ISBN 978-83-60810-27-9
ISSN 1896-7094

Effectiveness of Solving Traveling Salesman Problem Using Ant Colony Optimization on Distributed Multi-Agent Middleware

Sorin Ilie and Costin Bădică
University of Craiova, Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
Email: sorin.ilie@software.ucv.ro,costin.badica@software.ucv.ro

Abstract—**Recently we have setup the goal of investigating new truly distributed forms of Ant Colony Optimization. We proposed a new distributed approach for Ant Colony Optimization (ACO) algorithms called Ant Colony Optimization on a Distributed Architecture (ACODA). ACODA was designed to allow efficient implementation of ACO algorithms on state-of-the art distributed multi-agent middleware. In this paper we present experimental results that support the feasibility of ACODA by considering a distributed version of the Ant Colony System (ACS). In particular we show the effectiveness of this approach for solving Traveling Salesperson Problem by comparing experimental results of ACODA versions of distributed ACS with distributed random searches on a high-speed cluster network.**

I. Introduction

As NATURAL phenomena are inherently distributed, 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 investigating new distributed forms of Ant Colony Optimization (ACO hereafter) using state-of-the-art multi-agent technology.

In our recent work [1] we proposed a scalable multi-agent system architecture called ACODA (Ant Colony Optimization on a Distributed Architecture) that allows the implementation of ACO in a parallel, asynchronous and decentralized environment. The novelty of our approach is: (i) the problem environment is conceptualized and implemented as a distributed multi-agent system ([2], [3]) and (ii) ant management is reduced to messages exchanged asynchronously between the agents of the problem environment.

Existing computational approaches of ACO [4] are based on sequential algorithms, are highly synchronous and require use of global knowledge. While few parallel and distributed versions of ACO exist [5], they are mainly based on their sequential counterparts, thus hindering the potential gain through parallelization. For example, in [5] the authors propose a parallel, distributed, asynchronous and decentralized implementation of ACO. However, their approach requires maintenance of the globally best solution currently known using global update each time a better solution is found. Moreover, the authors' claim that "their implementation does not effect the accuracy, speed and reliability of the algorithm" is not supported by any experimental evidence.

The focus of this paper is to experimentally evaluate the feasibility of ACODA by considering a distributed version of ACO inspired by the Ant Colony System (ACS) [4]. In particular, we show the effectiveness of this approach at solving Traveling Salesperson Problem (TSP hereafter) by comparing experimental results of ACODA versions of distributed ACS with distributed random searches on a high-speed cluster network. The results clearly show that our distributed version of ACS based on ACODA preserves the nice heuristic properties of standard ACS, while also providing scalability by exploiting distributed computing architectures – multi-agent middleware in this case.

The paper is organized as follows. In Section II we present some background on ACO and distributed approach and we briefly review the ACS model that inspired our initial experiments. There are however notable differences between classic ACS and our distributed version based on ACODA (see Section V). In Section III we introduce ACODA architecture and underlying search algorithm. Section IV presents experimental results that support the effectiveness of our approach by comparing results obtained with running on ACODA our distributed version of ACS with other three distributed search methods. Section V presents related works, while Section VI presents our conclusions and points to future works.

II. Background

ACO is inspired by behavior of real ants. When ants are searching for food, they secrete pheromone on their way back to the anthill. Other colony members sense the pheromone and become attracted by marked paths; the more pheromone is deposited on a path, the more attractive that path becomes. The pheromone is volatile so it disappears over time. Evaporation erases pheromone on longer paths as well as on paths that are not of interest anymore. 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, as that path gets the strongest concentration of pheromone. Artificial ants are programmed to mimic the behavior of real ants while searching for food. More details on the ACO metaphor can be found in [4].

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

The main and also new idea behind ACODA is to provide a multi-agent distributed architecture for modeling the problem environment. One or more anthills are located in this environment. Artificial ants originating from anthills will travel in the environment to find optimal solutions, following ACO rules. In order to approach TSP using ACO, the problem environment is modeled as a distributed set of interconnected graph nodes that are also anthills. Each graph node is modeled as a software agent that can host a population of ants. The 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. The ants traveling is modeled as messages exchanged by the agents that represent the graph nodes.

