Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments

Enda Ridge [a,1], Daniel Kudenko [a] , Dimitar Kazakov [a] and Edward Curry [b]
[a] *The Department of Computer Science, The University of York, U.K.*
[b] *The Department of Information Technology, The National University of Ireland, Galway*

Abstract. This paper motivates research into implementing nature-inspired algorithms in decentralised, asynchronous and parallel environments. These characteristics typify environments such as Peer-To-Peer systems, the Grid and autonomic computing which demand robustness, decentralisation, parallelism, asynchronicity and self-organisation. Nature-inspired systems promise these properties. However, current implementations of nature-inspired systems are only loosely based on their natural counterparts. They are generally implemented as synchronous, sequential, centralised algorithms that loop through passive data structures. For their successes to be relevant to the aforementioned new computing environments, variants of these algorithms must work in truely decentralised, parallel and asynchronous Multi-Agent System (MAS) environments. A general methodology is presented for engineering the transfer of nature-inspired algorithms to such a MAS framework. The concept of pheromone infrastructures is reviewed in light of emerging standards for agent platform architecture and interoperability. These ideas are illustrated using a particularly successful nature-inspired algorithm, Ant Colony System for the Travelling Salesman Problem.

Keywords. Ant Colony Algorithms, Ant Colony System, decentralised, parallel, asynchronous, Multi-Agent System, pheromone infrastructures

1. Introduction and Motivation

Nature-inspired algorithms such as genetic algorithms [1], particle swarm optimisation [2] and ant colony algorithms [3] have achieved remarkable successes. Indeed, they are the state-of-the-art solution technique for some problems. The algorithms share the characteristic of being loosely based on a natural metaphor such as evolution's search through the vast space of potential organisms, the movement of flocks of birds through a 3D space or the self-reinforcing chemical trails laid by ants while searching for a route through a 2D space.

The natural systems on which these algorithms are based possess many desirable properties that we would like to transfer to our computer systems.

---

[1]Correspondence to: Enda Ridge, The Department of Computer Science, The University of York, Heslington, York YO10 5DD, UK. Tel.: +44 1904 43 2722; Fax: +44 1904 43 2767; E-mail: ERidge@cs.york.ac.uk.

- They typically contain large numbers of relatively simple participants.
- They are completely decentralised.
- They operate in parallel and asynchronously.
- They use relatively simple signals
- Their desired functionality emerges from the interactions of their participants. This is achieved despite (and because of) their simple participants that have no global information.

These characteristics make natural systems robust to loss of members, parallel in ‘execution’, and adaptable to a changing problem domain.

However, the efficient pursuit of increasingly accurate nature-inspired solutions that can compete with more traditional algorithms has lead researchers to resort to quite ‘unnatural’ designs for their nature-inspired algorithms. The result is that the most successful versions of these algorithms are often centralised and sequential implementations that are highly tuned to a particular problem. They bear only a tenuous resemblance to their successful counterparts in nature. This renders them brittle in the face of the dynamism of changing problem specifications and operating conditions and limits their usefulness to industry’s direction of increasing distribution, decentralisation and adaptability.

The pursuit of improved performance at optimisation and search is a very worthwhile endeavour. Yet the pursuit of these goals in a nature-inspired algorithm, to the exclusion of all else, sacrifices the real strengths of natural systems—their robustness, adaptability, decentralisation and parallelism.

It is these very properties that are coming to the fore in emerging computer environments such as autonomic computing [4], ubiquitous and pervasive computing, Peer-to-Peer systems, the Grid [5] and the Semantic Web [6]. These environments demand systems that are robust to failures, adaptable to changing requirements and deployment scenarios, composed of relatively simple components for ease of development and maintenance and are preferably decentralised and parallel.

