ENHANCED ANT COLONY OPTIMISATION IN NETWORKS

Ho Xu Yang Damian[1], Liew Zhao Wei[1], Sheyuan Yeo[1], Chua Wei Ting[2], Ho Wei Xiong[2]

[1]Hwa Chong Institution (Secondary), 661 Bukit Timah Road, Singapore 269734

[2]DSO National Laboratories, 12 Science Park Drive, Singapore 118225

ABSTRACT

Stigmergic approaches to network routing such as the Ant Colony Optimisation (ACO) inspired AntNet [1, 2] algorithm are able to effectively provide load balancing in networks, while traditional routing algorithms such as Open Shortest Path First (OSPF) and Routing Information Protocol (RIP) are only able to provide equal-cost load balancing.

However, we observed that AntNet suffers from poorer performance on networks with articulation points, resulting in decreased throughput and increased packet latency. As such, our project aims to improve upon AntNet by developing an algorithm that is able to provide robust load balancing.

Simulation results suggest that our algorithm provides more robust performance than AntNet, in terms of both throughput and latency. As such, our algorithm serves as a more viable alternative to existing routing algorithms.

1. INTRODUCTION

Traditional routing algorithms such as OSPF, a link-state routing protocol, and RIP, a distance-vector routing protocol, aim to minimise a given cost function (usually propagation delay). Both algorithms route packets along the path of the minimum cost from source to destination. However, this can result in congestion along the minimum cost path, as alternative, more expensive, paths are not utilised despite being less congested.

AntNet is an alternative routing algorithm, based on mobile agents, that addresses the issue of congestion by distributing load amongst all possible paths, therefore improving network performance by reducing the likelihood of congestion along minimum cost paths.

In this paper, we present the algorithm described in [1, 2] as AntNet, discuss its limitations and ultimately describe our novel algorithm, Enhanced Ant Colony Optimisation (EACO). EACO, as well as AntNet and OSPF were implemented in Java. Through trials on various networks in a simulation, a better worst-case and average-case performance is demonstrated for EACO as compared to AntNet, in terms of throughput and packet latencies.

2. METHODOLOGY

2.1. Design Choices

In implementing AntNet, we made several design choices (some of which were inspired by other prior work done) that somewhat differ from existing implementations. This section thus aims to clarify the differences that our implementation possesses.

In the following subsections, we use the following variables:

- N_k: The neighbour nodes of node k.
- n_k: The cardinality of N_k.
- τ^k: The pheromone matrix of node k. τ_{ij}^k refers to the goodness (i.e. the probability) of visiting neighbour j for the purpose of reaching destination i.
- R^k: The data-routing table of node k (with a similar structure to τ^k). It is obtained by an exponential transformation and subsequent re-normalisation of the corresponding values in τ^k.
- η_{ij}: The heuristic value assigned to the link between nodes i and j.

2.1.1. Node Initialisation

The AntNet papers do not specify an initialisation method for the routing tables and as such, in most implementations a uniform distribution of probabilities is assumed. That is, for a neighbour node $i \in N_k$ and a destination d:

$$\tau_{di}^k = \frac{1}{n_k} (1)$$

However, Barán and Sosa [3] propose a new initialisation method which utilises each nodes' knowledge of the network topology, where a greater initial probability is assigned to neighbour nodes that could also be destinations. This was evaluated in a survey [4] as having the potential to save network resources, thus we utilise this version in our implementation. For a given node k:

If a destination node d is also a neighbour node (i.e. $d \in N_k$),

Node d will have an initial probability of:

$$\tau_{dd}^k = \frac{1}{n_k} + \frac{3}{2} * \frac{(n_k - 1)}{(n_k)^2} (2)$$

While the other neighbour nodes ($i \neq d$) will have initial probabilities of:

$$\tau_{di}^k = \frac{1}{n_k} - \frac{3}{2} * \frac{1}{(n_k)^2} (3)$$

If the destination node d is not a neighbour node,

A uniform distribution is assumed as per (1).

2.1.2. Node Routing

While Stüzle and Hoos [5] define η_{ij} as a function of the arc length d_{ij} ($\eta_{ij} = \frac{1}{d_{ij}}$), in AntNet it is a function of the queue length q_{ij} (number of bits waiting to be sent) along the link (i, j), as follows:

$$\eta_{ij} = 1 - \frac{q_{ij}}{\Sigma k \in N_i(q_{ik})} \quad (4)$$

In our implementation, we choose to use the formula described in [5], where we define d_{ij} as the propagation delay of the link (i, j) in ms.

