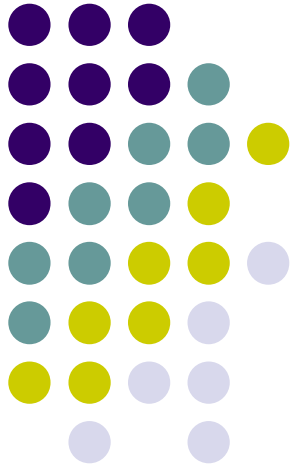
Implementation of Ant Colony Optimization Algorithm For Mobile Ad Hoc Network Applications: OpenMP Experiences

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)

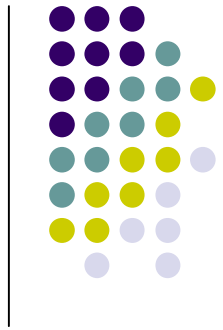
Ant Colony Optimisation

- 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

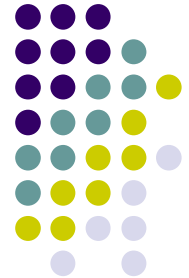
- This is an inherently parallelisable metaheuristic, and so is appropriate to tackle routing in MANets
- This approach has been used to “solve” previously intractable instances of:
  - Graph colouring problem
  - Travelling salesman problem
  - Quadratic assignment problem
  - Vehicle routing problem

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 ← 0
Stack ← (S, TotalTime)
VisitedHop[S] ← 1
Current ← S

While (Current ≠ D)
    NextHop ← (empty)
    if (exists(unvisited adjacent node))
        NextHop ← (unvisited adjacent node)
    else
        NextHop ← unvisited(max{φ(current,adjacent nodes)})
    if (NextHop = (empty))
        Pop Stack
        (Current, TotalTime) ← 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 */
$\varphi(v_a,S) \leftarrow (1-E)\varphi(v_a,S)$, for every adjacent v_a

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'

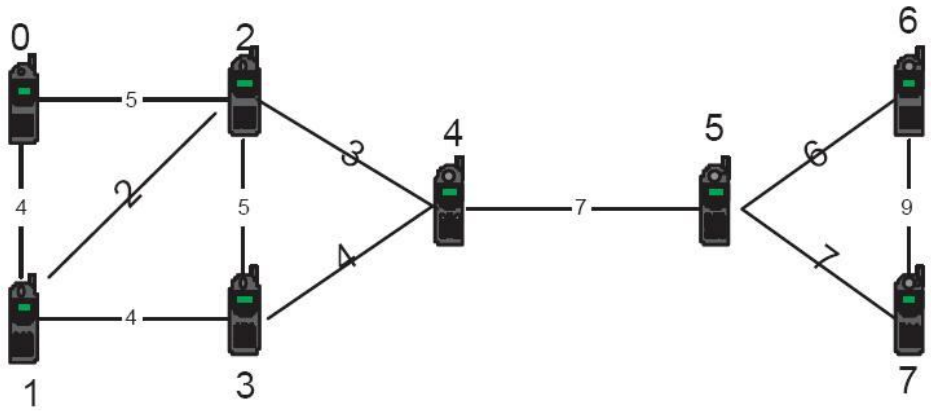
/* Update the routing table for the Current node (not shown here) */

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 *NextHop = 4*
- As a result, the current node *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 *NextHop = 5*
- This is reflected in the updated routing tables

An Example Illustrating the Source Update Algorithm

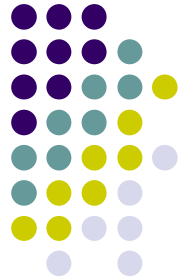
Before the source update:

Network Node	0		1		2		**3**		4		5		*6*		7		Total	
2	1	1	97.6	8	111.7	5	**38.7**	**5**	0	0	0	1	*0*	*2*	0	1	249	23
3	42	7	1	1	11.7	4	**32.8**	**4**	0	0	0	0	*0*	*1*	0	1	87.5	18
5	0	0	0	1	0	0	**0**	**1**	0	0	140	10	*67*	*10*	64	10	271	34

