A COMPARATIVE STUDY ON MULTICAST ROUTING
USING DIJKSTRA’S, PRIMS AND ANT COLONY SYSTEMS

Sudip Kumar Sahana
Dept. Of Computer Science & Engg
Birla Institute of Technology
Mesra, Ranchi, E-Mail: sudipsahana@gmail.com

Dr.(Mrs). Aruna Jain
Dept. of Information Technology
Birla Institute of Technology
Mesra, Ranchi, E-Mail: jaruna16@rediffmail.com

Abijit Mustafi
Dept. Of Computer Science & Engg
Birla Institute of Technology
Mesra, Ranchi, E-Mail: abhijit@bitmesra.ac.in

ABSTRACT
Dijkstra’s and Prim’s algorithms are well established methods for implementing multicast routing. We have emphasized a different approach ‘Ant Colony System (ACS)’ for multicast routing. This paper introduces a comparative study of implementing simple Ant Colony System, Dijkstra’s Algorithm and Prim’s Algorithm in a multicast routing communication scenario. For the sake of simplicity we have only considered the distance as the key parameter to choose the best path. We have analyzed directed as well as undirected graphs. ACS, an optimization approach is considered here as undirected system for generalization. The complexity study shows that ACS has faster convergence to reach the optimal solution.

Keywords: Dijkstra’s Algorithms, Prim’s Algorithm, Ant Colony System, Pheromone, Convergence.
1. INTRODUCTION

1.1 Dijkstra’s Algorithm

Let the starting node be called as initial node and the distance of node Y be the
distance from the initial node to Y. Dijkstra's algorithm[1] will assign some initial
distance values and will try to improve them step by step.

It is an algorithm to find shortest path[2] of routing in a network.

Distance Matrix \[ D = d_{ij} = 0 \quad \text{if } i=j. \]
\[ d_{ij} = \infty \quad \text{if there is no path between } i \text{ and } j. \]
\[ d_{ij} = \text{distance of the path from } i \text{ to } j. \]

1. Assign to every node a distance value. Set it to 0 for our initial node and to
infinity for all other nodes.

2. Mark all nodes as unvisited. Set initial node as current.

3. For current node, consider all its unvisited neighbors and calculate their tentative
distance \( d_{ij} \) (from the initial node). For example, if current node (A) has distance
of 6, and an edge connecting it with another node (B) is 2, the distance to B
through A will be 6+2=8. If this distance is less than the previously recorded
distance (infinity in the beginning, zero for the initial node), overwrite the
distance.

4. When we are done considering all neighbors of the current node, mark it as
visited. A visited node will not be checked ever again; its distance recorded now
is final and minimal.

5. If all nodes have been visited, finish.

Otherwise, set the unvisited node with the smallest distance (from the initial node) as the
next "current node" and continue from step 3.

1.2 Prim’s Algorithm

It determines a minimum spanning tree [2] for a connected weighted undirected
graph. Here, objective is to find a subset of the edges that forms a tree, which includes
every vertex, where the total weight of all the edges in the tree is minimized.
Algorithm:
Input: A connected weighted graph with vertices V and edges E.
Initialize: (i) \( V_{\text{new}} = \{ X \} \), where X is an arbitrary node from V, \( E_{\text{new}} = \{ \} \)
repeat until \( V_{\text{new}} = V \);
(ii) Choose edge \( (u,v) \) with minimal weight such that u is in \( V_{\text{new}} \)
and \( V \) is not in \( V_{\text{new}} \).
(iii) Add v to \( V_{\text{new}} \), add \( (u,v) \) to \( E_{\text{new}} \).
Output: \( V_{\text{new}} \) and \( E_{\text{new}} \) describe a minimal spanning tree

1.3 ANT COLONY SYSTEM (ACS)

Ant colony algorithm [3] determines optimal solution through simulating the process of ants searching for food. The ants collective behavior reflects an information positive feedback phenomenon. This optimization technique does not rely on mathematical description of the specific issues, but has strong global optimization feature [4], high performance [5] and flexibility. Three main aspects to determine ACS are:

1) State Transition Rule: Ants prefer to move from one place to another (i.e one node to other node) which are connected by short edges with a high amount of pheromone [3]. It can be done by using following rule:

\[
\text{PI}_k (r,s) = \begin{cases} 
\tau(r,s)\cdot [\eta(r,s)]^\beta, & \text{if } s \in J_k (r) \\
\sum_{u \in J_k (r)} \tau(r,u)\cdot [\eta(r,u)]^\beta, & \text{otherwise}
\end{cases}
\]

2) Local pheromone updating rule: While building a solution, ants visit edges and change their pheromone level by applying this rule:

\[
\tau(r,s) \leftarrow (1-\rho)\cdot \tau(r,s)+\rho\cdot \Delta \tau (r,s).
\]

Where \( 0 < \rho < 1 \) is a pheromone evaporation [3] parameter, and \( \Delta \tau (r,s) = \tau_0 \) where \( \tau \) is pheromone. We here assume a well accepted value of \( \rho = 0.5 \).

3) Global pheromone updating rule: Once all ants have built their tours, pheromone is updated on all edges by using the following rule:
τ(r,s) ← (1-α)*τ(r,s)+α*Δτ(r,s).

where 0<α<1 is pheromone decay [4],[6] parameter and we assume α=0.2 to get a better effect of probability on the globally shortest path.

Where Δτ(r,s)=

\[
\begin{cases} 
1/L_{gb} & \text{if } (r,s) \in \text{global best tour} \\
0 & \text{otherwise} 
\end{cases}
\]

and Lgb is length of globally best tour.

2. IMPLEMENTATION

Part I: Description of the example network.

Part II: Results of Dijkstra’s Algorithm considered for directed graph of example network for an instance.

Part III: Results of Prims Algorithm considered for undirected graph of example network for an instance.

Part IV: Results of ACS Algorithm considered for undirected graph of example network for an instance.

Part V: Overall Results Analysis and Complexity Analysis.

2.1 PART I:

Description of the network: In multicast network [7] one source can transmit packets to more than one destination at a time. In practical communication scenario, there may be different constraints [5] like bandwidth, delay, hop count, packet loss, etc. We are considering only distance as a main constrain for the sake of simplicity.

We have experimented on a set of networks having different vertices ranging from 5 to 25 and edges from 5 to 10. We have implemented the program on C language and Windows environment in a stand-alone machine. For a better understanding we are taking an instance of the example network where five nodes (vertices) and ten paths(edges) are considered as shown below in figure1. Here ‘r’, ‘v’ and‘ s’ are the source node respectively. And ‘u’ and ‘t’ are the destination nodes.
2.2 PART II:

Using Dijkstra’s approach for the said network the final solution can be tabulated as follows where Best(t) represents the best path distance from source node to the specified node and Tree(i) represents the intermediate nodes to reach the destination.

Taking ‘r’ as the source node, distance vector for rest of the nodes are as follows:

<table>
<thead>
<tr>
<th>‘r’ (source node)</th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best(i)</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Tree(i)</td>
<td>r</td>
<td>v</td>
<td>v</td>
<td>r</td>
</tr>
</tbody>
</table>

Taking ‘v’ as the source node, distance vector for rest of the nodes are as follows:

<table>
<thead>
<tr>
<th>‘v’ (source node)</th>
<th>r</th>
<th>s</th>
<th>t</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best(i)</td>
<td>11</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Tree(i)</td>
<td>t</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
</tbody>
</table>

Taking ‘s’ as the source node, distance vector for rest of the nodes are as follows:

<table>
<thead>
<tr>
<th>‘s’ (source node)</th>
<th>t</th>
<th>u</th>
<th>v</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best(i)</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Tree(i)</td>
<td>s</td>
<td>s</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

2.3 PART III:
For the above network in figure 2 (considered as undirected system) using Prims approach, the final solution can be tabulated as follows where Best d(i) represents the best path distance from source node to the specified node.