Many ACO algorithms have been proposed in the literature. A good survey is [4]. While with ACODA we aim at proposing a general distributed framework based on agent middleware for different ACO algorithms, the ACO model considered in this paper is based on a particular version of ACO – the ACS system [4]. ACS is a sequential implementation of ACO that chooses to move ants in parallel instead of moving each ant until it finishes its tour. Our approach presents a few differences due to the ACODA requirements: (i) distributed architecture based on asynchronous message passing and (ii) avoid to use global knowledge.

ACO rules determine the amount of pheromone deposited on edges, the edge chosen by each ant on its way, and how fast the pheromone deposited on each edge evaporates. For this purpose we use the mathematical model of ACO that is used in ACS.

In ACS, ant k located at node i decides to move to node j using "pseudo random proportional rule" (1). Equation (1) chooses to directs the ant either to a completely random node or to a node of high desirability, and this decision is taken probabilistically.

$$j = \begin{cases} argmax_{l \in N_i}((\tau_{i,l})^{\alpha}(\eta_{i,l})^{\beta}), & \text{if } q \leq q_0 \\ J, & \text{otherwise} \end{cases} \tag{1}$$

where:

- α is a parameter to control the influence of $\tau_{i,j}$
- $\tau_{i,j}$ is the amount of pheromone deposited on edge (i,j)
- $\eta_{i,j}$ is the desirability of edge (i,j) computed as the inverse of the edge weight, i.e. $1/w_{i,j}$
- β is a parameter to control the influence of $\eta_{i,j}$
- q is a random variable uniformly distributed in $[0,1]$
- q_0 such that $0 \leq q_0 \leq 1$ is a parameter that controls the selection between a random neighbor and most promising neighbor based on pheromone deposit and edge desirability
- J is a random node selected according to the probability distribution given by equation (2)
- N_i represents the set of neighbors of node i

An ant located in node i will randomly choose to move to node j with the probability $p_{i,j}$ computed as follows:

$$p_{i,j} = \frac{(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}}{\Sigma_j(\tau_{i,j})^{\alpha}(\eta_{i,j})^{\beta}} \tag{2}$$

where:

- α is a parameter to control the influence of $\tau_{i,j}$
- β is a parameter to control the influence of $\eta_{i,j}$
- j is a node reachable from node i that was not visited yet

Following equation (1), the ant makes the best possible move (as indicated by the learned pheromone trails and the heuristic information, i.e. the ant is exploiting the learned knowledge) with probability q_0, while it performs a biased exploration of the arcs with probability $(1-q_0)$.

Better solutions need to be marked with more pheromone. So whenever an ant k determines a new tour V_k of cost L_k 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.

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/L_k & \text{if edge } (i,j) \text{ belongs to found tour } V_k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

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, if a real ant travels more, pheromone will have more time to evaporate, thus favoring better solutions to be discovered in the future. We conclude that adding pheromone evaporation to our model can be useful, especially for solving a complex problem like TSP.

When an ant completes a tour it will retrace its steps marking the edges on the way with pheromone. The update will also take into account pheromone evaporation. Assuming an evaporation rate $0 \leq \rho < 1$, evaporation and pheromone update are implemented in ACS as follows:

$$\tau_{i,j} = (1-\rho)\tau_{i,j} + \rho\Delta\tau_{i,j}^{k} \tag{4}$$

Ants use equation (1) to probabilistically determine their next step. Therefore they will often choose the edge with the highest pheromone, while the exploration of less probable edges is low. This behavior can be compensated by decreasing the pheromone on edges chosen by ants using a local pheromone evaporation process. This has the effect of making them less desirable, increasing the exploration of the edges that have not been picked yet. Assuming that $0 \leq \xi < 1$ is the local evaporation rate and τ_0 is the initial amount of

