Parallel, Asynchronous and Decentralised Ant Colony System

Enda Ridge*
*Department of Computer Science
University of York
ERidge@cs.york.ac.uk

Edward Curry†
†Department of Information Technology
National University of Ireland, Galway
EdCurry@acm.org

Daniel Kudenko*
Kudenko@cs.york.ac.uk

Dimitar Kazakov*
Kazakov@cs.york.ac.uk

Abstract

This paper describes a multi-agent system architecture that would permit implementing an established and successful nature-inspired algorithm, Ant Colony System (ACS), in a parallel, asynchronous and decentralised environment. We review ACS, highlighting the obstacles to its implementation in this sort of environment. It is suggested how these obstacles may be overcome using a pheromone infrastructure and some modifications to the original algorithm. The possibilities opened up by this implementation are discussed with reference to an elitist ant strategy. Some related exploratory work is reported.

1 Introduction and Motivation

It is often desirable to design large scale distributed applications in a parallel, asynchronous and decentralised (PAD) fashion. Furthermore, many emerging and proposed applications such as Grids and Peer-to-Peer systems will be constrained to operate in such environments. However, despite several notable successes (Brueckner, 2000), designing such systems still presents formidable challenges.

How do we enable decentralised control? How do we coordinate large numbers of agents? How do we communicate efficiently between so many agents? Natural systems of flocks and swarms provide evidence that these questions can be addressed. However, with some exceptions, the majority of established and successful applications based on nature are sequential, synchronous and centralised (e.g. the genetic algorithm, ant colony optimisation). Can we harness the power of these established algorithms in PAD environments and so help address these challenges?

This paper proposes an architecture for implementing a well-known and successful nature-inspired algorithm, Ant Colony System (ACS) in a parallel, asynchronous and decentralized environment.

The next section reviews the ACS algorithm for completeness. Section 3 describes how to implement this algorithm in a parallel, asynchronous and decentralised environment. Section 5 reviews related work. Section 6 concludes with a summary of some ongoing exploratory research and an outline of future work.

2 ACS Algorithm

The original ACS algorithm was applied to the Travelling Salesperson Problem (TSP). The TSP involves finding the shortest route between a set of cities such that no city is visited more than once. The TSP is often represented as a graph structure (Figure 1).

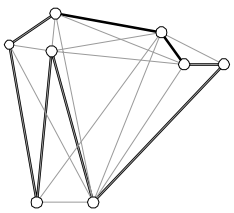

Figure 1: Illustrative TSP graph (some edges omitted for clarity). Darker edges illustrate a tour.

The Ant Colony System (ACS) algorithm is summarized in Figure 2. The 'pheromone' applied to edges is an abstraction of the chemical markers used by real ants. Edges with high pheromone levels are more attractive to ants. All ants build their tours using a probabilistic decision rule., The *local pheromone update* involves decaying the pheromone level on an edge traversed by an ant by a small amount. Once all ants have built a tour, pheromone is deposited along the best ant's tour in a *global pheromone update*. The whole process then repeats.

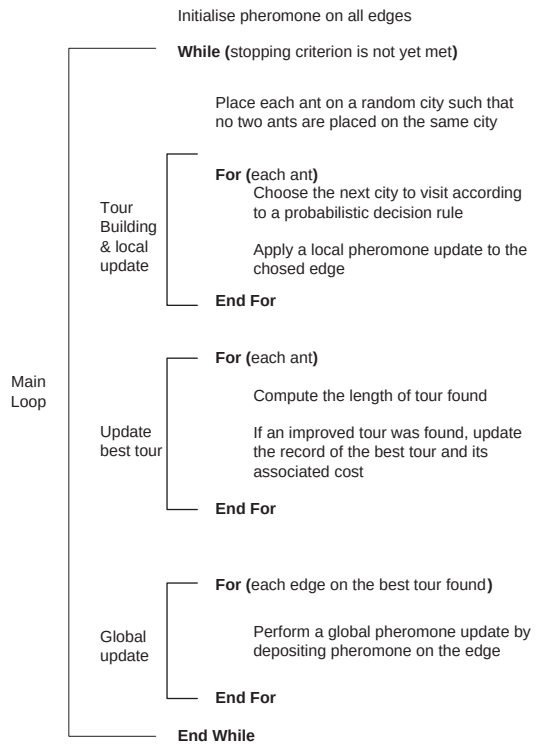

Figure 2: Pseudocode for ACS

2.1 Issues

This paper addresses the two major obstacles to implementing this algorithm in a parallel, asynchronous and decentralised fashion.