After the source update:

Network Node	0		1		2		**3**		4		5		*6*		7		Total	
2	1	1	97.6	8	111.7	5	**27.1**	**5**	0	0	0	1	*0*	*2*	0	1	237.4	23
3	42	7	1	1	11.7	4	**57.8**	**5**	0	0	0	0	*0*	*1*	0	1	112.5	19
5	0	0	0	1	0	0	**0**	**1**	0	0	140	10	*67*	*11*	64	10	271	34

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

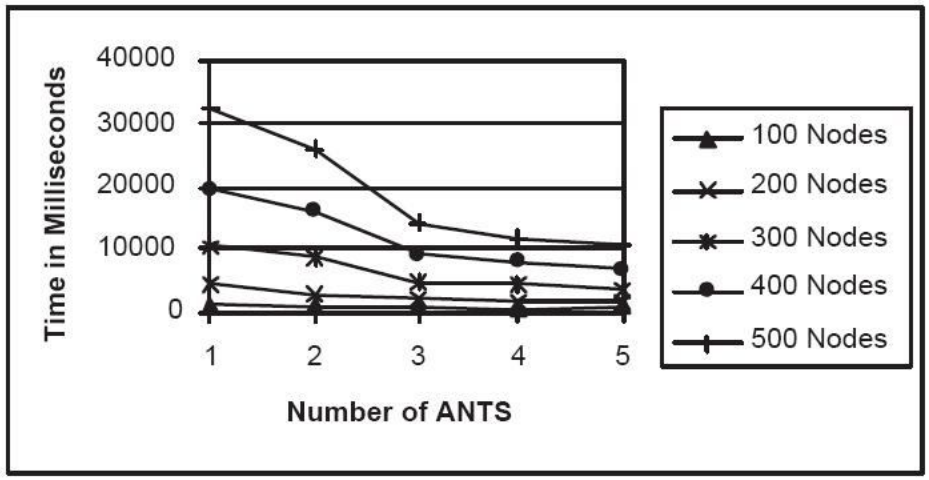


Figure 1: The performance of the algorithm is dependent on the number of nodes and ants

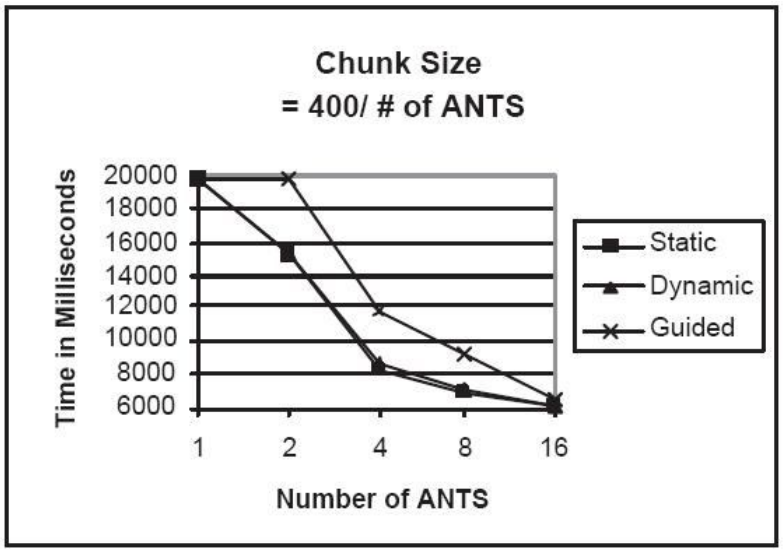
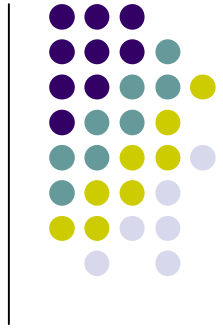


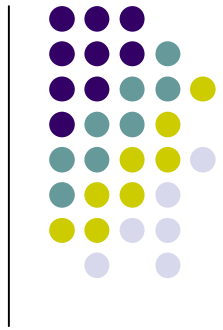
Figure 2: The performance of the algorithm when taking into account various scheduling methods

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