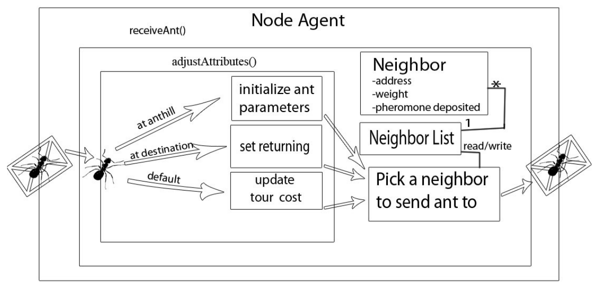

Fig. 1: Node structure in ACODA.

pheromone on each edge, whenever an ant traverses an edge it applies local evaporation by updating pheromone as follows:

$$\tau_{i,j} = (1-\xi)\tau_{i,j} + \xi\tau_0 \tag{5}$$

A good heuristics to initialize 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 tour cost generated by a reasonable tour approximation procedure [4]. For example we can set $C = nw_{avg}$ where w_{avg} is the average edge cost.

In order to observe the impact that evaporation has on the solutions, we have also considered a pheromone update scheme that does not include evaporation at all. In this case equations (5) and (4) are replaced with:

$$\tau_{i,j} = \tau_{i,j} + \Delta\tau^k_{i,j} \tag{6}$$

III. Architecture

In ACODA, the nodes of the graph are conceptualized and implemented as software agents [2]. 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 [3].

Node design (see Figure 1) must include a behavior for sending and receiving ants. Whenever an ant is received, the RECEIVE-ANT() behavior (see Table I) immediately prepares it and then sends it out to a neighbor node following ACO rules.

Ants are represented as objects with a set of attributes: cost of the currently search path (which becomes L_k when the ant completes a tour), pheromone strength (value of $\Delta\tau^k_{i,j}$), returning flag, best tour cost (value of the currently best tour that the ant knows, based on its search history) and a list of node 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 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.

Nodes (represented as software agents) manage a list of neighbor nodes and best tour cost (the value of the currently best tour that the node knows, based on ants that traveled through this node). For each neighbor node we record the weight and the value of deposited pheromone of the corresponding edge. Note that each node and each ant maintain their own values of the best tours they encountered so far. So, whenever an ant is traveling through a node, the ant and the node are able to exchange and update their best tour information accordingly.

In our approach each node creates its own ant population thus becoming an anthill. Nodes calculate pheromone strength according to the tour cost (see equation (3)) and also update the ants' pheromone strength attribute. Additionally, nodes set the returning flag for returning ants, exchange ant information with other nodes, deposit pheromone when needed, and update the cost of the currently search path of an ant.

The structure of a node is presented in Figure 1. RECEIVE-ANT() behavior parses a received ant message, adjusts ant's attributes using ADJUST-ATTRIBUTES() method and sends it out to the address determined by BEST-NEIGHBOR() method. This happens whenever the agent's message queue isn't empty [3]. ADJUST-ATTRIBUTES() method sets returning flag and calculates pheromone strength using equation (3) whenever an ant has completed a tour. Ants that have returned to the anthill are re-initialized.

BEST-NEIGHBOR() uses the RANDOM-CHOICE() method determine the address of the node where to send the ant. When the ant returns to the anthill, the ant is sent to the first node from its list of visited nodes, popping it from the list, and the method DEPOSIT-PHEROMONE() is then called. LOCAL-EVAPORATE-PHEROMONE() together with DEPOSIT-PHEROMONE() implement pheromone update (deposits and evaporation). In order to implement different forms of ACO it is usually sufficient to modify the methods LOCAL-EVAPORATE-PHEROMONE(), DEPOSIT-PHEROMONE() and RANDOM-CHOICE() (see Section IV).

