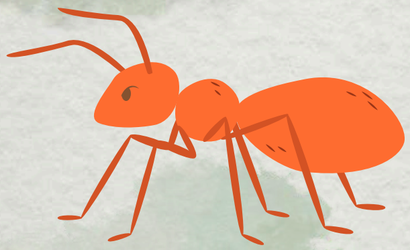
ANT COLONY OPTIMIZATION (ACO)



Description

ANTS LEAVE PHEROMONES ON THE PATH THEY TRAVELS. MAKING EACH PATH HAS 2 VALUES : PHEROMONE, COST (DISTANCE)

Description (2)

1. Ants travel to every node, changing pheromone of the paths they take
2. Best paths end up having the highest pheromones
3. Choosing next node by probability based on cost and pheromone

Formulas

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{L_k} & k^{th} ant\ travels\ on\ the\ edge\ i,j \\ 0 & otherwise \end{cases}$$

$$\tau_{i,j} = (1-\rho)\,\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}$$

With vaporization

Formulas

$$P_{i,j} = \frac{(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}}{\sum\left((\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}\right)}$$

$$where: \eta_{i,j} = \frac{1}{\mathrm{L}_{i,j}}$$

Sequential (Travel loop)

```
void travel(){
    // Loop for each ant
    for(int i = 0; i < totalAnts; i++){
        // current node (starts at node 0)
        int currNode = 0;
        bool nextNodeExist = true;
        // visited nodes
        vector<bool> visited(totalNodes, 0);
        float totalCost = 0; // Lk
        vector<pair<int, int>> traveledThrough;
        visited[0] = true;

        // loop until traveled every node
        for(int j = 0; j < totalNodes-1; j++){
            nextNodeExist = true;
            int nextNode = getNextNode(visited, currNode);
            if(nextNode < 0){
                nextNodeExist = false;
                break;
            }
```

Loop for n ant, each ant travels m nodes

Parallel (Travel loop)

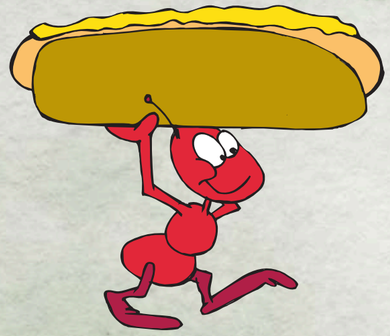
```
// Ant kth travel thorugh every node
if(gidx < totalAnts){
    for(int i = 0; i < totalNodes - 1; i++){
        // Calculate for next node
        // Getting totalProbs
        nextNodeExist = false;
        float probs[TOTAL_NODE];
        float totalProbs = 0;
        for(int l = 0; l < totalNodes; l++) probs[l] = 0;

        for(int j = 0; j < totalNodes; j++){
            if(!visited[j] && map[currNode*totalNodes+j].cost > 0){ // connects
                nextNodeExist = true; // probs is not all 0
                float Tij = map[currNode*totalNodes+j].pheromone;
                float nij = 1.0 / map[currNode*totalNodes+j].cost;
                probs[j] = Tij * nij;
                totalProbs += Tij * nij;
            }
        }
```

Call totalAnts number of threads, each thread represents kth ant

Sequential (Updating pheromones)

```
// update pheromone (every update old pheromone evaporates by evaRate)
void updatePheromones(){
    for(int i = 0; i < totalNodes; i++){
        for(int j = 0; j < totalNodes; j++){
            float oldPheromone = map[i][j].pheromone;
            map[i][j].pheromone = ((1-evaRate) * oldPheromone) + delta[i][j];
        }
    }
}
```

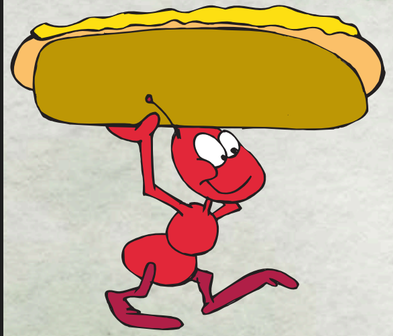


Sequential matrix operation

Parallel (Updating pheromones)

```cpp
__global__ void updatePheromones(int totalNodes, float evaRate, edges* map, float* delta){
    const int col = blockDim.x * blockIdx.x + threadIdx.x;
    const int row = blockDim.y * blockIdx.y + threadIdx.y;
    const int gidx = row*totalNodes + col;

    if(col < totalNodes && row < totalNodes){
        map[gidx].pheromone = (1-evaRate) * map[gidx].pheromone + delta[gidx];
    }
}
```



Parallelized matrix operation

Time comparison

(100 nodes, 60 ants, 20 epochs)

1. Sequential time

```
$ nvcc acoSequential.cu -o acoSequential && ./acoSequential.exe
nvcc warning : Support for offline compilation for architectures
DONE EPOCH 12
DONE EPOCH 13
DONE EPOCH 14
DONE EPOCH 15
DONE EPOCH 16
DONE EPOCH 17
DONE EPOCH 18
DONE EPOCH 19
Time taken by main: 911029 microseconds
```

2. Parallel time

```
$ nvcc acoParallel.cu -o acoParallel && ./acoParallel.exe
nvcc warning : Support for offline compilation for architect
DONE EPOCH 14
DONE EPOCH 15
DONE EPOCH 16
DONE EPOCH 17
DONE EPOCH 18
DONE EPOCH 19
Time taken by function: 276776 microseconds
```

Formulas

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

(NVIDIA, 2020a)

(Travel)

S = 1 / ((1 - 0.98) + (0.98 / 60))

= 27.5

(Pheromone update)

S = 1 / ((1 - 0.02) + (0.02 / 100*100))

= 1.02

Possible improvements

- Each thread (ant) use a lot of memory
- A lot of atomic operations on delta matrix
- Can be overall faster, based on the formula

Thank You