Next, in [5], the probability p_{di}^k of picking a neighbour node i at node k for a given destination d is given by:

$$p_{di}^k = \frac{(R_{di}^k)^\alpha (\eta_{ki})^\beta}{\Sigma x \in N_k (R_{dx}^k)^\alpha (\eta_{kx})^\beta} \quad (5)$$

Where α and β represent adjustable weightages for R_{di}^k and η_{ki} respectively.

On the other hand, AntNet uses a simpler formula with only one adjustable weightage, α, that has an empirical value of 0.4:

$$p_{di}^k = \frac{R_{di}^k + \alpha(\eta_{ki})}{1 + \alpha(n_k - 1)} \quad (6)$$

We choose to implement an original, simplified version of (5), by defining β = 1 - α, while setting α = 0.4. In our testing, this formula gave better performance than (6).

$$p_{di}^k = \frac{(R_{di}^k)^\alpha (\eta_{ki})^{1-\alpha}}{\Sigma x \in N_k (R_{dx}^k)^\alpha (\eta_{kx})^{1-\alpha}} \quad (7)$$

2.1.3. Cycle Detection / Prevention

In AntNet, ants possess a private memory H, one purpose of which is to store a list $V_{v_0 \to v_m} = [v_0, v_1, ..., v_m]$, where v_i is the ID of the i^{th} node visited on the path. Ants are routed at nodes in a similar manner to packets, with the main difference being that τ^k is used instead of R^k. In addition, as far as possible, ants try to visit neighbour nodes that have not already been visited yet. It should be noted that data packets, on the other hand, do not possess such private memories.

At the current node k, in the case that all neighbour nodes have been already visited by the forward ant, there is an equal probability of visiting all neighbour nodes (excluding the node the ant just came from, unless there is exactly one neighbour node). In such a case, this forms a cycle in the ant's path. If the time spent in the cycle is more than half of the ant's lifespan, then the ant is killed. Otherwise, it merely deletes its memory of the cycle.

We choose instead to kill all ants that are caught in a cycle, given that the bound given was rather arbitrary and has no empirical results backing it. In addition, depending on the rate of ant generation, it is often better to let new ants explore the network, rather than rely on ants with

potentially outdated views of the network status (which would only serve to increase overhead in the network).

2.2. AntNet's Limitations

At nodes that are articulation points in the network, packets and/or ants may be routed towards *non-viable neighbour nodes*. For a current node k and with a destination d, we define non-viable nodes as the set of nodes that would not be connected to d (directly or indirectly) if k were to be removed from the graph.

The implications of routing to such non-viable neighbour nodes is two-fold.

Firstly, ants following such paths are doomed. There is no way for them to reach the destination without forming a cycle, in which case they would be killed by the cycle detection. This results in fewer ants successfully sampling the network status and as such the algorithm is slower to adapt to changes in the network topology.

Secondly, packets following such paths would have their journeys unnecessarily lengthened and might cause unneeded congestion in the network. Therefore, by not routing to non-viable neighbour nodes, the average latency of packets becomes more optimal, as there is a greater likelihood of them following relatively better paths.

2.3. EACO Algorithm

At each node we utilise a disjoint-set data structure (with path compression and union by rank) in order to determine which neighbour nodes are non-viable.

Disjoint-set data structures are a collection of sets that support two operations:

1. *Find*: Determines the subset a particular element is, by returning a representative element of the subset.
2. *Union*: Merges two subsets into a single subset.

In order to determine the viability of nodes, we calculate the various components the network would be divided into if the current node were to be removed.

We achieve this by merging the endpoints of all links, except links that have the current node as one of its endpoints.

A sample (Java) implementation is as follows:

```
UFDS DSU;
ArrayList<Node> nodes;
ArrayList<Edge> edgeList;
void init() {
    DSU = new UFDS(nodes.size());
    for (Edge E : edgeList) {
        if (E.source == nodeID || E.destination == nodeID) continue;
        if (nodes.get(E.source).isOffline || nodes.get(E.destination).isOffline) continue;
        if (E.isOffline) continue;
```

```
        DSU.union(E.source, E.destination)
    }
}
```

Algorithm 2.1. Disjoint-set Initialisation

Afterwards, for a current node k and a given destination d, the set of viable neighbour nodes is given by $\{i \in N_k : \text{find}(i)=\text{find}(d)\}$.

The amortised worst-case running time of each operation is $\Theta(\alpha(n))$ where n is the number of elements in the disjoint-set structure and α is the inverse Ackermann function, and it is proven in [6] that this is the fastest possible amortised running time. The inverse Ackermann function α is defined as:

$$\alpha(n) = min\{k : A_k(1) \geq n\} \quad (9)$$

Where $A_k(j)$ is the extremely fast growing Ackermann function, thus α is effectively a constant. The initialisation procedure for each node runs in $O(|L|\alpha(|N|))$ time, where $|L|$ and $|N|$ are the number of links and nodes in the network respectively.

3. RESULTS

For each network, we ran each algorithm with a single source and destination and uniform traffic, and benchmarked their average performance via several metrics:

1. Time required for convergent throughput: It is the time in milliseconds for the number of data packets routed successfully in the past second to be ≥ the number of data packets generated per second.
2. Time required for consistent throughput: It is the time in milliseconds for the number the number of data packets routed successfully in the past second to be consistently ≥ 95% of the number of data packets generated per second.
3. Average latency: This refers to the average latency of packets that reach the destination within each second.

In the following tables, we set the packet generation interval Δt to 100 ms. In [2], it was found that the highest AntNet normalised power (defined as the ratio between delivered throughput and 90th percentile of packet delay) is achieved with a Δt value of between 10 ms and 100 ms. Thus, this gives a good approximation of AntNet's optimal performance.

In addition, "Convergent" refers to metric 1 while "Consistent" refers to metric 2. In the network graphs, the lighter-coloured nodes represent the sources and destinations used in the simulation trials. The numbers on the links are their propagation delays in ms.

We assume each packet to be of size 1.5 kb (Ethernet maximum transmission unit), and we set Time To Live to 15 s.

For testing purposes, we implemented a discrete-event simulation, where each tick of the simulation represents 1 ms. The numbers presented in the performance comparison tables are the average over 10 trials, while the average latency graphs display the result of a single trial.

We evaluate the algorithm performances under an "ideal" network condition, where the amount of data generated each second does not exceed the bandwidth of any link. This enables us to use OSPF as a "benchmark", given that OSPF has optimal performance in such situations.

Our main focus is to compare the performances of AntNet and EACO, given that AntNet has already been compared extensively with other algorithms in prior work [1, 2, 3].

3.1. NSFNET

The NSFNET network is the old USA T1 backbone (1988), which was sponsored by the National Science Foundation. It has 14 nodes and 21 bidirectional links, each with a bandwidth of 1.5 Mbps. Propagation delays range from 4 to 20 ms. NSFNET is considered a well-balanced network. We generate 1 packet per tick (1.5 Mbps).

