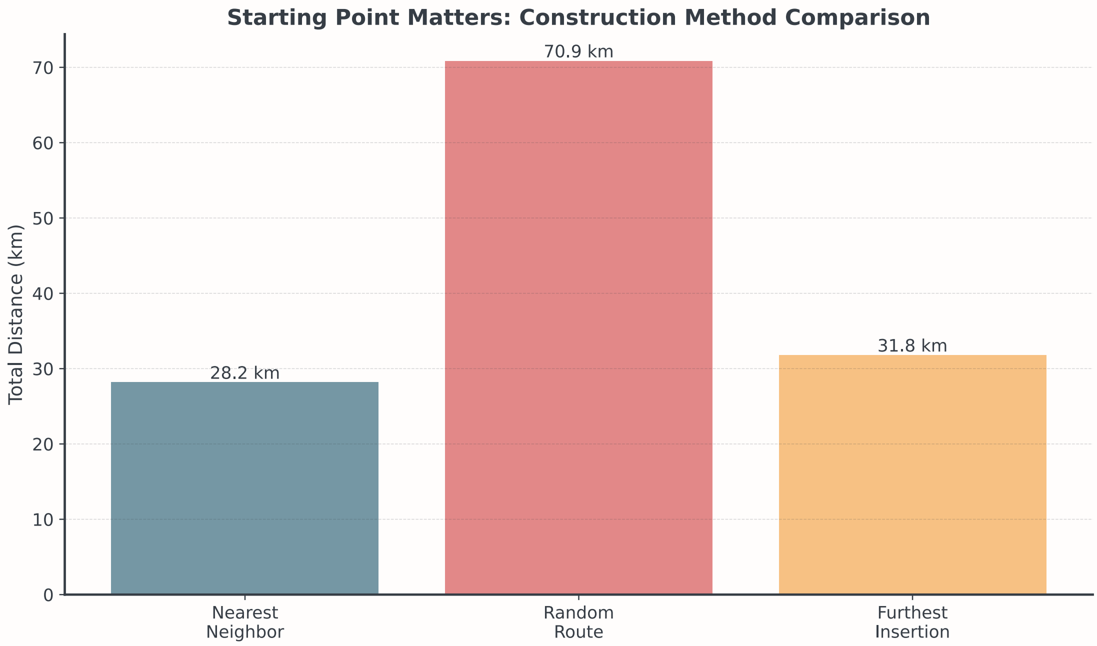
. . .

Tip

Intuition: Like picking low-hanging fruit - grab what’s easiest (nearest) first!

Construction Methods Comparison

Different ways to build initial routes:

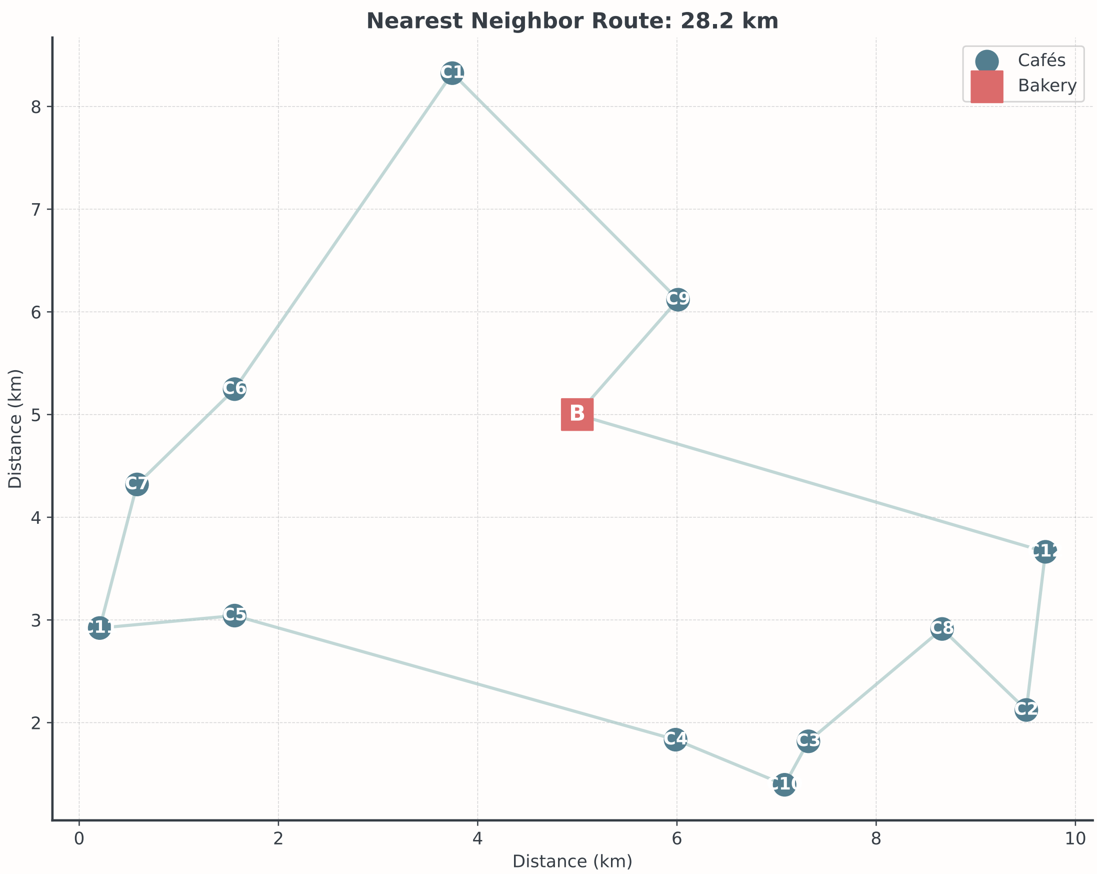


. . .

i Note

Different construction methods give different starting points. The better your start, the better your final result after improvement!

Nearest Neighbor in Action



The Problem with Greedy

Question: Can you spot any obvious inefficiencies in this route?

. . .

Common Issues: ::: incremental - Crossing paths: Route crosses over itself - Long return: Far from bakery at the end - Myopic decisions: Can't see the "big picture" :::

. . .

Warning

Nearest neighbor typically gives solutions 15-25% worse than optimal. Can we improve it?

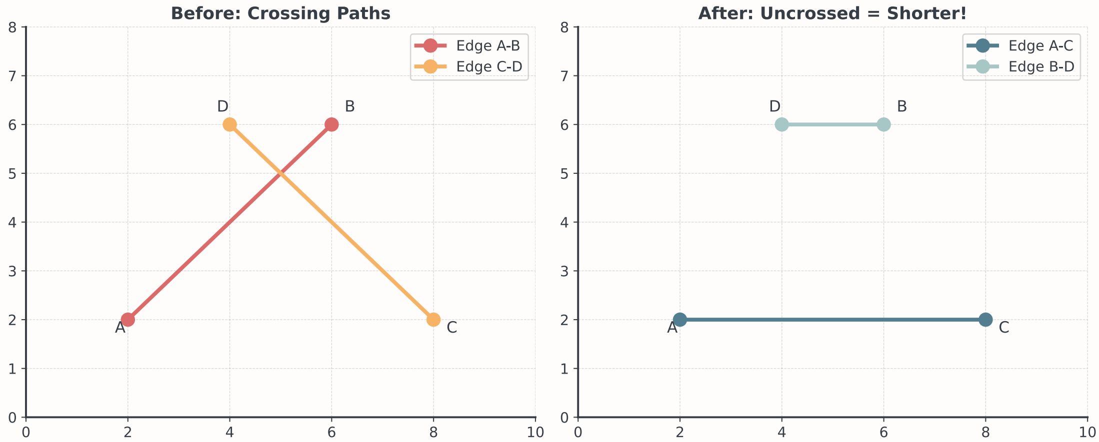
Local Search Improvements

The 2-Opt Algorithm

Systematically improve routes by removing crossing paths.

. . .

The Idea: Take two edges and swap them to uncross the route



How 2-Opt Works

The systematic improvement process:

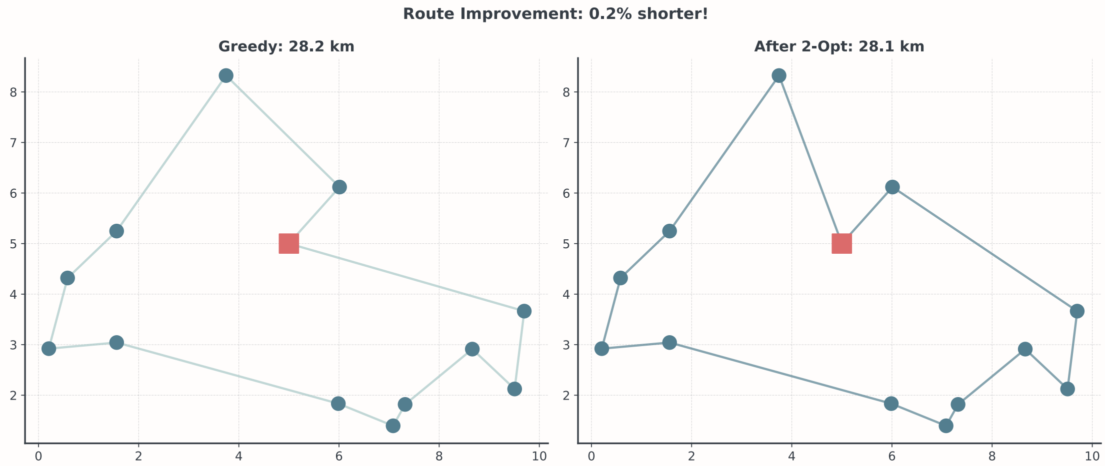
. . .

```
# Simplified 2-opt logic (conceptual)
for i in range(route_length):
    for j in range(i+2, route_length):
        # Try swapping edges (i,i+1) and (j,j+1)
        new_distance = calculate_with_swap(i, j)
        if new_distance < current_distance:
            perform_swap(i, j)
            improvement_found = True
```

. . .

