r/ThingsYouDidntKnow • 2 mo. ago
TheStocksGuy

Explanation of the Mathematical Equations in TSP Code

Explanation of the Mathematical Equations in TSP Code -Exactly.py
`https://github.com/BadNintendo/tsp/blob/main/resolved/Exactly.py`

This code solves the **Traveling Salesman Problem (TSP)** by using **Morton ordering**, a **nearest-neighbor heuristic**, and **2-opt optimization**. Here's a breakdown of the key equations involved:

1. Distance Calculation between Cities

- To calculate the Euclidean distance between two cities (`city1` and `city2`) with coordinates `(x1, y1)` and `(x2, y2)`, we use the formula:
- `distance = sqrt((x2 - x1)^2 + (y2 - y1)^2)`
- This computes the straight-line distance between two points in 2D space.

2. Total Distance of a Path

- The total distance `D` of a path that visits `n` cities in sequence (path = `[p0, p1, ..., pn]`) and returns to the start city is calculated as:
- `D = sum(distance(p_i, p_(i+1 % n)) for i in range(n))`
- In this formula, `distance(p_i, p_(i+1 % n))` calculates the distance between each consecutive pair of cities along the path, and `(i + 1) % n` ensures the path returns to the starting city by connecting the last city back to the first.

3. Morton Order (Z-order Curve)

- The Morton order (or Z-order) helps us order cities spatially by interleaving the bits of their `x` and `y` coordinates.
- **Scaling:** Each `x` and `y` coordinate is scaled by `10,000` to avoid floating-point issues.
- **Interleaving Bits:** The bits of `x` and `y` are then interleaved to form a Morton code:Shift bits and apply bit masks iteratively to create an interleaved bit pattern that provides spatial locality.
- This ordering groups cities that are close in both `x` and `y` coordinates, allowing us to process nearby cities more efficiently.

4. Nearest Neighbor Heuristic

- The nearest neighbor heuristic starts at the first city, then selects the closest unvisited city at each step until all cities are visited. At each step, it selects the city that minimizes:
- `distance(current_city, next_city)`
- This heuristic quickly generates a path by always choosing the nearest unvisited city from the current location.

+ Create

length. For any two edges `(p_(i-1), p_i)` and `(p_j, p_(j+1))`, we calculate:

- **Before Swap:** The distance sum of the current segments:before_swap = distance(p_(i-1), p_i) + distance(p_j, p_(j+1))
- **After Swap:** The distance sum if we reverse the segment between `i` and `j`:after_swap = distance(p_(i-1), p_j) + distance(p_i, p_(j+1))
- If `after_swap < before_swap`, then the swap reduces the path length, so we perform it. This process repeats until no further improving swaps are possible, resulting in an optimized path.

Validating the Solution

- Finally, the solution is validated by checking that:
- The tour visits each city exactly once.
- The path returns to the starting city.

1 0 Share Approved 2 months ago

Post Insights

Only the post author and moderators can see this

173	100%	0	0
Total Views	Upvote Rate	Comments	Total Shares

Hourly views for first 48 hours

Some insights are no longer available because this post is older than 45 days

Add a comment

Be the first to comment

Nobody's responded to this post yet.
Add your thoughts and get the conversation going.