Ant Based Routing Algorithms

L.J.M. Rothkrantz, J.C. Wojdeł, A. Wojdeł, H. Knibbe
Delft University of Technology, Knowledge Based Systems,
Zuidplantsoen 4, 2628 BZ Delft, The Netherlands
L.J.M.Rothkrantz@cs.tudelft.nl

Abstract

Recently, agents algorithms based on the natural behavior of ants were successfully applied to route data in telecommunication networks. Mobile agents created the routing tables in the simulated network with behaviors modeled on the trail-laying abilities of agents. This paper describes an adapted version of the Ant Based Control algorithm. To test the algorithms in different network topologies, a simulation environment was designed and implemented.

1 Introduction

In recent years there has been an enormous growth in demand and supply of communication network services. An optimal and flexible use of these services puts high demands on network control. The routing algorithm is the core of network control. Algorithms developed in the field of artificial life can be applied for optimal routing in networks. It proved that the ant-based algorithms were superior to commonly used routing algorithms [2, 3].

In [3] an ant colony algorithm was used for routing and load balancing in a telephony network. An adapted version of that algorithm was used for routing in symmetrical and asymmetrical packet switched networks. Basic ideas from [2] were integrated in that adapted version.

2 Emergent behavior of ants

One of the abilities of a colony of ants is to find the shortest route from their nest to a food source. Ants can lay pheromone trails when going from the nest to the food source, from the food source to the nest or in both directions. When looking for food, the ants follow the pheromone trails with a probability proportional to the strength of the trail. They do not follow the trail with precisely the same probability, but with a certain amount of error (or noise). The strength of the trail sensed by an ant depends on the original strength and the time elapsed since laying the trail, since pheromones diffuse in time. Multiple ants can travel the same route, so the resulting trail can consist of multiple trails laid by different ants at different times. The pheromone trail sensed by ants is therefore a composite one. The probability that an ant chooses a particular route depends on the concentration of the pheromones. In the beginning returning ants give quicker update of the trail along the shortest path and increase the chance that the next ant chooses the same path. In this way ants will bias towards the shortest paths.

3 Ant-Based Control for network management

The idea of emergent behavior of natural ants can be used in telecommunication networks to build routing tables. The network is most naturally represented by an undirected graph. Each node in the graph corresponds to a switching station; the links between nodes correspond to communication channels. Mobile agents, whose behavior is modeled on the trail-laying abilities of ants, replace the ants. Pheromone trails will be replaced by the probability tables at the switching points. The agents move across the network between randomly chosen pairs of nodes; as they move, pheromone is deposited as a function of their distance from their source node and the congestion encountered on their journey. They select their path at each intermediate node according to the distribution of simulated pheromone at each node. Each node in the network has a probability table for every possible final destination. The tables have entries for each next (neighbour) node. The probabilities influence the agents' selection of the next node in their journey to the destination node. The probability of the agents choosing a certain next node is the same as the probability in the table.

The probability tables only contain local information and no global information on the best routes. Each time an agent visits a node, the next step in the route is determined. This process is repeated until the agent reaches its destination. Thus, the entire route from a source node to a destination node is not determined beforehand.

Agents are launched at each node at every time step with a random destination node taken from the list of destination nodes in the probability table. They travel around the network using the probabilities in the probability tables. The probability tables are initially filled with equal values. Agents update the probability tables (see section 4). We can use these probability tables to route packets in the network. Packets have a size different than zero. So in routing the packets we have to take into account the limitation of capacity at the links or at the nodes. Packets can be routed according to the probabilities or according to the highest probability.

4 Agent algorithm

The agent algorithm uses forward and backward agents. The forward agents collect the data and the backward agents update the corresponding probability tables in the associated direction. The algorithm consists of the following steps:

At regular intervals, i.e. each time step, from every network node $\mathbf{s}$, a forward agent is launched with a random destination $\mathbf{d}$: F_{sd}. This agent has a memory that is updated with new information at every node **k** that it visits. The identifier **k** of the visited node and the time elapsed since its launch time to arrive at this node **k** is added to the memory. This results in a list of (**k**, $\mathbf{t}_k$)-pairs in the memory of the agent. Each travelling agent selects the link to the next node using the probabilities in the probability table. Next, nodes that have already been visited by this agent are filtered out in the agent's copy and the new, temporary probability distribution is normalized to 1. Only this agent uses this new, temporary probability distribution. The probability table in this node is not updated. If an agent is forced to return in an already visited node, the agent backtracks to the previous node. The duration of this cycle is subtracted from the trip times calculated after this cycle.