IV. Experiments and Discussions

In order to facilitate experimentation with ACODA, we created a distributed platform based on JADE framework [3] that can be configured to run on a computer network. The focus of the experiments was to show the effectiveness of ACODA to support distributed ACO-based algorithms for solving TSP. We configured ACODA platform on a high-speed cluster network of 7 computers and then we analyzed experimental results that we obtained by running (i) our distributed ACS-based algorithm and (ii) other distributed random search algorithms.

Setup. An experiment is structured as a fixed number of independent experimental rounds. A round consists of one

TABLE I: Algorithms for processing ant information.

```
RECEIVE-ANT()
1. RECEIVE($ant$)
2. ADJUST-ATTRIBUTES($ant$)
3. SEND-TO($ant$,BEST-NEIGHBOR($ant$))

ADJUST-ATTRIBUTES($ant$)
1. if AT-ANTHILL($ant$) then
2.   if RETURNING($ant$) then
3.     INITIALIZE($ant$)
4.   else
5.       SET-RETURN-FLAG($ant$)
         $\triangleright$ calculate ant pheromone using equation 3.
6.       CALCULATE-ANT-PHEROMONE-STRENGTH($ant$)
7. UPDATE-BEST-TOUR()

BEST-NEIGHBOR($ant$)
1. if RETURNING($ant$) then
2.   DEPOSIT-PHEROMONE()
3.   return LAST-VISITED-NODE($ant$)
4. $bestNeighbor \leftarrow$ RANDOM-CHOICE()
5. UPDATE-CURRENT-PATH-COST($bestNeighbor$)
6. LOCAL-EVAPORATE-PHEROMONE()
7. ADD-TO-VISITED-LIST($bestNeighbor$,$ant$)
8. return $bestNeighbor$
```

execution of an algorithm on ACODA, for a given set of parameters. All parameters are initialized at the start of each round. Experimental data are collected during each round. At the end of the experiment (when the fixed number of rounds is reached) these data are used to calculate performance measures.

ACODA is a distributed platform. Setting-up and running it on several computers assumes two stages: (i) initialization; and (ii) execution.

During the initialization stage: JADE platform is started, a number of containers are created (typically one container for each available machine), and finally node agents are created and evenly distributed on available containers of the JADE platform.

The experiment is ran during the execution stage. Experiment execution is controlled using special control messages. Control messages are distinguished from ant exchange messages using their conversation ID. Command messages have their conversation ID set to "command", while ant exchange messages have their conversation ID set to "ant". Command messages are given higher priority than ant exchanging messages.

An experimental round starts when all node agents receive a "start" command, and ends when all node agents receive a "stop" command. A designated node agent – called *MasterNode* is responsible with issuing commands and calculating performance measures. *MasterNode* counts the total number of ants that it receives. When this number reaches a given maximum value M, *MasterNode* stops the current round (issuing a "stop" command) and starts a new round (issuing a "start" command). Note that *MasterNode* is needed only for evaluation and consequently it is not part of the distributed approach.

Duration T_i of a round i is recorded by *MasterNode* agent as time elapsed between issuing a "start" and "stop" command. While the distributed ACO algorithm is running, minimum tour costs are passed from node to node via ants. Whenever a node agent updates its current value of the minimum tour, the time elapsed since the node received the "start" command is also recorded. When an experimental round is finished, *MasterNode* agent saves the experimental data collected during the round.

We cannot assume that the best tour recorded by *MasterNode* is actually the best tour computed during an experimental round. So we must determine minimum of the best tours computed by all node agents together with the time when this tour was found.

Let us suppose that we have n node agents and k experimental rounds. For each node agent $j \in \{1, 2, \ldots, n\}$, let $t_{i,j}$ be the time of the last update of the best tour performed by node agent j in round $i \in \{1, 2, \ldots, k\}$, and let $v_{i,j}$ be the cost of the corresponding tour. *MasterNode* agent will collect values $v_{i,j}$ and $t_{i,j}$ and will determine the solution v_i and associated time t_i in round i as shown in first two rows of Table II.