Multi-agent Systems (MAS) [7] provide a platform on which many independent, parallel and asynchronous software entities can interact using standard protocols and ontologies. It is clear that if nature-inspired algorithms are to be useful in the aforementioned environments, then they must embrace the characteristics of a MAS-type platform. It is not sufficient for researchers to claim their algorithms are robust or parallel while implementing them as sequential, synchronous and centralised loops through passive data structures. For example, cellular automata, which can be considered a sequential abstraction of MASs, yield different results when different approximations to parallel update are used [8]. It is reasonable to suspect that approximations of other characteristics such as asynchronicity do not accurately predict the true system behaviour but merely hint that asynchronicity may be a *factor* in performance. Approximations are not sufficient for drawing conclusions. They are only the first stage of a complete research process [9]. Furthermore, decentralised agents can exhibit other phenomena such as hyperactivity [10] and diversity [11] and their designers must address the real-world costs of messaging [12]. Experiments with MAS implementations of a genetic algorithm have yielded unexpected results that were partly due to the cost of messaging [13]. Equally unexpected results may be uncovered with asynchronous and parallel implementations of other nature-inspired algorithms.

The similarities between natural environments and emerging computing environments motivate disciplined scientific and engineering investigations into the successful transfer of these *algorithms*, *techniques* and *infrastructures* into such environments.

This paper studies Ant Colony System (ACS) [14], a very successful nature-inspired algorithm from the ant colony family of algorithms. It explores how ACS might be transferred to such emerging computing environments as can be represented by a truly decentralised, asynchronous and parallel Multi-Agent System, in a way that complies with emerging standards for agent interaction and architecture [15].

The next section gives an overview of the ACS algorithm for the Travelling Salesman Problem. Section 3 reviews the idea of pheromone infrastructures in light of maturing agent standards. Section 4 presents a general strategy for transferring nature-inspired algorithms to MAS-type platforms and illustrates this with reference to the ACS algorithm. The paper concludes with a review of related work, our conclusions and our direction for future work.

2. Ant Colony System for the Travelling Salesman Problem (ACS-TSP)

We briefly review Ant Colony System for the Travelling Salesman Problem (ACS-TSP) for completeness and to draw attention to the issues with centralised, sequential and synchronous nature-inspired algorithms. The reader is directed to other works [3,16] for a comprehensive background, description and discussion of this algorithm.

The Travelling Salesman Problem (TSP) [17] involves finding the shortest route through a given set of cities such that no city is visited twice. The graph representation of the problem is constructed as a set of nodes and edges where nodes represent the cities and edges join every node (city) to every other node. Activities of the ants then select a subset of these edges that represents a valid travelling salesman tour. The TSP is heavily researched and has a well-established set of benchmark problems with solutions [18]. Furthermore, it is an abstraction of a large class of discrete combinatorial optimisation problems that is easily visualised and understood by non-experts. These types of problems are particularly relevant to Grid and Autonomic computing as efficient task and resource allocation are recurring themes in these environments.

The Ant Colony System algorithm [14] for solving the TSP is loosely based on natural ant colony behaviour when foraging for food. Ants wander away from their nest in search of food. After finding food, they return to their nest, laying a chemical marker called a pheromone. Subsequent ants leaving the next are more inclined to follow strong pheromone trails. This positive reinforcement builds efficient trails to food sources in a decentralised fashion. The Ant Colony System algorithm adapts this process to a graph structure. Figure 1 gives the pseudocode for the algorithm.

