A DISTRIBUTED APPROACH TO ANT COLONY OPTIMIZATION

Eng. Sorin Ilie[1] Ph. D Student
University of Craiova
Software Engineering Department
Craiova, Romania
Prof. Costin Bădică Ph. D
University of Craiova
Software Engineering Department
Craiova, Romania

Abstract: : Swarm Intelligence(SI) is the emergent collective intelligence of groups of simple agents. Economy is an example of SI. Simulating an economy using Ant Colony algorithms would allow prediction and control of fluctuations in the complex emergent behavior of the simulated system. Such a simulation is far beyond SI's capabilities, which is still in its infancy. This paper presents a distributed approach implementing Ant Colony Optimization(ACO). We present our agent based architecture of ACO and initial experimental results on the Travelling Salesman Problem. The innovation of our work consists of: i)representing network nodes as software agents, ii) representing software agents as software objects that are passed as messages between the nodes according to ACO rules.

JEL classification: Y90, C61

Key words: Swarm Intelligence, Ant Colony Optimization, Multi-Agent, Distributed, Heuristis.

1. INTRODUCTION

Swarm Intelligence(SI) is the emergent collective intelligence of groups of simple agents. The economy is the emergent behavior of a population of individuals acting on local knowledge, therefore it is an example of SI. Nature is a great source of inspiration for SI as it provides us with the successful models of ant colonies, termite colonies, bee colonies, particle swarms and others. Simulating an economy using nature inspired algorithms would allow prediction and control of fluctuations in the complex emergent behavior of the simulated system. This is not yet possible however efforts are being made to understand and implement such systems [1].

Natural phenomena are inherently distributed, so we would expect distributed computing to have a lot of potential for the application of nature-inspired computing. In this context, we think that nature-inspired computing should allow a straightforward mapping onto existing distributed architectures. Therefore, to take advantage of the full potential of nature inspired computational approaches, we have setup the goal of

[1] **This work was partially supported by the strategic grant POSDRU/88/1.5/S/50783, Project ID50783 (2009), co-financed by the European Social Fund -- Investing in People, within the Sectoral Operational Programme Human Resources Development 2007 - 2013.**

investigating new distributed forms of Ant Colony Optimization (ACO hereafter) using state-of-the-art multi-agent approaches.

We propose a multi-agent system architecture that allows the implementation of ACO in a parallel, asynchronous and decentralized environment. The novelty of our approach consists in: i) designing and implementing the computing system as a network of intelligent software agents ([2] [5]) that represent the problem environment, i.e. the nodes of the graph in the context of TSP; ii) reduction of ants management to messages exchanged between the agents.

Existing sequential implementations of ACO [1] are highly synchronous and require global knowledge. These are obstacles in implementing a distributed version.

In [4] the authors claim a parallel, distributed, asynchronous and decentralized implementation of ACO but their approach requires the centralized collection of the best tours known and pheromone update every time a better solution is found. The authors do not present any experimental data on their approach but claim that the parallelization and asynchronicity has no effect on the accuracy, speed and reliability of the algorithm.

2. BACKGROUND

ACO is inspired by the behavior of real ants. When ants find food, they secrete pheromone on their way back to the anthill. Other members of the colony sense the pheromone and become attracted by marked paths; the more pheromone is deposited on a path, the more attractive that path becomes to ants. The pheromone is volatile so it disappears over time. Evaporation erases longer paths because it takes ants longer traverse them (see figure 1 and 2). Paths that are not of interest anymore will no longer be marked with pheromone. However, shorter paths are more quickly refreshed, thus having the chance of being more frequently explored. Intuitively, ants will converge towards the most efficient path because that path gets the strongest concentration of pheromone(see figure 2). Artificial ants are programmed to mimic the behavior of real ants while searching for food [1].

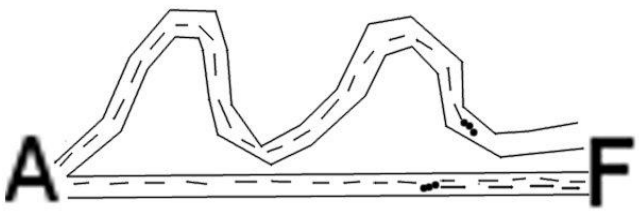

Figure no 1. Ants that traveling on shorter paths can mark them with pheromone faster because it takes less time for them to return from the food F to the anthill A

