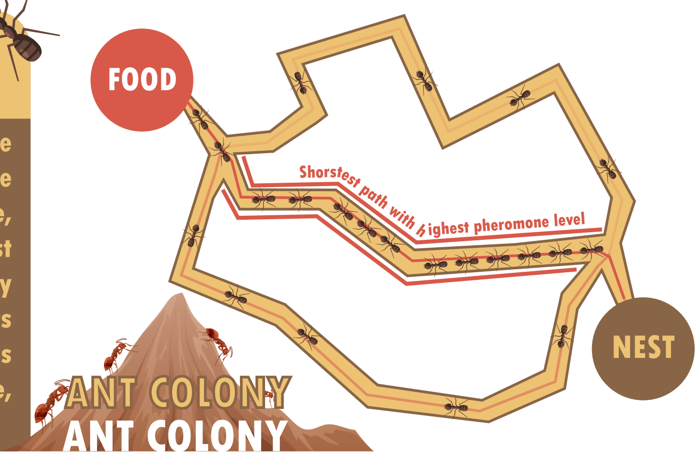
Solving Traveling Salesman Problem through the application of Ant Colony Optimization Algorithm

Introduction

The traveling salesman problem (TSP) derives from the reality that the "salesman" must find a tour that visits every city he knows once, while minimizing travel costs by reducing the distance traveled. Meanwhile, scientists are astounded by the fact that ants typically take the shortest route to food. Pheromone laid by ants moving on long paths will mostly be evaporated by the time a new ant revisit it. Only pheromone of ants that travels the shortest distance remains sensible. This occurrence is known as ant colony optimization (ACO). Due to similarities in nature, the ACO algorithm can be used to approximate TSP.



Ant Colony Optimization (ACO) Algorithm

IMPLEMENTATION

In each iteration, ants will start at every vertex. Each ant must traverse the graph in a **closed tour** for its share of pheromone deposition to be accepted. In constructing the tour, each ant k lists a set $A_k(x)$ of incident edges.

The probability can be calculated by the formula:

$$p^k_{xy} = \frac{(\tau^{\alpha}_{xy})(\eta^{\beta}_{xy})}{\sum_{z\in allowed_x}(\tau^{\alpha}_{xz})(\eta^{\beta}_{xz})}$$

- τ_{xy} The quantity of pheromone
- $\alpha \geqslant 0$ Parameter to control the effect of τ_{xy}
- η_{xy} The attractiveness of the edge xy
- $\beta \geqslant 1$ Parameter to control the effect of η_{xy}
- $\eta_{xz}\tau_{xz}$ The trail level and attractiveness for other possible edges

Volatility causes **pheromone levels** to fluctuate. After all ants have finished their tours, we'll update pheromone levels.

- The quantity of pheromone τ_{xy}
- Coefficient of pheromone evaporation ρ
- The number of ants m

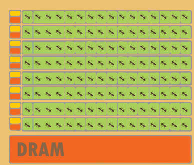
$$(1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau^k_{xy} \to \tau_{xy}$$

$\Delta\tau^k_{xy}$ Quantity of pheromone deposited by kth ant:

$$\Delta\tau^k_{xy} = \begin{cases} Q/L_k & \text{if ant } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases}$$

MULTIPROCESSING

The ACO algorithm is notable for its ability to run on large cluster of computers to solve TSP with millions of cities. To do so, we need to divide the problem into smaller parts. There are several ways:



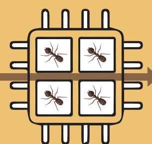
Matrices operations can be accelerated by parallel computation with GPUs → Do not have the resources (GPUs)

Pheromone update can be performed edgewise, updating independent segments of the adjacency matrix, which can be performed in parallel → Do not have the the coding expertise

NOT IMPLEMENTED

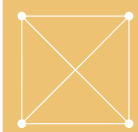
We decided to solve big TSP with the size of **20-200 cities.**

In an iteration, every ant's work is independent

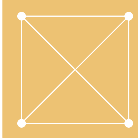


Build ACO algorithm that can be run parallel on multiple CPU cores

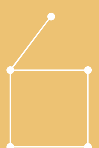
Test cases and time complexity



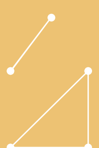
K_4 graph with weight 2→9



K_4 graph with weight from 9.992 to 9.999



A connected graph with 5 vertices



A disjoint graph of 5 vertices

Test on graphs that is generated randomly, with density (number of edges) is either 0.3 (sparse) to 0.9 (dense) relative to a complete graph with same number of vertices

Dense 50 vertices graph	Dense 100 vertices graph	Dense 200 vertices graph
1.84s	7.74s	48.40s
Sparse 50 vertices graph	Sparse 100 vertices graph	Sparse 200 vertices graph
1.28s	4.26s	25.27s

POLYNOMIAL TIME

$\sim O(n^2)$

for small number of vertices (50-200)