When the destination node **d** is reached, the agent F_{sd} generates a backward agent: B_{ds}. The forward agent transfers all its memory to the backward agent and then it will destroy itself. During the transfer of the memory, all cycles/loops in the agent's path are filtered out. The backward agent travels from the destination node **d** to the source node **s** along the same path as the forward agent, but in the opposite direction. It uses its memory to know the path, so it does not use the probability tables. The backward agent (with previous node **f**) updates the probability table in the current node **k**. The probability P_{df} associated with node **f** and destination node **d** is incremented and the probabilities P_{dx} associated with other next nodes **x** for the same destination node **d** are decreased using the formulas below.

4.1 Updating the probability tables

The probability of the entry corresponding to the node from which the backward agent has just arrived is increased using the following formula:

$$P_{new,i} = \frac{P_{old,i} + \Delta P}{1 + \Delta P} \tag{1}$$

Here, $P_{new,i}$ is the new probability, $P_{old,i}$ the old probability and ΔP the probability increase. Relative to the increasing age of the forward agent ΔP has to decrease. The formula we will use is:

$$\Delta P = \frac{a}{t} + b \tag{2}$$

Where a and b are constants and t is the trip-time of the forward agent at some node. This trip time is the sum of the number of time steps passed since the launch of the agent and the delay at the nodes and the links. The backward agent realizes the updating. We don't take in account that the network conditions can change from the moment that the node is visited by the forward agent and the updating visit of the backward agent.

The other entries on the same row are decreased using the formula:

$$P_{new,j} = \frac{P_{old,j}}{1 + \Delta P}, \forall j \neq i \tag{3}$$

This assumes that the sum of the probabilities remains 1. Probabilities can only decrease if another probability increases. Probabilities can approach zero if other probabilities are increased much more often. This is not desirable, because new, better routes will not be discovered quickly. To solve this problem, an exploration probability is used as a minimum value of each probability. An example is 0.05 divided by the number of next nodes. After setting this minimum, the row of probabilities is normalized to 1. This ensures that none of the entries in the probability table will approach zero.

For a given value of ΔP, the absolute and relative increase of P_{new} is much larger for small values of P_{old} than for large values of P_{old}. This results in a weighted change of probabilities. The formulas were taken from [3], because of their simplicity.

The probability tables are initialized with equal values in such a way that each row sums up to 1. The first agents do not have any information about routes, let alone the quality of the routes. The performance of the network is therefore very bad and cannot be evaluated properly. The quality of the routes found by the agents improves with time. At first, the agents will find many cycles. The number of cycles decreases as the probability tables are filled with information that is more accurate. Further adaptation is caused by network congestion.

4.2 Delaying agents

The routing algorithm should minimize congestion at the nodes and at the links of the network. We assume that the links have a limited capacity. Also the links at a node share the buffer similarly, with a limited capacity. Minimization of congestion can be realized by delaying ants at the parts of the network that are heavily used. As a result the probability tables on the congested nodes will not be updated anymore. So the agents are forced to look for alternative, less congested routes. The delay in time steps that is given to an ant is a decreasing function of the spare capacity s of the nodes or links. Agents on highly congested nodes should have more than proportional bigger delay than agents on lightly used nodes. An exponential function is chosen:

$$delay = c \cdot e^{-ds} \tag{4}$$

Where c and d are constants and s is the spare capacity. Because there can be congestion on a node and on a link too, we take the sum of the delay at the node and the delay at the link.

5 Experiment

To test the algorithm in different network topologies, a simulation environment was designed and implemented. We used two different types of agents (see section 3). The ant-agents, which we will call agents, are used to update the probability tables. They are routed according to the probability tables. We assume that agents have size zero, so agents can always be routed along the links or queued in the buffer at the nodes. At each time step an ant-agent is launched with a random destination. Packets are the second type of agents and have a size different from zero. They are

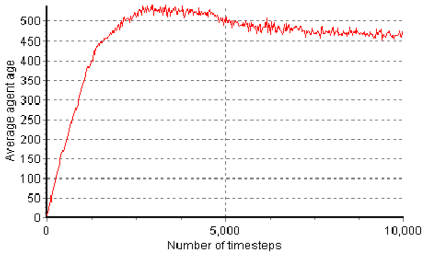


Figure 1: Average agent age

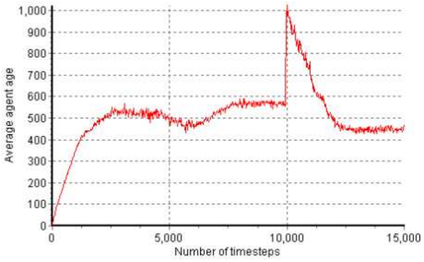


Figure 2: Average agent age in changing network

routed according to the highest probability. But in routing the packets we have to take in account the limitation of capacity at the links or at the nodes. Packets can and will be routed if there is spare capacity on the links. On the arriving node the packet will be buffered or deleted if there is no spare capacity available. The topology of the network in our experiments is modeled after the British Telecom network. It has 20 nodes and 28 links connecting the nodes.