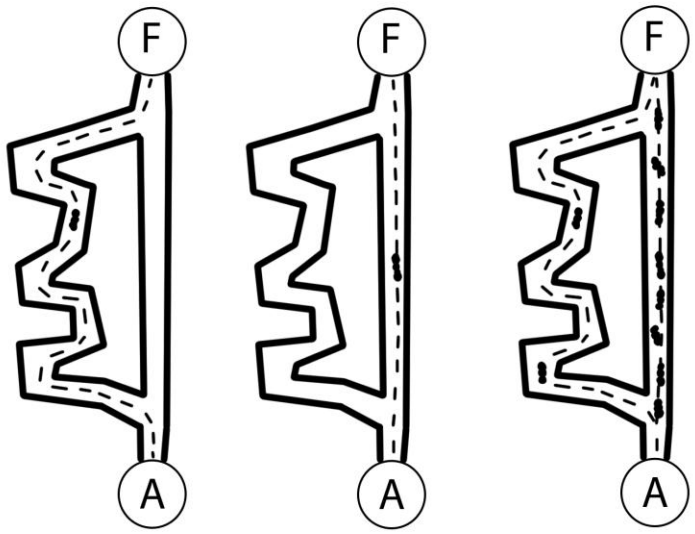

Figure no 2. Over time evaporation will erase the longer paths as they cannot be marked with pheromone as fast as the short ones

In this paper we propose a distributed approach to ACO and show how it can be applied for solving TSP. The goal of TSP is to compute a shortest tour that visits each node in a weighted graph exactly once. The decision version of TSP is known to be NP-complete which basically means that it is very unlikely that a polynomial solution for solving TSP exists. So TSP is a very good candidate for the application of heuristic approaches, including ACO.

The main idea behind our approach is to provide a distributed architecture for modeling the problem environment. Artificial ants originating from the anthills that are located in the environment will travel to find optimal solutions, following ACO rules. In order to use ACO to solve TSP, the problem environment is conceptualized as a distributed set of interconnected graph nodes. Additionally, each graph node is also an anthill. Ants travel between nodes until they complete a tour. Once they return to their originating anthill, they mark the solution with pheromone by retracing their path.

Our model is based on the ACS algorithm presented in [1], which is a sequential implementation of ACO in which it is preferred that the ants move in parallel (according to [1]), with a few differences due to the restrictions imposed by our distributed architecture ant the lack of global knowledge. These differences are presented in section 5.

3. MATHEMATICAL MODEL

ACO rules determine the amount of pheromone deposited on edges, the edge chosen by each ant in each node on their way, and how fast the pheromone deposited on each edge should evaporate. For this purpose we use the following mathematical model.

The function that determines the most lucrative hop is the same as for other ACO algorithms. An ant located in node i will choose to move to node j with the probability pi,j computed as follows:

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \quad (1)$$

where:

$\tau_{i,j}$ is the amount of pheromone deposited on edge (i,j)
α is a parameter to control the influence of $\tau_{i,j}$
$\eta_{i,j}$ is the desirability of edge (i,j) computed as the inverse of the weight of edge (i,j), i.e. $w_{i,j}$
β is a parameter to control the influence of $\eta_{i,j}$
j represents a node reachable from node i that was not visited yet

Better solutions need to be marked with more pheromone. So whenever an ant k determines a new tour the ant will increase pheromone strength on each edge of the tour with a value that is inversely proportional to the cost of the tour. Moreover, when an ant improves the cost of its currently best found tour, a bonus is added to the pheromone strength (see equation 2).

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k \; if\_ant\_k\_travels\_on\_edge(i,j) \\ 0 \qquad\qquad otherwise \end{cases} \quad (2)$$

where:
Lk is the cost of the k-th ant's tour.
$\Delta\tau_{i,j}^{k}$ is the amount of pheromone ant k deposits on edge (i , j)

When an ant travels along a given path, this traveling takes an amount of time that is proportional with the travel distance (assuming the ants move with constant speed). As pheromone is volatile, when an ant travels more, pheromone will have more time to evaporate, thus favoring better solutions to be discovered in the future. We can conclude that adding pheromone evaporation to our model can be useful, especially for solving a complex problem like TSP.

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \rho\tau_{i,j}^{k} \quad (3)$$

where:
$\tau_{i,j}$ is the amount of pheromone on edge (i,j)
$\tau_{i,j}^{k}$ is the amount of pheromone ant k deposits on edge (i , j)
ρ is the evaporation rate $0 \leq \rho < 1$

All ants use formula (1) to determine the probability of their next step. Therefore they will often choose the edge with the highest probability, this means the exploration of less probable edges is low. The solution is to decrease the pheromone on edges chosen by ants. This has the effect of making them less desirable, increasing the exploration of the edges that have not been picked yet. Whenever an ant traverses an edge it updates its pheromone deposit using the formula:

$$\tau_{i,j} = (1-\zeta)\tau_{i,j} + \zeta\tau_0 \quad (4)$$

where:

ζ is the evaporation rate $0 \leq \zeta < 1$

τ_0 is the initial amout of pheromone on each edge

A good heuristic to initialize the pheromone trails is to set them to a value slightly higher than the expected amount of pheromone deposited by the ants in one tour; a rough estimate of this value can be obtained by setting, $\tau 0 = 1/(nC)$, where n is the number of nodes, and C is the length of a tour generated by a reasonable tour approximation procedure, for example C=nwavg where wavg is the average edge cost.