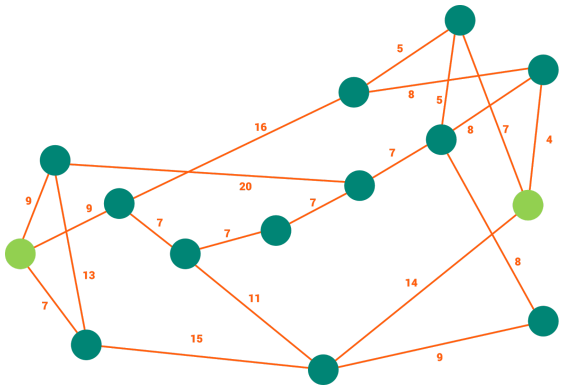


Figure 3.1. NSFNET

	AntNet	EACO	OSPF
Convergent [ms]	1064	1061	1036
Consistent [ms]	1008	1002	986.0

Table 3.1. NSFNET performance comparison

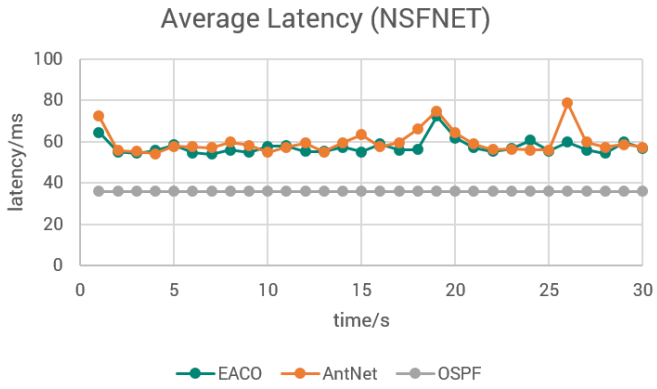


Figure 3.2. NSFNET average latency

3.2. NTTnet

The NTTnet network is the former Nippon Telephone and Telegraph company fiber-optic corporate backbone. It has 55 nodes and 162 bidirectional links, each with a bandwidth of 6 Mbps. Propagation delays range from 10 to 50 ms. NTTnet is considered a poorly balanced network. We generate 4 packets per tick (6.0 Mbps).

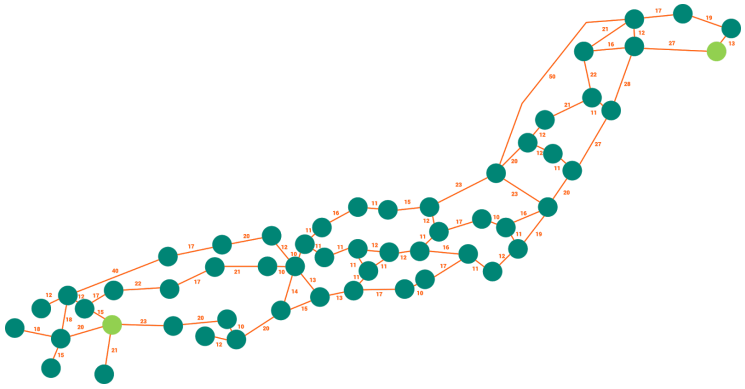


Figure 3.3. NTTnet

	AntNet	EACO	OSPF
Convergent [ms]	2083	1856	1272
Consistent [ms]	1941	1786	1222

Table 3.2. NTTnet performance comparison

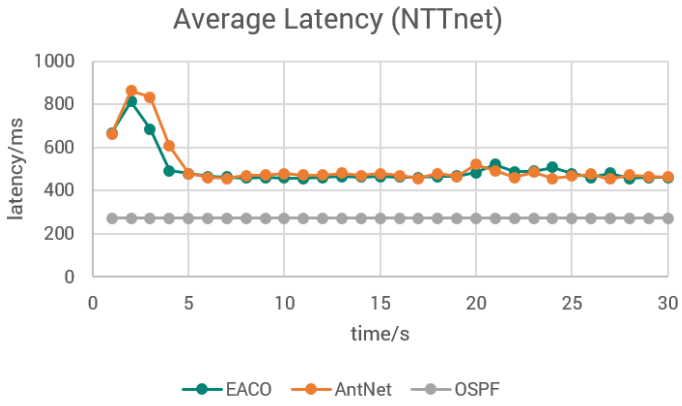


Figure 3.4. NTTnet average latency

3.3. Banana Tree

The Banana Tree network is a tree network. It has 30 nodes and 29 bidirectional links, each with a bandwidth of 3.0 Mbps and a propagation delay of 1 ms. It is a pathological case for AntNet. We generate 1 packet per tick (1.5 Mbps).

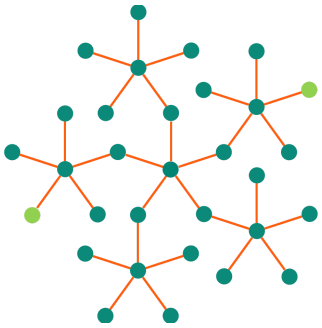


Figure 3.5. Banana Tree

	AntNet	EACO	OSPF
Convergent [ms]	1056	1039	1024
Consistent [ms]	1007	974.0	974.0

Table 3.3. Banana Tree performance comparison

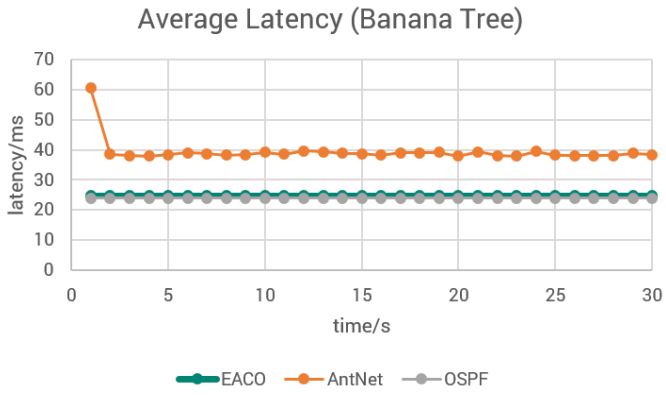


Figure 3.6. Banana Tree average latency

4. DISCUSSION

It was observed that none of the algorithms had a performance inconsistent to the extent that consistent throughput was obtained after convergent throughput.