Taking ‘r’ as the source node

<table>
<thead>
<tr>
<th></th>
<th>r</th>
<th>v</th>
<th>s</th>
<th>u</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best d(i)</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Taking ‘v’ as the source node

<table>
<thead>
<tr>
<th></th>
<th>v</th>
<th>s</th>
<th>u</th>
<th>t</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best d(i)</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Taking ‘s’ as the source node

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>t</th>
<th>u</th>
<th>v</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best d(i)</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

2.4 PART IV:

The cost and pheromone intensity of the paths are:

- Cost of (r,v)=(r,u) = 2, Pheromone intensity = 4.
- Cost of (r,u)=(r,t) = 6, Pheromone intensity = 1.4.
- Cost of (v,u)=(v,s)=(u,t)=(s,t) = 4, Pheromone intensity = 2.
- Cost: (v,t)=(s,u) = 5, Pheromone intensity = 1.6.

Taking ‘r’ as source node, final optimal solution is obtained after 3 iteration which is the average case of complexity. The final shortest path is path (rvu) with path length 6.

Taking ‘v’ as the source node, final solution is obtained after 4 iterations which is the worst case of complexity. The final shortest path is path (vu) with path length 4.

Taking ‘s’ as the source node, final solution is obtained after 3 iterations which is the average case of complexity. The final shortest path is path (su) with path length 5.

Similarly for t as destination node the same can be obtained.

2.5 PART V:

2.5.1 OVERALL RESULTS ANALYSIS:

The Table1 shown below depicts the summary of the run time results in milliseconds for varying number of vertices and varying number of edges but a constant
number of destination nodes (two) for multicast routing. In case of varying number of destination nodes, the algorithms exhibit same hierarchy of performance.

### Table 1

<table>
<thead>
<tr>
<th>Edge</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
<th>Prim's</th>
<th>Dijkstra's</th>
<th>ACS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex=10</td>
<td>100</td>
<td>210</td>
<td>16.6096</td>
<td>100</td>
<td>212</td>
<td>19.316</td>
<td>100</td>
<td>214</td>
<td>23.2535</td>
<td>100</td>
<td>216</td>
<td>26.5754</td>
<td>100</td>
<td>218</td>
<td>29.9974</td>
<td>100</td>
<td>220</td>
<td>33.2193</td>
</tr>
<tr>
<td>Vertex=20</td>
<td>400</td>
<td>810</td>
<td>21.6096</td>
<td>400</td>
<td>812</td>
<td>25.8356</td>
<td>400</td>
<td>814</td>
<td>30.2535</td>
<td>400</td>
<td>816</td>
<td>34.5754</td>
<td>400</td>
<td>818</td>
<td>38.8974</td>
<td>400</td>
<td>820</td>
<td>43.2193</td>
</tr>
<tr>
<td>Vertex=25</td>
<td>625</td>
<td>1260</td>
<td>23.2193</td>
<td>625</td>
<td>1262</td>
<td>27.8631</td>
<td>625</td>
<td>1264</td>
<td>32.507</td>
<td>625</td>
<td>1266</td>
<td>37.1508</td>
<td>625</td>
<td>1268</td>
<td>41.7947</td>
<td>625</td>
<td>1270</td>
<td>46.4086</td>
</tr>
</tbody>
</table>

The Table 2 shown below depicts the summary of the run time results in milliseconds for varying number of vertices and varying number of edges but a constant number of destination nodes (two) for multicast routing for advanced performance analysis of the said algorithms implemented using heap data structure. In case of varying number of destination nodes, the algorithms exhibit same hierarchy of performance.