Real ants try not to stray too far from the anthill unless they have to. In order to simulate this instinct we introduce the time to live (TTL hereafter) attribute: if the path cost exceeds this value, the ant will return to its anthill. If no solution is found given the current TTL then the value of this attribute is increased, thus giving ants the chance to travel more during their next round. Note that, as our ants have the TTL attribute and they age by decrementing TTL with the weight of every edge on their current path, the value of Lk representing the cost of a solution will be immediately available for updating pheromone with formula 3.

4. ARCHITECTURE

In our Architecture the nodes of the graph are conceptualized and implemented as software agents [5]. For the purpose of this work, by software agent we understand a software entity that: (i) has its own thread of control and can decide autonomously if and when to perform a given action; (ii) communicates with other agents by asynchronous message passing. Each agent is referenced using its name, also known as agent id.

The activity carried out by a given agent is represented as a set of behaviors. A behavior is defined as a sequence of primitive actions. Behaviors are executed in parallel using interleaving of actions on the agent's thread with the help of a non-preemptive scheduler, internal to the agent [2].

Node design must include behaviors for sending and receiving ants and for pheromone evaporation. Whenever an ant is received, the receiveAnt() behavior immediately prepares it and then sends it out to a neighbor node following ACO rules.

Ants are represented as objects with the following set of attributes: TTL, pheromone strength, returning flag, goal reached flag, best tour cost and a list of agent ids representing the path that the ant followed to reach its current location. The list is necessary for two reasons: i) the ant needs to retrace its steps in order to mark the tour with pheromone and ii) we need to avoid loops so only unvisited nodes, the nodes that are not yet in the list, are taken into account as possible next hops. Attributes are initialized when an ant is created and updated during the process of ant migration to reflect the current knowledge of the ant about the explored environment. For example, whenever an ant reaches a destination node (i.e. its anthill), the ant saves its currently best tour onto this node. So, whenever another ant from the colony travel through this node, it senses the environment and eventually updates its ``knowledge'' about the value of the currently best tour. So ants have the possibility to exchange through the environment (i.e. the set of graph nodes) not only pheromone information, but also information about the best solutions found so far.

Nodes have the following parameters: initial TTL, goal reached flag, TTL increment, ρ, best tour cost and a list of neighbors with their respective edge weight and deposited pheromone. The first two parameters are used to create the initial population of ants. The goal reached flag indicates if the ants have succeeded in completing at least one tour. All nodes remember the best tour by reading the ant's "best tour cost" attribute and update the ants accordingly.

In our approach nodes create the ant population and update returning ants using tweak() method. If all the ants belonging to this node have exhausted their TTL before finishing a tour the initial TTL is increased. Nodes also calculate pheromone strength according to the path length cost (see equation 3), set the goal reached flag for successful ants, exchange ant information, deposit pheromone when needed, decrease ant TTL and sends ants with the TTL<=0 to their anthill.