The goal of the agent algorithm is to find optimal probability tables and the agent algorithm is supposed to converge to this optimum. Figure 1 shows the average agent age per time step. At first the average agent age number increases up to a maximum. The probability tables are gradually filled with better information, which leads to faster routing of agents and packets. Faster routes reduce the time agents spend in the network. And thus reduces the total number of agents in the network and the average agent age.

The topology of the network and the traffic can change over time. In an experiment we disabled a node at t=5,000 and enabled again at t=10,000. Figure 2 shows an increase in the average agent age between t=5,000 and t=10,000. Disabling a node also disables the links to or from that node, thus reducing the number of possible routes between nodes in the network. This explains the increase in average agent age. The peak at t=10,000 comes from the fact that there are many agents in the network trying to find a route to the re–enabled node. This creates a temporarily increase in the average agent age.

5.1 Setting the values of the parameters using genetic algorithms

In section 4.1 and section 4.2, we introduced four parameters a, b, c and d in the formulas used for updating the probability tables and delaying the ants. The choice of these parameters isn't arbitrary. In [3] the choice was based on experiments with a simulated network. We developed an additional software module in our simulation system based on genetic algorithms, to find good values for the parameters a, b, c and d. Let us assume that the parameters belong to the interval [0,1]. These numbers can be represented in a binary code using N bits. The selection strategy was randomized with the probability of selection proportional

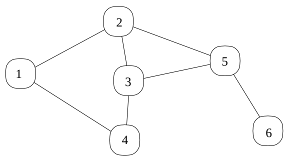


Figure 3: Network to test the stability of the probability tables

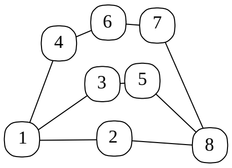


Figure 4: The network to test shortest route recovery

to the fitness. Reproduction was accomplished by crossover. Individuals selected for reproduction are randomly paired. For each pair a crossover point is randomly chosen.

As fitness function we chose the average path-length of the agents in the following way. Given the architecture of the network we can compute the "Ideal Path Length" (IPL), which is the shortest path calculated using a recursive algorithm. Then, for every choice of the parameters, we run the network 10 times. We compute the "Average Path Length" (APL) every time-step. Then we calculate q=IPL/APL. If q is greater than 0.95, we stop running the network and note the time t. If such a value of q is not reached, we stop the simulation after 10000 time-steps. In this way we find "optimal" values of the parameters. In [3] we chose $a = 0.08$ and $b = 0.005$, but using GA we found that $a = 0.02$ and $b = 0.015$ is a better choice of the parameters. We note that the results are heavily dependent of the choice of the fitness function. In one of our experiments we took as fitness function the number of deleted packets. As a result we found that $a = 0.068$ and $b = 0.695$. This means that the network does not take into account the average length of the path (see formula 2 with $a \to 0$).

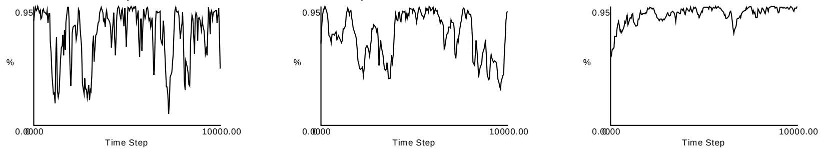
5.2 Stability of the probability tables

One of the interesting questions is how the choice of the probability update parameters influences the stability and flexibility of obtained solutions. We decided to work out two example networks and to test them in a range of parameters a and b.

In the first experiment, the network with 6 nodes was used (see Fig. 3). As the starting point, we chose the values of $a = 0.08$ and $b = 0.005$ Those values were reported in [3] as the optimal ones for one of the networks. When monitoring the changes of the probabilities on the selected nodes, we have found that with such parameters the network is far from being stable. To counteract it, we decreased the values of a and b simultaneously (Figure 5) and independently (Figures 6, 7).

In order to stabilize the solution, the values of a and b must be decreased to very small values in such a small network. Figure 6 and Figure 7 show that while the value of parameter a highly influences the stability, the parameter b can be chosen more freely. The most interesting result that can be seen in Figure 6 is that the highest stability of the network can be obtained with the value of $a = 0$.