Initially, every edge in the problem is given the same pheromone level τ_0. A number of ants m are randomly assigned to their starting cities such that no more than one ant occupies any city. The following main loop then proceeds for a maximum of t_{max} iterations where t is the count of the current iteration. Each ant builds its own tour by repeatedly applying a decision rule to determine the next city it will visit. The ant's list of cities that remain to be visited is stored in a list J. The ant first creates a random number q. This q is compared to an exploration/exploitation threshold q_0. When q exceeds the threshold, the ant's choice of next city favours choosing less frequented trails

```
/******** Initialisation ********/
Set the pheromone level on each edge to $\tau_0$
Place ant k on a randomly chosen city such that there is no more than one ant per city.
Let $T^+$ be the shortest tour found so far and $L^+$ its length.
/**************** Main Loop ****************/
For $t = 1$ to $t_{\max}$ do
    For $k = 1$ to m do
     /**** Build ant's tour by applying the following rule $n-1$ times ****/
        If there exists at least one unvisited city on the candidate list then:
            Choose a random value for the exploration variable $q$
            Choose next city $j \in J_i$ from candidate list using
            $j = \begin{cases} P & \text{if } q > q_0 \\ \arg\max_{u \in J_i} \{\tau_{iu}{}^{\alpha} \cdot \eta_{iu}{}^{\beta}\} & \text{if } q \le q_0 \end{cases}$
            where city $P$ is chosen using the following probability function:
            $p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in J_i} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}}$
            and $i$ is the ant's current city.
        Else
            Choose the next closest city
        End If
     /********* Local Pheromone Update *********/
        With the edge $(i, j)$ just chosen by the ant, update pheromone as follows:
        $\tau_{ij} = (1 - \rho_{\text{local}})\tau_{ij} + \rho_{\text{local}}\tau_0$
    End For
    If an improved tour is found then
        Update the values of $T^+$ and $L^+$
    End If
     /********* Global Pheromone Update *********/
    For every edge $(i, j)$ on the best tour $T^+$ Do
        Update pheromone trail by applying the rule
        $\tau_{ij} = (1 - \rho_{\text{global}})\tau_{ij} + \rho_{\text{global}} \frac{Q}{L^+}$
    End For
End For
/******* End of Program *********/
```

Figure 1. Pseudocode for ACS-TSP algorithm (adapted from [3])

(exploration). When q does not exceed the threshold, the ant favours well-established trails (exploitation). When at a current city i, the decision rule for a possible next city j is a function of the pheromone intensity τ_{ij} on the edge (i, j) and η_{ij}, the inverse of the length of edge (i, j). Each city has a candidate list of preferred cities to which the decision rule is applied. If all cities in the candidate list have already been visited, an ant does not use its decision rule and simply chooses the next nearest city. The decision rule to choose the next unvisited city $j \in J_i$ from the candidate list is given by

$$j = \begin{cases} P \quad \text{if } q > q_0 \\ \arg\max_{u \in J_i} \left\{ [\tau_{iu}]^\alpha \cdot [\eta_{iu}]^\beta \right\} \quad \text{if } q \leq q_0 \end{cases}$$

where P is chosen from the candidate list using probabilities p_{ij} generated with the following function:

$$p_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{l \in J_i} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}$$

When an ant moves between two cities, it immediately performs a *local pheromone update* on the edge joining those two cities. Once all ants have built their tours, any improvement on the best tour found is recorded. A *global pheromone update* is then performed on the current best tour. This is one of the most obvious centralised components of the ACS algorithm. The main loop then repeats.

ACS-TSP was competitive at finding shorter tours than other techniques such as a genetic algorithm, evolutionary programming and simulated annealing [14]. Comparisons were made on three benchmark problems [18] of size 50, 75 and 100 cities. However, for larger problems, a local search technique had to be introduced to the algorithm (ACS-3-opt) [14]. Again, we see the tendency for nature-inspired algorithms to drift away from their original natural counterparts. This research uses the original ACS-TSP.

3. Pheromone Infrastructures

In this section, we suggest how to implement a pheromone infrastructure suitable for the TSP and relate it to emerging agent standards for interoperability and architecture [15]. Pheromone infrastructures provide a truly parallel, asynchronous and decentralised environment that can support an ant colony nature-inspired system. The adoption of agent standards is an identified short-term goal of the agent community [19].

3.1. Overview of Pheromone Infrastructures

The idea of a 'pheromone infrastructure' has been proposed in the literature in the context of manufacturing control [20]. A pheromone infrastructure is an agent environment that supports some, if not all, of the 4 basic pheromone operations of *aggregation*, *evaporation*, *propagation* and *sensing* [21]. The environment is represented by a collection of *environment agents* restricted to local information and interaction. This facilitates building a distributed and decentralised environment that can run its own environment processes independently. These environment agents are termed *Place Agents* in keeping with the original literature [20]. Place agents manage the topology of the problem and the pheromone functions.