The experimental data that were acquired during all the rounds of an experiment are post-processed to calculate performance measures as shown in last five rows of Table II.

Initial performance analysis. We experimented with ACODA on increasingly complex search versions (see Table III) in order to establish its effectiveness for implementing ACO-based algorithms. We considered our ACS-based version of ACO, together with other three distributed search methods – Random Choices, Cost Only, and Pheromone Search. The first two are not ACO-based, while the third is ACO-based.

For the Random Choices (RC) version we replaced equation (1) with randomly choosing the next hop of an ant from the unvisited nodes – i.e. the deposited pheromone and the edge weights are ignored.

For the Cost Only (CO) version we choose the next hop using equation (2), setting $\alpha = 0$ and $\beta = 1$ – i.e pheromone deposits are not taken into account.

Pheromone Search (PS) is an ACO-based search that uses equation (1) to determine the next hop. PS allows all ants to deposit pheromone regardless of tour cost. For this model, the ACO parameters were set to the values recommended by [4] for the ACS algorithm: $\tau_0 = 1/(n^2 w_{avg})$, $\rho = \xi = 0.1$, $\alpha = 1$, $\beta = 5$.

Finally, the ACS-based version of ACODA uses the same parameters as PS, but it allows only the ants that found the best tour so far to deposit pheromone. For all searches, the total number of ants is chosen equal to the number of nodes n (for each node the ant population consists of a single ant) as recommended in [4].

Note that, as we focus on finding the best solution, we chose $q_0 = 0$ in Equation (1), thus avoiding the convergence to a suboptimal solution (this fact is relevant for PS and ACS versions).

We ran several experiments using the following benchmark TSP maps selected from TSPLIB [6]: eil51, st70, kroA100,

TABLE II: Calculation of performance measures.

$v_i = \min_{j=1}^{n} v_{i,j}$	v_i is the solution found by round i.
$t_i = \min_{j=1}^{n}\{t_{i,j} \mid v_{i,j} = v_i\}$	t_i is the time in which solution was found by round i.
$v_{avg} = (\sum_{i=1}^{k} v_i)/k$	v_{avg} is the average value of solutions found in all rounds.
$t_{avg} = (\sum_{i=1}^{k} t_i)/k$	t_{avg} is the average time in which solutions were found in all rounds.
$v_{min} = \min_{i=1}^{k} v_i$	v_{min} is the best solution found after carrying out all rounds.
$t_{min} = \min_{i=1}^{k}\{t_i \mid v_i = v_{min}\}$	t_{min} is the time taken to find the best solution in all rounds for the first time.
$T_{avg} = (\sum_{i=1}^{k} T_i)/k$	T_{avg} is the average execution time of the rounds in an experiment.

TABLE III: Algorithms implemented on ACODA. "nop" means that the corresponding function has an empty body for that specific case.

	RC	CO	PS		ACS	
evaporation	no	no	no	yes	no	yes
LOCAL_EVAPORATE_PHEROMONE()	nop	nop	nop	eq (5)	nop	eq (5)
DEPOSIT_PHEROMONE()	nop	nop	eq (6)	eq (4)	eq (6)	eq (4)
Observations:	no pheromone update	no pheromone update	all ants deposit pheromone		only the best tour is marked	
RANDOM_CHOICE()	random unvisited neighbor	eq (2) $\alpha = 0$, $\beta = 1$	eq (1) $\alpha = 1$, $\beta = 5$		eq (1) $\alpha = 1$, $\beta = 5$	

ch150. Note that the number in the map name indicates the number of nodes of the map, so for example map st70 contains 70 nodes. These maps were chosen to experiment with different values of w_{avg} and $\Delta w = w_{max} - w_{min}$ where Δw is the difference between the maximum and the minimum edge weight and w_{avg} is the average edge weight.

Taking into account that we create an agent for each node, it follows that for st70 problem 70 agents were created. These agents are evenly distributed on 7 computers, so 10 agents must be created on each computer for solving st70 problem.