- Firstly, the algorithm is highly synchronous. Each ant performs its own tour and local pheromone update in turn [1]. The algorithm waits for all ants to finish this phase before a global pheromone update is performed. Once the global pheromone update is complete, all ants begin the tour building phase again.
- Secondly, the global pheromone update requires a *centralised* comparison of all the ant tours to acquire the *global knowledge* of the system's best tour.

Clearly, a direct implementation of this algorithm would be expensive and inefficient in a parallel, asynchronous and decentralised environment. The next section proposes how these obstacles can be overcome.

[1] We have seen conflicting pseudocodes in the literature. Some ACS implementations construct solutions in parallel and not one after the other as seen here. We are currently investigating whether these differences have an effect on performance.

3 Parallel, Asynchronous, Decentralised ACS

We propose a Multi-Agent System (MAS) platform as the basic framework on which the algorithm should be implemented. Common MAS platforms (Bellifemine et al., 2001) provide a convenient means of distributing computation over machines while coordinating with asynchronous messaging.

3.1 Pheromone Infrastructure for ACS

Brueckner first introduced the concept of a pheromone infrastructure in the context of manufacturing control (Brueckner, 2000). Briefly, this approach involves representing the environment by a topology of *Place agents*. These Place agents manage 4 pheromone functions: aggregation, evaporation, propagation and sensing (Parunak et al., 2004). Such an infrastructure can be used to methodically move the ACO algorithm into a PAD environment (Ridge et al., 2005).

Each city in the TSP (Figure 1) is represented by a Place agent. Each ant from ACO becomes a *Solution agent* that interacts with this pheromone infrastructure. Figure 3 is a schematic of our architecture overlaying the previous TSP graph of Figure 1.

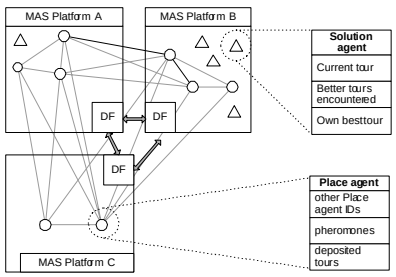

Figure 3: Architecture showing 3 MAS platforms. Nodes in the previous TSP graph are now Place agents distributed on the platforms. Some platforms are also hosting Solution agents.

3.2 Agent Interactions

In this scenario, all Place agents register with one another using a service description containing their city's coordinates. Place agents can then calculate the distance to any other Place agent. Place agents maintain a record of the pheromone levels on each link connecting to other Place agents.

The Solution agent lifecycle consists of the following interactions with the pheromone infrastructure (Figure 4).

1. A Solution agent 'arrives' at a given Place agent with an INFORM message containing details of where the Solution agent has come from.
2. The Place agent performs a local pheromone decay on the relevant link.
3. The Place agent responds with (1) a list of other Place agent IDs (acquired from the Place agent registrations), (2) the calculated cost to each of the other Place agents and (3) the latest pheromone level on the links to each of the other Place agents.
4. The Solution agent uses this information to decide which Place agent to visit next.
5. The Solution agent informs the Place agent of its decision to move to a given destination Place agent.
6. The Place agent updates its pheromone value for that link using the equivalent of the algorithm's *local pheromone update*.
7. Life cycle returns to step 1, with the Solution agent arriving at the destination Place agent

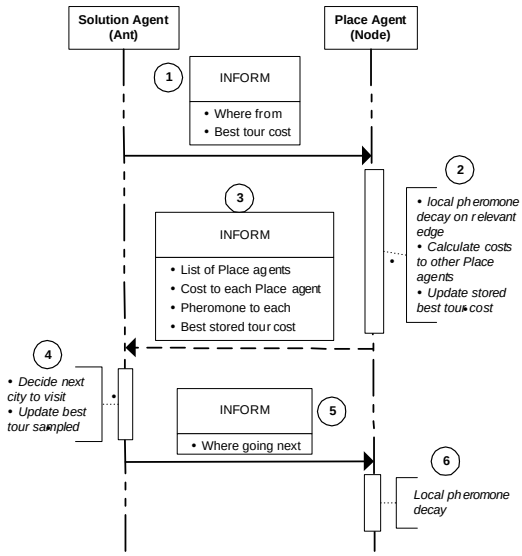

Figure 4: Solution agent interactions

3.3 Global Pheromone Update

Global pheromone updates are performed by Solution agents as follows. When a solution agent moves between Place agents (Step 4), it keeps track of the cost associated with that move and the Place agents involved in that move. Once it has visited all Place agents, it has a total cost for the tour it has completed. The Solution agent can then deposit the associated amount of pheromone with the Place agents and the Place agents can adjust their pheromone levels accordingly.