### Table 2

<table>
<thead>
<tr>
<th>Edge</th>
<th>Ext-Prim's</th>
<th>Ext-Dijkstra's</th>
<th>ACS</th>
<th>Ext-Prim's</th>
<th>Ext-Dijkstra's</th>
<th>ACS</th>
<th>Ext-Prim's</th>
<th>Ext-Dijkstra's</th>
<th>ACS</th>
<th>Ext-Prim's</th>
<th>Ext-Dijkstra's</th>
<th>ACS</th>
<th>Ext-Prim's</th>
<th>Ext-Dijkstra's</th>
<th>ACS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex=15</td>
<td>63.6034</td>
<td>64.6034</td>
<td>21.6096</td>
<td>64.6034</td>
<td>54.6865</td>
<td>27.942</td>
<td>65.6034</td>
<td>62.507</td>
<td>31.2535</td>
<td>66.6034</td>
<td>70.324</td>
<td>39.0689</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex=20</td>
<td>91.4386</td>
<td>92.4386</td>
<td>23.2193</td>
<td>92.4386</td>
<td>69.1508</td>
<td>34.5754</td>
<td>93.4386</td>
<td>77.7947</td>
<td>43.2193</td>
<td>94.4386</td>
<td>86.4386</td>
<td>43.2193</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex=25</td>
<td>121.0964</td>
<td>122.0964</td>
<td>23.2193</td>
<td>122.0964</td>
<td>95.014</td>
<td>32.507</td>
<td>123.0964</td>
<td>124.0964</td>
<td>74.3017</td>
<td>125.0964</td>
<td>126.0964</td>
<td>46.4086</td>
<td>127.0964</td>
<td>128.0964</td>
<td>46.4086</td>
</tr>
</tbody>
</table>
A visual representation of dataset of Table 1 and Table 2 are as shown below:

![Graph 1](image1.png)

![Graph 2](image2.png)

### 2.5.2 COMPLEXITY ANALYSIS:

Our first simple implementation of the Dijkstra's algorithm stores vertices of set $Q$ in an ordinary array, and extract minimum from $Q$ is simply a linear search through all vertices in $Q$. In this case, the running time is $O(|V|^2 + |E|) = O(|V|^2)$. We have further implemented it using heap data structure and the complexity reduced to $O(|E|\log|V|)$.

A simple implementation using an adjacency matrix graph representation and searching an array of weights to find the minimum weight edge to add requires $O(|V|^2)$ running time. The time required by Prim's algorithm is $O(|V|^2)$. Using a binary heap data structure and an adjacency list representation, Prim's algorithm requires run in time $O(|V|)$.
+ E) log(V)) where E is the number of edges and V is the number of vertices. Using a more sophisticated Fibonacci heap, this can be brought down to O (E + V log V). Where as an easy ACS implementation produces O (E log V) complexity.

According to various parameters the comparative study of these algorithms are tabulated as follow:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Prim’s</th>
<th>Dijkstra’s</th>
<th>ACS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>(a) Using Array: O(V^2)</td>
<td>(a) Using Array: O(m*(V^2+E))</td>
<td>(a) Using Array/Adjacency List: O(E log V)</td>
</tr>
<tr>
<td></td>
<td>(b) Using Heap: O(V+E) log V</td>
<td>(b) Using Heap: O(m*</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>(c) Using Fibonacci Heap: O(E +V logV)</td>
<td>Where m=no of destination nodes</td>
<td></td>
</tr>
<tr>
<td>Nature of the graph used</td>
<td>Undirected</td>
<td>Directed</td>
<td>Undirected</td>
</tr>
<tr>
<td>Convergence speed</td>
<td>Medium</td>
<td>Medium</td>
<td>Faster</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Do not allow</td>
<td>Do not allow</td>
<td>Allow constraints</td>
</tr>
<tr>
<td>Dynamic Approach</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Incur any Extra cost</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

3. CONCLUSION:

Though Dijkstra’s and Prim’s approaches are reliable and are in use, ant colony system can compete for the multicast implementation as we have already discussed the different advantages of ACS over other two. Complexity is relatively better in the case of ACS. The main advantages of ACS are it’s flexibility, dynamicity and the capability to find alternate paths. If there is congestion in the shortest path, ACS can dynamically route the packets on the alternative path. Ant colony system can be applied to different routing problems like vehicle routing problems [8][9], MANET[10], Network Flow, etc.

REFERENCES:


