Implementation of Ant Colony Optimization Algorithm For Mobile Ad Hoc Network Applications: OpenMP Experiences

Mohammad Towhidul Islam
Parimala Thulasiraman
Ruppa K. Thulasiram

Presented by James Freckleton
Presentation Outline

- Optimisation problem to solve: Routing in mobile ad hoc networks
- Ant colony optimisation
- An ant-based solution: The source update algorithm
- OpenMP implementation of the algorithm
Mobile Ad Hoc Networks

- These networks consist of a group of mobile wireless nodes
- Nodes communicate in a distributed way
- Nodes dynamically form a network on the fly
- Each node operates as both host and router
- Nodes operate on low power batteries, which limits range
- Out-of-range nodes are reached through intermediate nodes (or hops)
Mobile Ad Hoc Networks

- Used when the geographical nature of the system is unknown
- Also used in situations where communication must be entirely distributed
- Applications include:
  - Health
  - Military
  - One Laptop Per Child organisation
  - VANet (Vehicular ad hoc networks)
Ant Colony Optimisation

- Used for “hard” problems that can be reduced to finding optimal paths through graphs
- Stigmergy: Ants communicate by modifying their local environment (laying pheromone)
- Positive feedback: The higher the pheromone content on a path, the greater probability it is a good solution
- Ants act like mobile nodes in MANets: they both create paths dynamically
Ant Colony Optimisation

- This is an inherently parallelisable metaheuristic, and so is appropriate to tackle routing in MANets
- This approach has been used to “solve” previously intractable instances of:
  - Graph colouring problem
  - Travelling salesman problem
  - Quadratic assignment problem
  - Vehicle routing problem
Previous Approaches To Implementing MANets

- Multipoint relays
  - These are selected nodes which forward broadcast messages during the flooding process

- Hybrid routing
  - Reactive: discovers paths only when required
  - Pro-active: active paths rebuilt from scratch every 3 seconds

- Route request messages
  - Contains source, destination, and lifespan data
Ant Colony Optimisation Algorithm for MANets

- MANet is a graph, $G = (V, E)$; the problem is to find the best path between source node $S$ and destination node $D$
- Each edge $e(v_i, v_j)$ has an amount of pheromone $\varphi(v_i, v_j)$ and the connection “time”, $w(v_i, v_j)$
- Each ant maintains a VisitedHop array, an elapsed exploration time TotalTime, and a Stack containing all nodes that may give a promising path to $D$
- Each node $i$ has a routing table of size $N * d_i$, where $N$ is the number of nodes and $d_i$ is the degree of $i$
- The algorithm is run on all possible $S$-$D$ pairs
The Source Update Algorithm Over A Single S-D Pair

TotalTime $\leftarrow 0$
Stack $\leftarrow (S, \text{TotalTime})$
VisitedHop[S] $\leftarrow 1$
Current $\leftarrow S$

While (Current $\neq D$)

NextHop $\leftarrow$ (empty)
if (exists(unvisited adjacent node))

NextHop $\leftarrow$ (unvisited adjacent node)
else

NextHop $\leftarrow$ unvisited(max{$\phi$(current, adjacent nodes)})
if (NextHop = (empty))

Pop Stack
(Current, TotalTime) $\leftarrow$ Stack
else

MoveTo(NextHop)
MoveTo(NextHop)

PreHop ← Current
Current ← NextHop
TotalTime ← TotalTime + W(PreHop,Current)

/* Update pheromone concentration for path from S to the current node */
/* ε is a user-supplied parameter */
φ(PreHop,S) ← φ(PreHop,S) + ε / (T(S,PreHop) + w(PreHop,Current))

/* Evaporate pheromone on other paths by a given quantity, E */
φ(vₐ,S) ← (1-E)φ(vₐ,S), for every adjacent vₐ

Stack ← (Current, TotalTime)
VisitedHop[Current] ← 1
The Source Update Algorithm Over A Single S-D Pair

/* Launch backward ant after forward path from S to D has been completed */
While (Current ≠ S)
   PreHop ← Current
   (Current, TotalTime) ← Stack
   T’ = T(S,D) – TotalTime

   /* Update the pheromone quantity of the active link */
   ϕ(Current,D) ← ϕ(Current,D) + ε / T’

   /* Update the routing table for the Current node (not shown here) */

   Stack ← (Current, TotalTime)
   VisitedHop[Current] = 1
An Example Illustrating the Source Update Algorithm

- Consider the following network:

- Assume the ant is moving from source $S = 3$ to destination $D = 6$, and assume the ant has selected the next link as $\text{NextHop} = 4$
- As a result, the current node $\text{Current}$ is set to 4
- At this point, node 4’s routing table is updated by the ant
An Example Illustrating the Source Update Algorithm

- Next hop node isn’t chosen by looking at the individual links from *Current* to its neighbour.
- Instead, we look at the best path that will reach a destination from a particular node.
- Since all nodes have been visited by previous ants, this ant looks at pheromone concentration and picks $\text{NextHop} = 5$.
- This is reflected in the updated routing tables.
An Example Illustrating the Source Update Algorithm

Before the source update:

<table>
<thead>
<tr>
<th>Network Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>97.6</td>
<td>8</td>
<td>111.7</td>
<td>5</td>
<td>38.7</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>11.7</td>
<td>4</td>
<td>32.8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

After the source update:

<table>
<thead>
<tr>
<th>Network Node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>97.6</td>
<td>8</td>
<td>111.7</td>
<td>5</td>
<td>27.1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>11.7</td>
<td>4</td>
<td>57.8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
OpenMP Implementation

- The above algorithm was designed to be implemented on parallel processors
- OpenMP is a language supporting such parallelisation (along with multithreading)
- Tests were run on an 8-node 7.5GB shared memory architecture running Linux RedHat 7.1
- Each node contained an Intel Xeon processor, clocked at 700 MHz
- All networks were generated by the NETGEN random graph generator
Results

Figure 1: The performance of the algorithm is dependent on the number of nodes and ants.

Figure 2: The performance of the algorithm when taking into account various scheduling methods.
Evaluation

- The analysis and simulations of the protocol on shared memory architectures were strong.
- However, no realistic simulations among MANets were undertaken, and so bandwidth constraints may affect results.
- The paper itself presents a thorough overview; it is rarely vague on details (such as “best path”).
Summary

- Topology of Mobile ad hoc networks is chaotic
- Ant Colony Optimisation is an inherently parallelisable search technique
- It has similar properties to MANets
- A new ACO-based algorithm finds optimal routes
- OpenMP was used to implement this algorithm
- This implementation compares favourably with others such as MPI