We define an ant move as the action of transferring an ant from a source node to destination node along a given edge. The number of ant moves was chosen according to the proposal from [4]. There, 1000 moves per ant were chosen for a 19 node map. Taking into account the sizes of our maps and scaling up proportionally, we determine a number $M = 10000$ of ant moves for our experiments.

We ran 10 rounds for each experiment on networks of 7 computers with dual core processors at 2.5 GHz and 1GB of RAM memory. These workstations were interconnected using a high-speed Myrinet interconnection network at 2Gb/s.

In the CO model the ants pick the smallest available weight (i.e the nearest unvisited neighbor) with a greater probability than the rest. This strategy guides the ants to worse solutions than the RC model. As expected, in the RC model, the difference $v_{avg} - v_{min}$ is larger as the tours variate in cost more.

In Table V we considered both evaporation scheme suggested by [4] for ACS (equation (4); see rows marked with $evap = 1$ on Table V) and absence of evaporation (equation (6); see rows marked with $evap = 0$ on Table V). Note that the PS model has no significant benefit from using the ACS evaporation scheme with the recommended parameter values in [4]. As a matter of fact in most cases evaporation decreases the quality of the solutions, see Table V for $evap = 0$ the values of v_{min} and v_{avg} are better than those for $evap = 1$. This is not the case for the ACS version of ACODA where evaporation improves the solutions. This suggests there is still room for improvement in the PS model. However, further study is needed in order to establish wether the evaporation parameters should be adjusted or a completely new approach should be developed.

Our experiments (see Table V) clearly show that the ACODA versions that take into account pheromone deposits are much more efficient at determining good solutions since the values of v_{min} and v_{avg} from Table V are better than those in Table IV. The fact that all these variations can be easily implemented using ACODA supports the flexibility of this architecture, while the fact that the best results are obtained when pheromone deposits are taken into account shows the effectiveness of ACODA in supporting distributed forms of ACO.

V. Related work

TSP is a classic benchmark problem for heuristic search algorithms. With the advent of distributed computing technologies, distributed versions of heuristic algorithms for TSP were also proposed. However, based on our literature review, there are very few works that propose ACO-based truly distributed TSP algorithms. Moreover, there are even fewer proposals that utilize recent advances of multi-agent systems middleware for ACO-based TSP [5]. Nevertheless, we could find references to multi-agent approaches to ACO algorithms for other combinatorial optimization problems ([7], [8], [9]).

A closely related approach to agent-based distributed ACO is presented in [5]. There, both graph nodes and ants are

TABLE IV: Experimental results for RC and CO.

	RC					CO				
map	v_{avg}	$t_{avg}[s]$	v_{min}	$t_{min}[s]$	$T_{avg}[s]$	v_{avg}	$t_{avg}[s]$	v_{min}	$t_{min}[s]$	$T_{avg}[s]$
eil51(426)	1390	85.6	1240	1.72	37.6	1263.8	22	1236	1	38.4
st70(675)	2504.6	53.6	2386	48.9	58.6	2612.6	18.3	2596	6.2	57.8
kroA100(21282)	38729.8	31.4	35276	29.6	99.3	78510.4	49.7	76387	90	145.4
ch150(6528)	46409	58.5	44815	148.1	209.4	31724.8	45.3	30766	186.8	201.2

TABLE V: Experimental results for ACS and PS.

		ACS					PS				
map	evap	v_{avg}	$t_{avg}[s]$	v_{min}	$t_{min}[s]$	$T_{avg}[s]$	v_{avg}	$t_{avg}[s]$	v_{min}	$t_{min}[s]$	$T_{avg}[s]$
eil51	1	448.6	14	440	27.2	40.1	482.2	15.8	471	2	38.1
(426)	0	453.2	19.3	444	7.2	39.7	449.4	12.8	442	10.9	39
st70	1	688.4	37.4	682	47.2	55.2	688	25.3	684	6.4	56.2
(675)	0	729.2	29.2	722	20.2	54.5	684.6	28.1	679	20.8	54.6
kroA100	1	23066	66.7	22380	60	91.9	22601.8	53.8	22286	31.6	92.3
(21282)	0	24887	53	24396	76.7	94.2	22997	29.1	22605	77.3	92
ch150	1	7263	86.6	7150	131.2	188.6	7729.4	17.4	7638	26	194.2
(6528)	0	7930.4	45.3	7806	6	194.7	6753.8	71.7	6692	57.5	192.8

implemented as JADE software agents. Ants centralize information about pheromone deposits and nodes' best tour cost through a single ACL message exchange per node [3]. This procedure adds up to $2n$ messages per tour per ant, where n is the number of nodes. Each ant has to notify the node about its next hop and the cost of its tour in order for the node to be able to update its pheromone levels. This generates other n messages. When an ant completes a tour, it compares the tour cost with the collected best tours from the nodes. A best tour synchronization is triggered for all the nodes if a better tour has been found. This brings an additional overhead of n messages. So, [5] approach requires at most $4n$ messages per tour per ant, while our approach requires at most $2n$ messages: n messages (ant moves) to complete a tour and n messages to deposit the pheromone. We avoid the additional overhead of sending to nodes all the information necessary to carry out their tasks, as in ACODA this information is already contained in ant messages exchanged between nodes.

ACODA implementation reported here is based on the sequential ACS algorithm presented in [4]. There are however three notable differences: i) We avoid the explicit iterations of the standard ACS. An iteration lasts until each ant has found a tour. With the distributed architecture it would be time consuming to provide the synchronization necessary for implementing the explicit iterations, because it would cancel the main benefits of an asynchronous, distributed architecture; ii) In ACS, best tours are compared after each iteration, thus allowing only the best ant to mark its tour. Again, this would require synchronization and centralized computation and we avoid it by allowing all ants to mark their tours; iii) In ACS ants move synchronously, taking one step at a time, while in our approach ants move asynchronously.

Papers [8] and [9] propose JABAT – a JADE-based middleware for agent teams. JABAT supports distributed implementation and collaboration of population-based optimization algorithms. In particular, JABAT was applied to TSP. There is however an important difference between JABAT and ACODA. JABAT agents represent improvement algorithms, which basically means that improvement algorithms are sequential and they cooperate for solving a problem in a distributed way. ACODA agents provide a natural distributed model of the problem environment that is suitable for distributed ACO algorithms. Moreover, we could not find scalability studies referring to JABAT.

Paper [7] proposes a JADE-based multi-agent environment for dynamic manufacturing scheduling that combines intelligent techniques of ACO and multi-agent coordination. However, the focus in [7] is to evaluate the impact of ACO intelligence on multi-agent coordination, rather than to utilize multi-agent middleware for improving ACO. ACO intelligence is embedded into job and machine agents with the role of ants, which is different from ACODA where ants are passive objects exchanged by software agents that provide a distributed model of the problem environment.

In [10] authors compare a distributed form of ACS with flooding algorithm applied on resource discovery problem. They experiment with both algorithms using *ns-2* network simulation tool [11]. The down side of this is that no real execution time measure can be made using their approach. They showed that ACS is the better approach in terms of: best success rate, least number of hops and least traffic. The detailed algorithm and ACO parameters are not presented in order to duplicate their approach using our ACODA framework, for a realistic comparison. The differences between their approach and ACODA are: (i) resource queries are handled centrally at a single anthill, thus introducing single point of failure, (ii) they do not take edge weights into account as they are trying to solve the resource discovery problem, not the TSP problem, and (iii) ants are implemented as *ns-2* mobile agents, while in ACODA we are using JADE. Moreover, in practice there is no need for code mobility as every ant is governed by the same behavioral rules. So, in our approach we use "nodes as agents" to control ants' movement using messages, rather than using code (i.e. JADE agent) mobility.