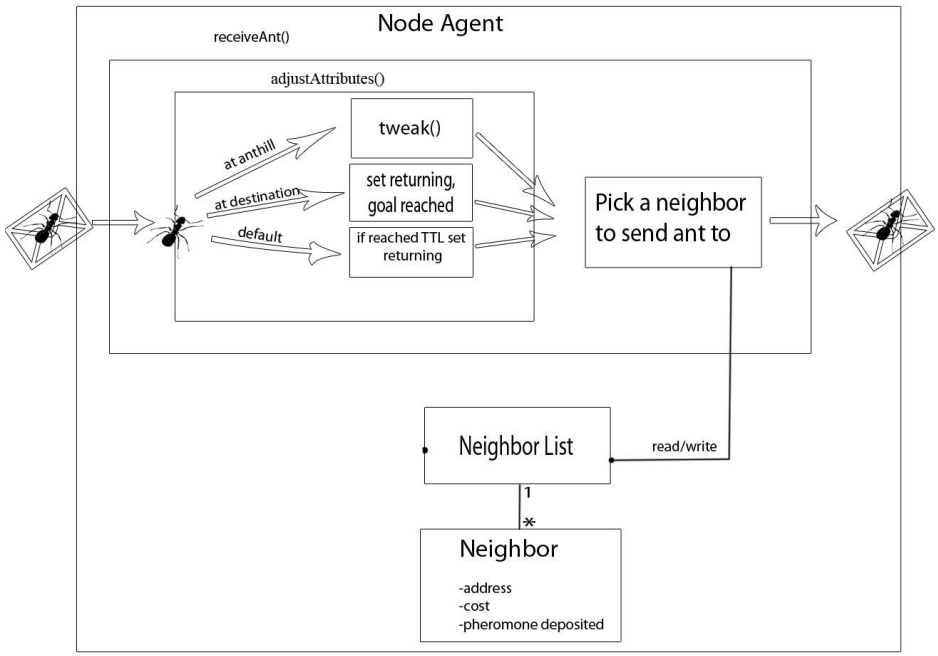

Figure no 3. The structure of a Node Agent

The structure of a node is presented in figure 3. receiveAnt() (see algorithm 1) behavior parses an ant message, adjusts ant's attributes using adjustAttributes() method and sends it out to the address determined by pickBestNeighbor() method. This happens whenever the Jade message queue isn't empty. adjustAttributes() (see algorithm 2) method sets the goalReached flag and calculates pheromone strength using equation (2) whenever an ant has completed a tour. tweak() method is invoked either if the ant has returned to the anthill after marking its tour or if it has exhausted its TTL (i.e. the ant died). This method (see algorithm 3) is used to update ants that have returned to the anthill after exhausting their TTL or have set the goalReached flag.

The pickBestNeighbor() method (see algorithm 4) uses equation (1) to determine the address of the node where to send the ant. When the ant returns to the anthill, this method sends the ant to the first node from its list of visited nodes, popping

it from the list, and deposits the ant's pheromone. This method also implements evaporation using equations (3) and (4).

```
Ant ant=new Ant(receive().getContent());
 adjustAttributes(ant);
 sendTo(pickBestNeighbor(ant),ant);
```

Algorithm no 1. Behavior receiveAnt

```
if (ant.getReturning() AND ant.atAnthill())
        tweak(ant);
    else{
        if (ant.getReturning()=false AND ant.atAnthill()){
             ant.setGoalReached();
             ant.calculatePheromoneStrength();
             updateBestTourRecord();
             }

        if((ant.getGoalReached() OR (ant.getTTL()<=0)))
             ant.setReturning();
    }
```

Algorithm no 2. Method adjustAttributes

```
if (goalReached=false AND ant.getGoalReached()=false)
          initialTTL+=TTLIncrement;
     if(ant.getGoalReached()) goalReached=true;
     a.refreshAnt();
```

Algorithm no 3. Method tweak

```
   if(a.getReturning()){
       if(a.getGoalReached())
          depositPheromone();//formula (3)
       return a.getLastTrack();
   }
   bestNeighbor=pobabilisticRandomChoice();//formula(1)
   if (a.getTTL() > 0)
        a.growOlder(weight[bestNeighbor]);
    else return a.getAnthill();
   updatePheromone();//formula (4)
   a.pushTracks(bestNeighbor);
   return bestNeighbor;
```

Algorithm 4. Method bestNeighbor

We tested out approach on the TSP map eil51 from TSPLIB[3]. This map has the following characteristics: 51 nodes, maximum edge weight 86, minimum edge weight 2, average edge weight 32.39 , official optimum tour 426.

An important issue in our experiments was how to detect experiment termination. This is not simple for at least two reasons: (i) as we are in a distributed setting, information about the found tours is spread onto the network of agents and minimum cost tours are continuously updated while ants discover new better tours; (ii) TSP is a complex computational problem and in order to tackle this complexity, our approach is inherently heuristic. Therefore even when ants are not able to improve the currently best tour we cannot be sure that a better tour does not exist. Experimentally we observed that when the currently best tour is not updated for a given quiescence time Tq it is safe to assume that the ants have settled on a solution and the experiment can be stopped.

The ACO parameters were set to the recommended values in [1] for the ACS algorithm on which our approach is based. The number of ants is equal to the number of nodes n= 51, $\tau_0=1/(n2w_{avg})$,$\zeta=\rho=0.1$, $\alpha=1$, $\beta=5$, the TTL is our own contribution and we set it at the value TTL= nw_{avg}. We chose T_q =30s experimentally, its value must large enough for all the ants to complete a tour which is dependent on network and computational speed.