4.1. NSFNET

As NSFNET is biconnected (i.e. there are no articulation points), AntNet and EACO have almost identical performances. In addition, as NSFNET is a well balanced network, both ACO algorithms showcase performance comparable to OSPF - similar convergent and consistent timings, and an average latency of only around 20 ms more.

4.2. NTTnet

In addition, the presence of 4 articulation points in the network causes AntNet to send the occasional packet or ant to a non-viable neighbour node, reducing adaptiveness and increasing packet latency.

EACO, on the other hand, is able to detect and avoid these non-viable nodes, and as such achieves consistent and convergent throughput faster than AntNet. In addition, average packet latency decreases faster.

Thus, it can be seen that EACO has a better average-case performance than AntNet.

4.3. Banana Tree

In the Banana Tree network, there is exactly one path from any node to another. Every internal node is an articulation point, making this a pathological case for AntNet.

EACO once again achieves consistent and convergent throughput faster than AntNet. In addition, EACO achieves consistent throughput in the same amount of time as OSPF. It can be observed that AntNet's average packet latency is consistently greater than both EACO and OSPF by about

15 ms, due to the presence of many articulation points causing packets to accidentally take roundabout paths.

Thus, it can also be seen that EACO has a better worst-case performance than AntNet.

5. CONCLUSION

In this project, we aimed to improve upon AntNet and develop an algorithm that was not susceptible to the limitations we identified.

To that end, the results demonstrate that EACO is able to offer a better worst-case and average-case performance as compared to AntNet, in terms of the time it takes to adapt to the network topology (to achieve consistent and convergent throughput) as well as the average latency of packets over time.

This makes EACO a much more robust algorithm as compared to AntNet, and as such serves as a more viable alternative for existing routing algorithms.

6. FURTHER RESEARCH

Through the course of this project, we have identified several key areas with potential for further research:

1. Alternative heuristic functions: Alternative definitions of the heuristic function d_{ij} could possibly lead to better performance. For example, instead of defining it as the propagation delay in ms of the link (i, j), defining it instead as the shortest distance from j to the destination could provide the algorithm with a better idea of the optimality of utilising the link.
2. Evaluation of performance under different levels of load: Setting the packet generation rate as a % of the maximum flow of the network could help provide a more objective evaluation of the algorithm performances under different levels of load.
3. Alternative metrics: Rather than using fixed benchmarks of 100% and 95% for the metrics, it might be more objective to instead measure the highest % achieved and the highest % consistently achieved.

7. ACKNOWLEDGEMENTS

We would like to thank our mentors Chua Wei Ting and Ho Wei Xiong for their invaluable support and guidance throughout the course of this project. We would also like to thank DSO for giving us the opportunity to embark on this learning journey.

8. REFERENCES

[1] Gianni Di Caro and Marco Dorigo. 1998. AntNet: distributed stigmergic control for communications network. Journal of Artificial Intelligence Research 9(1): 317-365.

[2] Gianni Di Caro. 2004. Ant Colony Optimisation and its Applications to Adaptive Routing in Telecommunication Networks (Unpublished doctoral dissertation). Université Libre de Bruxelles, Brussels, Belgium.

[3] Benjamín Barán and Rubén Sosa. 2000. A new approach for AntNet routing. Proceedings Ninth International Conference on Computer Communications and Networks: 303-308.

[4] Kwang Mong Sim and Weng Hong Sun. 2003. Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 33(5): 560-572.

[5] Thomas Stüzle and Holger H. Hoos. 2000. MAX-MIN Ant System. Future Generation Computer Systems 16(9): 889-914.

[6] Michael Lawrence Fredman and Michael E Saks. 1989. The cell probe complexity of dynamic data structures. Proceedings of the twenty-first annual ACM symposium on Theory of computing: 345-354.