**Solving Traveling Salesman problem through the application of Ant Colony Optimization Algorithm (ACO algorithm)**

Nguyễn Thành Long

Vương Khả Siêu

Discrete Math

Instructors: Trần Vĩnh Linh

TA: Lê Hoàng Phúc, Nguyễn Thu Huyền

March 27th 2022

1. **Introduction**

Imagine a merchant life, where we must visit numerous cities to conduct trades. The travelling salesman problem (TSP) stems from this reality, where the “salesman” must find a tour that visit every city which he knows once, while saving costs by minimizing the distance travelled. Although TSP can be expressed so easily, it is a very difficult problem. The search for a globally optimal solution calculable in a reasonable time by mathematicians, who were fueled by the great applicability of such solution to many other fields, has so far not yielded so satisfiable results.

Meanwhile, scientists are astonished with the fact ants all travel on an often-shortest path to food. They studied the movement and concluded that the ants would sense the pheromone that preceded ants had released. Surprisingly, the very first, “explorer” ants travelled randomly, which suggests inefficiency. However, pheromone laid by ants moving on long paths will mostly be evaporated by the time a new ant revisit it. Hence, only pheromone on the shortest path remains strong enough to be sensible. Thus, contradicts to our initial assumptions, ants can make surprisingly good paths. This phenomenon is called ant colony optimization (ACO).

Because of similarities in nature, TSP can be approximated using ACO algorithm. In the algorithm, real world map is modelled as a weighted graph and the swarm as artificial ants whose functions are almost identical to real ants. This algorithm cannot be proven to always yield the best route due to its stochastic nature. However, we can believe it will produce a result that is approximately optimal in a reasonable amount of time. The proof in believing the correctness of this model is inspiration from nature selection, which evolved through millions of years. This fact gives us the hope of proving a modified version of ACO will universally solve TSP in the future.

1. **Ant Colony Optimization (ACO) Algorithm**
2. **An implementation of the algorithm**

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 of incident edges. For every edge xy in the set, the probability of choosing the edge is determined by a combination of two values: the attractiveness , computed by a function involving the edge’s weight, and the trail level which indicates the artificial pheromone level of the edge.

The probability can be calculated by the formula:

is the quantity of pheromone deposited for travelling from ***x***to ***y***

: parameter to control the effect of

: the attractiveness of the edge ***xy*** (typical , where ***d*** is the edge weight)

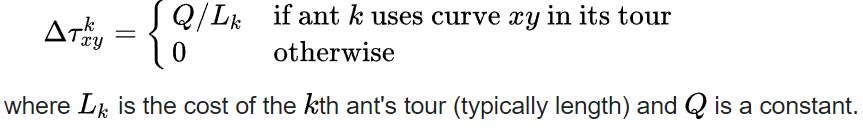
: a parameter to control the influence of

: The trail level and attractiveness for other possible edges.

However, pheromone levels will fluctuate due to pheromone volatility. Therefore, we will implement pheromone level update function which will run after all ants have finished their tours:

: the quantity of pheromone deposited on the edge ***xy***ρ: the coefficient of pheromone evaporation

***m****:* the number of ants.

: the quantity of pheromone deposited by ***kth*** ant: 

After a specified number of iterations, the algorithm stops, and the edge list is then sorted according to pheromone level in descending order. The first n edges are selected and printed out as the result. To improve accuracy, one can tweak the parameters as well as increasing the number of iterations.

1. **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. Firstly, in an iteration, every ant’s work is independent (1). Thus, we can let every computer in a cluster to be an ant. Secondly, matrices operations can be accelerated by parallel computation with GPUs (2). Thirdly, pheromone update can be performed edgewise, updating independent segments of the adjacency matrix, which can be performed in parallel (3). Therefore, the ACO algorithm can be easily modified to run on a large computer cluster.

To demonstrate we decided to solve big TSP with the size of 20-200 cities. We used argument (1) to build ACO algorithm that can be run parallel on multiple CPU cores. We do not have the resources (GPUs) to implement (2), or the coding expertise for (3), therefore we will not implement them in this project.

1. **Test cases and time complexity**

* graph with weight 2->9: test for basic usability and correctness. Result: **PASSED**, optimal solution on every run.
* graph identical to the above with weight in range 9.992 to 9.999: test algorithm stability on small differential weight. Result: **UNSTABLE**, output the optimal solution on ~50% of the run, other run results is close to optimal.
* A connected graph with 5 vertices: test how missing edges affect the outcome. Result: **PASSED**, optimal solution on every run for graphs with closed Eulerian tours; no tour is found for graphs with no Eulerian tour.
* A disjoint graph of 5 vertices: test how the algorithm will behave with disconnected graphs. Result: **PASSED**, no tour is found with disconnected graphs
* 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, result is expressed as execution time on i5-3427U CPU using all core:

1. Dense 50 vertices graph: 1.84 seconds
2. Sparse 50 vertices graph: 1.26 seconds
3. Dense 100 vertices graph: 7.74 seconds
4. Sparse 100 vertices graph: 4.26 seconds
5. Dense 200 vertices graph: 48.40 seconds
6. Sparse 200 vertices graph: 25.27 seconds

From the experimental runtime, we conclude that our algorithm run in polynomial time for small number of vertices (50-200).