We term the other agents in the system *Solution Agents*. Solution agents move about on the pheromone infrastructure, interacting with the Place agents to perform pheromone operations such as sensing, deposition and perhaps pheromone removal.

3.2. FIPA-compliant Pheromone Infrastructures

Maturing agent standards facilitate applying standard protocols and ontologies to the original pheromone infrastructure idea. The Foundation for Intelligent Physical Agents (FIPA) is an 'IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies.' This standardisation is necessary to fulfil one of autonomic computing's 8 requirements[1] [4]. Calculations show that there is a significant messaging cost in enforcing such standards.

3.2.1. Directory Facilitators as Place Agents

The FIPA Agent Management Specification [22] provides for so-called Directory Facilitator (DF) agents. These offer a 'yellow pages' service by allowing other agents to advertise their particular service(s) with the DF. Other agents can then query a DF about its advertised services and subscribe for notification about the appearance of new services.

We can use a collection of DF agents to build a decentralised pheromone infrastructure of Place agents. Each Place (DF) agent offers its own service description as a Place agent. This description includes a description of its location in the environment. Neighbouring Place agents register their service descriptions with one another. These registrations between place agents form the topology of the environment.

Solution agents occupy a location in the environment by registering themselves with the Place agent representing that location. The solution agents can query their current Place agent for its advertised neighbours. This provides Solution agents with a 'visibility' of the environment. The Solution agents 'move' by deregistering from their current Place agents and registering with one of its advertised Place agent neighbours.

Each Place agent maintains a record of the deposited pheromone types and their associated concentrations. Solution agents can query this pheromone record and modify it by depositing and removing pheromones. The Place agents perform the pheromone functions of evaporation and propagation independently of the Solution agents.

3.2.2. Flexibility of Pheromone Infrastructures

This pheromone infrastructure approach is flexible enough to represent both graph-type and discrete space environments, the typical problem representations to which ant colony algorithms are applied. These environments are illustrated in Figure 2. In a discrete space partitioned into tiles, each Place agent represents a tile in the space. Adjacent tiles are listed in the registered service descriptions of the Place agents. Tiles typically describe themselves with an index. The approach for a graph-type environment is very similar. Each Place agent represents a node in the graph. 'Adjacent' nodes that share a common edge with the given node register their Place agent service.

[1] "An autonomic computing system cannot exist in a hermetic environment. While independent in its ability to manage itself, an autonomic computing system must function in a heterogeneous world and implement open standards—in other words, an autonomic computing system cannot, by definition, be a proprietary solution." ..