Thus far, we have described the mechanisms and interactions that permit a *direct* implementation of ACS on a pheromone infrastructure. Recall the two main issues identified previously: that of the *synchronous* tour building and global update phases and that of a global update that depends on the *centralised* global information of the best tour produced. We now address these issues with suggested modifications to the architecture and algorithm.

3.4 Modifications

Let each Place agent be capable of storing a description of a tour and that tour's cost. When Solution agents 'move' between Place agents (Steps 1 to 3), they deposit with the destination Place agent the cost of the best tour the Solution agent has ever performed. If the Place agent's current stored cost is worse (higher cost) than the deposited cost, the Place agent overwrites its stored cost. The Place agent informs the Solution agent of its stored cost. In this way, a measure of the globally best tour is distributed around the pheromone infrastructure of Place agents. There is no longer a centralised comparison requiring the global knowledge of all tours. Triggering a global update is a small addition to this procedure. When a Solution agent completes its tour, it checks to see what the lowest tour cost encountered was. If the Solution agent's best ever tour cost is less than the lowest tour encountered in the environment, the Solution agent knows its has the best tour. That Solution agent then performs a global update.

3.5 Further Possibilities

When we relax the constraint of remaining as true as possible to the original algorithm, our idea of depositing tour information in the environment introduces many possibilities. The 'Elitist' ant strategy has been found to improve Ant System, the precursor to ACS. This strategy permits e elitist ants to perform a global update on the best tour found so far. In

our framework, Solution agents can count the number of better tours encountered in the environment and perform a global update if they are within the top t tours encountered. Alternatively, they could track the percentage difference between their own tour and the best tour encountered and perform a global update accordingly.

4 Preliminary Results

Currently, we are testing the effect of asynchronicity, parallelism and concurrency on the performance of the original Ant Colony System algorithm. There is an inherent bias in its standard implementation (Figure 2) in that ants build full tours one at a time and this *process order* is repeated for the lifetime of the algorithm. For the implementation investigated with this research, we should like to rule out that asynchronous and parallel tour building have any effect on algorithm performance. We should not assume this is the case. Recall that stigmergic mechanisms such as pheromones rely on sensing signals previously deposited in the environment.

In our experiments, we tested two independent variables—the process order used by ants when building tours and the number of steps towards a full tour during an ant's turn. We used two levels of process order—fixed and random. We also used two levels of the process size—an ant solves the whole problem on its turn or an ant makes one step towards solving the problem on its turn. The accuracy, speed and reliability of the algorithm were measured. We found no statistically significant difference in ACS's performance between all combinations of all levels of the independent variables. We are now performing similar experiments on the effect of the number of concurrently active ants on the original algorithm's performance.

5 Related Work

Randall and Lewis (Randall and Lewis, 2002) applied a parallelisation strategy to 8 TSPLIB problems ranging in size from 24 to 657 cities. Their results showed an improvement in speedup and efficiency of solution for problem sizes greater than 200 cities. However, their scheme had a high communication cost.

Stützle has experimented with parallel independent runs of Max-Min Ant System on 7 instances from TSPLIB ranging in size from 198 to 1291 cities (Stutzle, 1998). These experiments showed performance improvements over a single sequential run. Both Stützle's and Randall and Lewis' experiments were in the vein of a traditional parallel computing master/slave approach as opposed to the completely decentralised MAS approach described by this paper. The reader is referred to the literature (Janson et al., 2005) for a comprehensive overview of approaches to parallelisation of ACO.

6 Future Work

Our immediate future work will involve building and testing the architecture and ACS implementation described in this paper. We would also like to consider other ant colony variants such as Max-Min Ant System and other nature-inspired approaches such as Particle Swarm Optimisation.

References

F. Bellifemine, A. Poggi, and G. Rimassa. JADE: a FIPA2000 compliant agent development environment. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 216–217. 2001.

S. Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Phd, Humboldt University, 2000.

S. Janson, D. Merkle, and M. Middendorf. Parallel Ant Colony Algorithms. In *Parallel Metaheuristics*. Wiley, 2005.

H. V. D. Parunak, P. Weinstein, P. Chiusano, and S. Brueckner. Agents Swarming in Semantic Spaces to Corroborate Hypotheses. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1488–1489. 2004.

M. Randall and A. Lewis. A Parallel Implementation of Ant Colony Optimization. *Journal of Parallel and Distributed Computing*, 62(9):1421–1432, 2002.

E. Ridge, D. Kudenko, D. Kazakov, and E. Curry. Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments. In *Self-Organization and Autonomic Informatics*, pages 35–49. IOS Press, 2005.

T. Stutzle. Parallelization Strategies for Ant Colony Optimization. In *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*. 1998.