> **i Note**
>
> 2-opt examines all possible pairs of edges and swaps those that reduce total distance. Keep iterating until no improvement found!

2-Opt Applied to Our Route



Beyond 2-Opt: More Powerful Moves

The k-opt family of improvements:

2-opt - Removes 2 edges - 1 way to reconnect - $O(n^2)$ combinations - Fast, good results

3-opt - Removes 3 edges - 7 ways to reconnect - $O(n^3)$ combinations - Better but slower

Or-opt - Moves 1-3 nodes - Different philosophy - Good for time windows - Preserves orientation

. . .

> Tip
>
> Industry practice: Start with 2-opt (fast), use 3-opt if you have time!

Local Search Philosophy

Start with any solution, then iteratively improve it.

. . .

The Process: ::: incremental 1. Initial Solution: Use greedy (fast but suboptimal) 2. Define Neighborhood: What changes can we make? 3. Search: Try neighboring solutions 4. Accept: Move if better 5. Repeat: Until no improvements found :::

. . .

> Tip
>
> Local search transforms “quick and dirty” solutions into “pretty good” ones!

The Local Optimum Trap: A Hiker's Dilemma

Imagine you're a hiker dropped in foggy mountains at night…

. . .

Your Mission: Find the highest peak (global optimum) Your Tool: An altimeter (objective function) Your Vision: Only the ground at your feet (local neighborhood)

. . .

The Greedy Strategy: Always step uphill

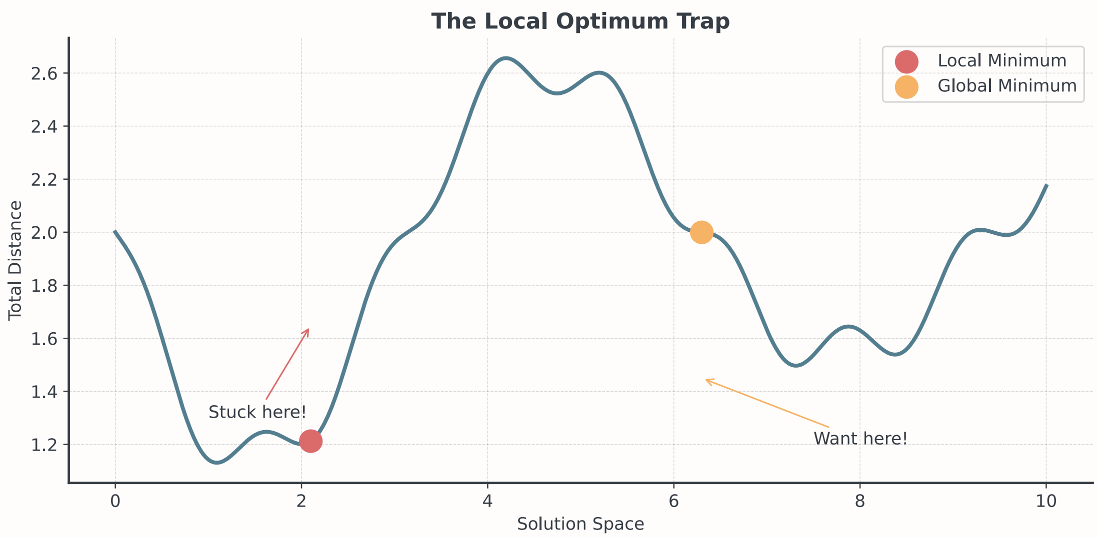
. . .

Question: What happens when you reach the top of a small hill?

. . .

> ⚠ Warning
>
> You're stuck! Every step is downhill, but you might be on a tiny hill while Mount Everest is nearby. This is the local optimum trap!

Visualizing Local Optima



Escaping Local Optima: Multi-Start Strategy

Simple fix: Don't put all eggs in one basket!

. . .

```python
def multi_start_optimization(n_starts=10):
    """Try multiple random starts, keep the best"""
    best_route = None
    for i in range(n_starts):
```

```
        # New random start
        route = random_initial_route()
        # Improve with 2-opt
        route = improve_with_2opt(route)
        # Keep if best so far
        if better_than(route, best_route):
            best_route = route
    return best_route
```

. . .