Paper [12] proposes purely theoretical frameworks for multi-agent systems. No experiments or implementations are mentioned. The authors present a distributed form of ACO based on so called "smart messages" approach to multi-agent systems where agent mobility is used to implement complex communication over dynamic networks. They use delegate multi-agent systems to manage these smart messages in order to design a multi-agent approach for ACO. They do not present ways of representing the environment, determining convergence or stopping condition of ACO experiments.

In [13] authors thoroughly analyze multi-colony ACO algorithms applied on TSP. The main difference between their and our approaches is that they run a separate colony on each available processor. At predetermined points in time, the solutions are centrally collected and local search is performed on the best solution provided by each of the colonies. This is not a fully decentralized approach to ACO, as it requires synchronization and centralized sequential local search after each solution was centrally collected. Our approach is completely decentralized, parallel and asynchronous, and thus it could lend itself to heavy parallelization.

VI. Conclusions

In this paper we presented experimental results with our new multi-agent framework for truly distributed ACO. An initial implementation of the framework using JADE multi-agent platform was outlined. This implementation followed the ACO approach initially proposed for the ACS system. However, three important differences between ACODA and original ACS system were highlighted. This approach was initially evaluated on sample benchmark TSP problems. Experimental results were encouraging, as they clearly support the effectiveness of our proposal for distributed ACO, thus confirming the feasibility of our ACODA framework. We already identified suitable directions for future works: i) strengthen the scalability results by experimenting with this approach on larger TSP problems and larger computer networks; ii) support the generality of our framework by considering other forms of ACO rather than only ACS; iii) improve the ACODA architecture to enable experimentation with larger TSP problems than the current version of ACODA is able to support.

Acknowledgment.

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.

References

[1] S. Ilie and C. Bădică, "Distributed multi-agent system for solving traveling salesman problem using ant colony optimization," in *Intelligent Distributed Computing IV, Proc.4th International Symposium of Intelligent Distributed Computing, IDC'2010*, ser. Studies in Computational Intelligence. Springer, 2010, vol. 315, pp. 119–130.

[2] M. Wooldridge, *An Introduction to MultiAgent Systems.* John Wiley & Sons Ltd, 2002.

[3] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE.* John Wiley & Sons Ltd, 2007.

[4] M. Dorigo and T. Stutzle, *Ant Colony Optimization.* MIT Press, 2004.

[5] E. Ridge, D. Kudenko, and D. Kazakov, "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 (NISPADE)*, 2006.

[6] G. Reinelt, "Tsplib - a traveling salesman library," *ORSA Journal on Computing*, 1991, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[7] W. Xiang and H. P. Lee, "Ant colony intelligence in multi-agent dynamic manufacturing scheduling," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 1, pp. 73–85, 2008.

[8] D. Barbucha, I. Czarnowski, P. Jedrzejowicz, E. Ratajczak, and I. Wierzbowska, "Jade-based a-team as a tool for implementing population-based algorithms," in *Proc.6th International Conference on Intelligent Systems Design and Applications: ISDA'2006.* IEEE Computer Society, 2006, pp. 144–149.

[9] I. Czarnowski, P. Jedrzejowicz, and I. Wierzbowska, "A-team middleware on a cluster," in *Proc.3rd KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications: KES-AMSTA'2009*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2009, vol. 5559, pp. 764–772.

[10] S. M. Fattahi and N. M. Charkari, "Ant distributed acs algorithm for resource discovery in grid," *Special Issue of the International Journal of the Computer, the Internet and Management*, vol. 17, no. SP1, 2009.

[11] The ns-2 project, a network simulation tool. Http://nsnam.isi.edu/nsnam/index.php/.

[12] T. Holvoet, D. Weyns, and P. Valckenaers, "Patterns of delegate mas." Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 1–9.

[13] C. Twomey, T. Stützle, M. Dorigo, M. Manfrin, and M. Birattari, "An analysis of communication policies for homogeneous multi-colony aco algorithms," *Inf. Sci.*, vol. 180, no. 12, pp. 2390–2404, 2010.