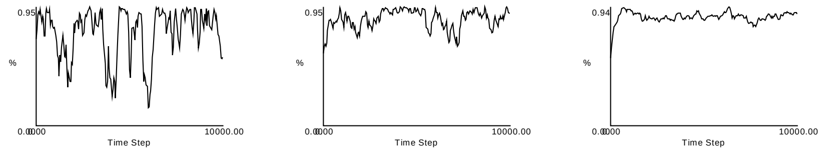
Route 1-6 (optimal choice - 1-2)



$a = 0.08$; $b = 0.005$ $a = 0.02$; $b = 0.00125$ $a = 0.01$; $b = 0.000625$

Figure 5: Stability of probability tables with a and b decreasing

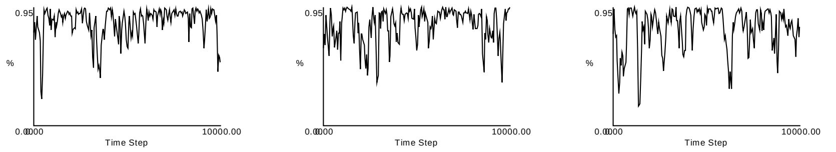
Route 1-6



$a = 0.04$; $b = 0.005$ $a = 0.01$; $b = 0.005$ $a = 0.0$; $b = 0.005$

Figure 6: Stability of probability tables with a decreasing and b constant

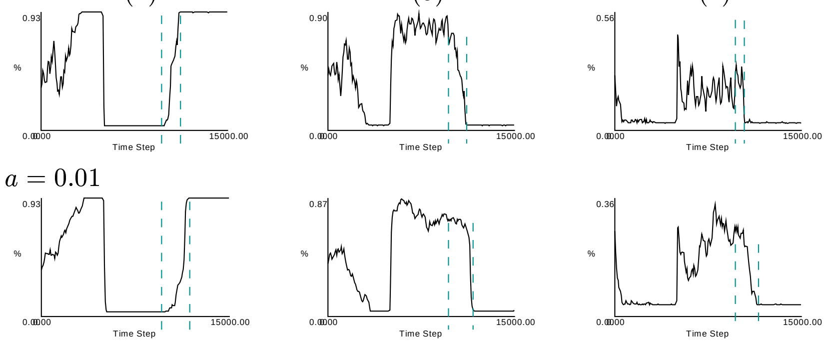
Route 1-6



$a = 0.08$; $b = 0.0025$ $a = 0.08$; $b = 0.000625$ $a = 0.08$; $b = 0.0$

Figure 7: Stability of probability tables with a constant and b decreasing

P(2) P(3) P(4)



$a = 0.00$

Figure 8: Changes of probabilities on the way from node 1 to node 8

Of course one can see also that the time in which the agent behavior stabilized was a little longer than for small but non-zero values of a.

On the other hand we may expect that if a is too small, the behavior of agents will not be flexible enough. To test it, we decided to build another simple network (see Figure 4) in which at some time the optimal route will be disabled by disabling the node number 2 and restoring it later. Node 2 was disabled at time-step 5000 and re-enabled at time-step 10000. The optimal route is with maximum probability P(2) when the node 2 is enabled and with maximum probability P(3) when it is disabled.

We tested the network with two sets of parameters as it is described in Figure 8. We checked the probabilities of choosing the nodes 2, 3 and 4 on the way from node 1 to 8. Although the results show once more that the solution found with the value $a = 0$ is much more stable, the "recovery time" after enabling the node 2 at 10000 time-steps is noticeably longer.

The conclusion that can be derived from the above observations is that there is probably no optimal set of parameters. Very small values of parameter a give stable solutions that steadily converge to the optimum, while higher values give the network higher flexibility and shorter response times in changing situations. This means that one should pay more attention to the definition of how well the agents behave; simple measurement of the average path length is not enough.

6 Conclusions

The ABC algorithm is used in some applications and tested by many researchers [1]. It proved to be a very robust and efficient algorithm for routing in packet switched networks. Nevertheless, there are still a lot of unsolved problems. In this paper, we describe some results again based on simulation studies. A mathematical foundation is still lacking. One of the problems is that the results are very dependent of the way we implemented the algorithms, the architecture of the network and the settings of important parameters and last but not least, the network load. We have to perform some simulation studies based on real data and a simulation of an existing network.

References

[1] Bonabeau, E, M. Dorigo, G. Theraulaz. Swarm Intelligence. Oxford University Press, 1999.

[2] Di Caro, G., M. Dorigo, AntNet: Distributed Stigmergetic Control for Communications Networks, Journal of Artificial Intelligence Research (JAIR), Vol 9, 2000, pp. 317–365.

[3] Schoonderwoerd, R., O. Holland, J. Bruten, L.J.M. Rothkrantz, Load Balancing in Telecommunication Networks, Adaptive Behaviour, 5 (2), 1997, pp. 169–207.