ℹ Note

Like hiring 10 drivers, letting each plan their own route, then picking the best! Simple but effective.

Real-World Impact: How Good is Good Enough?

Industry benchmarks for delivery optimization:

Method	vs Optimal	Industry Use
Human intuition	+40-60%	Still common!
Nearest Neighbor	+20-25%	Quick dispatch
NN + 2-opt	+5-15%	Standard practice
Advanced Meta	+2-5%	Premium logistics
Exact (if possible)	0%	Research only

. . .

! Important

Business Reality: A 10% improvement = millions in savings for large logistics companies. Your 2-opt implementation could literally pay for itself in one day!

Time Windows Add Complexity

Some cafés open early and need priority delivery:

. . .

Artisan Bakery's Constraints: - Café Europa: Opens 6:30 AM (early birds) - Sunrise Bistro: Opens 6:30 AM (breakfast rush) - Morning Glory: Opens 6:45 AM (commuter stop) - Others: Open 7:00 AM or later

. . .

! Important

With time windows, we must balance distance minimization with deadline satisfaction. Pure distance optimization might arrive too late!

Mission Briefing

Choosing Your Weapon: Algorithm Selection

Different situations call for different approaches:

Situation	Best Approach	Why
Need solution NOW (< 1 sec)	Nearest Neighbor	Lightning fast
Have 1 minute	NN + 2-opt	Good balance
Have 5 minutes	Multi-start + 2-opt	Explore more options
Time windows critical	NN (prioritize early) + Or-opt	Preserves time feasibility
Academic benchmark	3-opt or SA	Maximum quality

...

Tip

Today's Competition: You have 60 minutes - use multi-start with 2-opt!

Implementation Pitfalls to Avoid

Common bugs that cost you 30 minutes:

❌ Forgetting return to bakery: Your distance calculation must include the trip back!

```
# WRONG
total = sum(distances between consecutive stops)
# RIGHT
total = sum(distances) + distance(last_stop, bakery)
```

❌ Index confusion in 2-opt: Remember Python slicing!

```
# The 2-opt swap reverses route[i+1:j+1], not route[i:j]
```

❌ Modifying while iterating: Make a copy!

```
new_route = current_route.copy()  # Don't modify original
```

Your Toolkit for Today

Construction + Improvement = Better Routes

. . .

Construction (Greedy) - Nearest Neighbor - Cheapest Insertion - Savings Algorithm

Improvement (Local Search) - 2-opt swaps - 3-opt (advanced) - Or-opt (move sequences)

. . .

ℹ Note

Most commercial routing software uses this two-phase approach: Build quickly, then polish!

Next: Bean Counter Practice

In the notebook, you'll help Bean Counter optimize coffee bean deliveries to 10 franchises.

. . .

You'll implement: ::: incremental - Distance calculations between locations - Nearest neighbor construction - Route distance measurement - 2-opt improvement - Multi-start optimization :::

. . .

Then: Apply these skills to Artisan Bakery's 12-café challenge!

The Competition Challenge

Artisan Bakery needs your help with their morning route!

. . .

Your Mission: - Optimize delivery to 12 cafés - Handle 3 early time windows - Minimize total distance - Beat the current 3-hour route

. . .

> Tip
>
> Hint: Start with nearest neighbor, but don't forget about those time windows! Sometimes a slightly longer route that meets deadlines is better than the shortest route that arrives late.

Next Week: When Local Search Isn't Enough

Today we learned to climb hills. Next week: How to jump mountains!

. . .

Coming Attractions: - Simulated Annealing: Accept worse solutions probabilistically (like heating metal) - Tabu Search: Remember where you've been (search with memory) - Genetic Algorithms: Evolve solutions (survival of the fittest routes)

. . .

> Note
>
> All use the same local search foundation - just with clever tricks to escape local optima!

Literature

Resources

Essential Reading: - Applegate et al. (2011): The Traveling Salesman Problem - The definitive TSP reference - Laporte (1992): The Vehicle Routing Problem - Overview of routing algorithms - Lin & Kernighan (1973): Classic paper on k-opt improvements

Python Libraries: - `scipy.spatial.distance` - Fast distance calculations - `networkx` - Graph algorithms including TSP approximations - `ortools` - Google's optimization tools with routing

> Summary
>
> - TSP is computationally hard (factorial growth)
> - Local search is a universal framework (4 pillars)
> - Greedy construction gives fast initial solutions
> - 2-opt improves solutions iteratively
> - Multi-start helps escape local optima
> - Real constraints (time windows) add complexity
> - Two-phase approach: Build then improve!

Bibliography