We ran the algorithm 20 times independently and collected the average data. The experiments were carried out on a network of computers with dual core processors at 2.5 GHz and 1GB of RAM memory. These workstations were connected in a Myrinet Network for MPI low latency communication. When running the algorithm on a single computer we obtained the average best tour 459.4 in 17 seconds. When we ran the same experiment on two computers with 25 and 26 node agents respectively we obtained an average best tour of 458.6 in 10 seconds. These results are encouraging as they show that the algorithm is scalable, however further tests are required for a comprehensive analysis. Our next step is to thoroughly test the architecture to conclude if the execution time will decrease sufficiently to make this a viable parallel, asynchronous and distributed approach to ACO.

The found solutions are inferior to the solutions found by the sequential algorithm due to the restrictions imposed by the distributed approach however this architecture offers a suite of possible improvements that could outweigh the disadvantages in the future. For example in Jade behaviors[2] can be triggered on a timer (ticker behaviors). These can be used to implement an evaporation dependent on time and edge weight without the need of ant presence as in the current implementations.

4. RELATED WORK

In [4] a similar approach is presented, where both nodes and ants are implemented as agents. In their approach the ant agents visit a node by requesting, through an ACL message [2], the list of possible next hops along with their costs, pheromone deposits and the node's best tour cost. The ant then has to notify the node about the next hop it decided to make in order for the node to be able to update its pheromone level. We avoid the three messages needed to move an ant, by sending the nodes all the information necessary to make all the necessary decisions on its own. This information is contained in an ant message received directly from another node. In [4] when an ant completes a tour it compares it's cost with the collected best tours from the

nodes. A global best tour update is triggered if a better tour has been found. Such global, centralized synchronization of the best tour should be avoided in a distributed system by introducing a pheromone bonus for ants that find a better tour, this will be the object of our future work.

Our implementation is based on the ACS algorithm presented in [1] with three differences: i) we do not have iterations as it would be time consuming to find out when all ants have found a tour, plus this would be a synchronization defeating the purpose of this being a distributed architecture ii) as a consequence of the first difference, we have no way to implement a global evaporation at each iteration iii) we do not compare the best tours after each iteration allowing only the best ant to mark the tour, instead we allow every ant to mark its tour with a pheromone quantity inversely proportional to the tour cost. These differences are due to the restrictions imposed by a distributed architecture. The result of these restrictions is a decrease the efficiency of the algorithm in finding the shortest tour, however, the distributed agent based architecture offers other advantages such as scalability or the ability of implementing real life evaporation that depends on time and edge cost, not on ant presence.

The multiple travelling salesmen problem presumes that salesmen starting in various nodes are trying to find the shortest tour that visits a number of l nodes. This is a generalization of our approach where we chose l to be n, the number of nodes. In [6] ACO based algorithm that solves this problem is presented but it is sequential. This algorithm updates the pheromone trails after a number of m solutions have been determined. The ants are allowed to make a number of moves then the next ant is allowed to move but only in the unvisited nodes by the first ant. This is global knowledge. For the algorithm to be distributed, ants can only interact via the environment. Instead of global knowledge, we use equation (4) to favor exploration of unvisited nodes.

In [7] an algorithm is described that reduces the number of nodes that are to be taken into consideration when trying to solve TSP with ACO. ACO is applied repeatedly on the set of nodes determined by the algorithm adjusting the candidate node list each time. This approach requires absolute knowledge of the environment which is not the case in a distributed architecture such as ours. In our approach the environment is not data but the algorithm itself, as each node in the map is actually an "intelligent" agent that manages ants in the form of messages.

REFERENCES

1. Dorigo, M. and Stutzle, T. Ant Colony Optimization. MIT Press, 2004
2. Bellifemine, F.L., Caire,G. and Greenwood, D. Developing Multi-Agent Systems with JADE. John Wiley & Sons Ltd, 2007.
3. Reinelt G. a traveling salesman library. ORSA Journal on Computing. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.
4. Ridge, E., Kudenko,D., and Kazakov,D. Parallel, asynchronous and decentralised ant colony system. In In Proc.of the First International Symposium on Nature-Inspired Systems for Parallel, Asynchronous and Decentralised Environments (NIS-PADE), 2006.
5. Wooldridge, M. An Introduction to MultiAgent Systems. John Wiley & Sons Ltd, 2002.

6. Junjie, Pan and Dingwei, Wang An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem. ICICIC '06: Proceedings of the First International Conference on Innovative Computing, Information and Control, pages 210--213, isbn 0-7695-2616-0, 2006
7. Li,Lijie, Ju, Shangyou and Zhang, Ying Improved Ant Colony Optimization for the Traveling Salesman Problem. Proceedings of the 2008 International Conference on Intelligent Computation Technology and Automation - Volume 01. Pages: 76-80. 2008