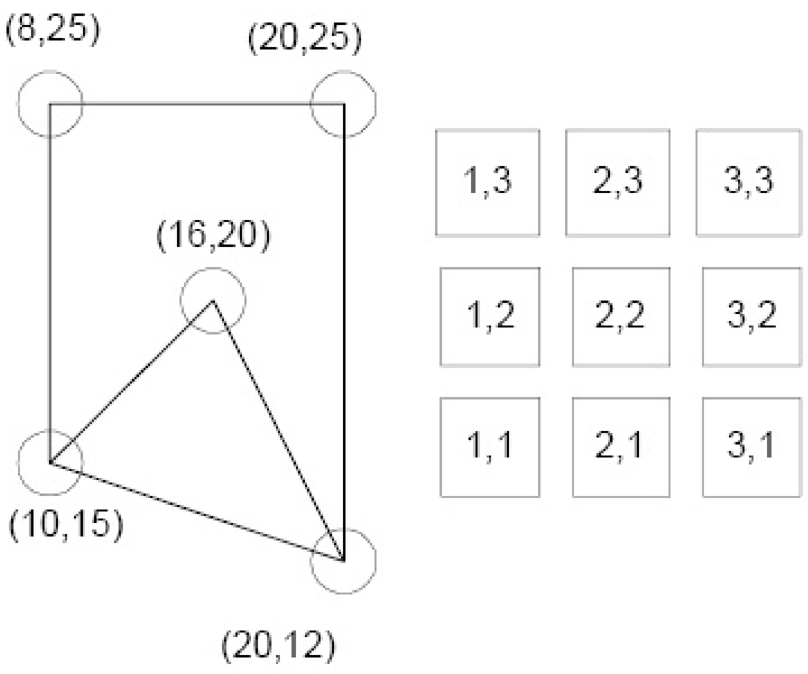

Figure 2. Graph-type (left) and discrete tile (right) environments

3.2.3. General Protocols

From the previous discussion, we can identify the necessary protocols and ontological concepts that would be common across pheromone infrastructures for all ant colony systems. These are summarised in Table 1.

4. A Methodology for Transferring Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments

In this section, we describe a general methodology for transferring a nature-inspired algorithm to a truly parallel, asynchronous and decentralised MAS-type environment. The strategy is illustrated with reference to the ACS-TSP algorithm described in Section 2. The steps in the methodology are as follows:

- Identifying solution agents
- Identifying global components
- Identifying parameters
- Identifying standard protocols
- Estimating messaging cost
- Deciding on ‘termination’ criteria

Table 1. FIPA standard protocols for a general pheromone infrastructure

Initiator	**Participant**	**Scenario**	**Standard Protocol**	**Comments**
Place agent	Place agent	Place agents register their Place service description with one another. This is a standard DF registration	FIPA Request Interaction Protocol [23]	
Place agent	Place agent	Place agents deregister their Place service description from one another. This is a standard DF deregistration.	FIPA Request Interaction Protocol [23]	
Solution agent	Place agent	A Solution agent asks its current Place agent for a list of neighbouring Place agents. This is a standard DF search.	FIPA Request Interaction Protocol [23]	
Solution agent	Place agent	A Solution agent asks its current Place agent for information on the local pheromone infrastructure.	FIPA Query Interaction Protocol [24]	query-ref and inform-result version is used.
Solution agent	Place agent	A Solution agent deposits pheromone with its current Place agent.	FIPA Request Interaction Protocol [23]	agree is omitted. inform-result is used to conclude.

4.1. Identifying Solution Agents

The first step is to identify the Solution Agents in the system. This identification should try to map agents to ‘physical entities’ rather than system functions [25]. This is because physical entities are relatively simple and have a locality of interaction and information whereas functions can be complicated and are usually defined globally. Other authors use the example of factory scheduling and draw the distinction between the physical machines on a factory floor and the global function of machine scheduling for the entire factory [25]. Solution agents must support the fipa-agent-management ontology and the associated protocols (Table 1) so that they may interact with and move around the pheromone infrastructure described in Section 3.2.

In ACS-TSP, the obvious assignment is a Solution Agent for each ant.

4.2. Identifying Global Components

As mentioned in the motivation, many nature-inspired algorithms have a centralised global component. The exact nature of such a component varies hugely between various algorithm implementations. However, it very presence prevents a decentralised implementation. This component must be removed or replaced with some decentralised equivalent.

In ACS-TSP, there is a global pheromone update stage after all the ants have completed their tours. This is global in two ways. Firstly, the ants must share some knowledge of the best tour found between them. Secondly, the ants must know that they have all finished their tours.

This can be avoided in a decentralised MAS by having ant Solution agents write their trails to some common blackboard. If the Solution agent sees that its tour is the best so far, the agent backtracks over that trail performing the 'global' update procedure.

4.3. Identifying Parameters

The categorisation of parameters is important. It allows us to clearly determine how decentralised our implementation can be while remaining true to the original algorithm.

Agent model parameters divide into three categories: problem parameters, solution parameters and environmental parameters [26]. We propose further dividing the Solution parameters category into *Global* and *Agent* Solution parameters.

1. **Problem parameters:** these vary the problem for which the algorithm is developed. For example, in the Travelling Salesman Problem, the number of cities is a problem parameter.
2. **Solution parameters:** these vary the characteristics of the algorithm itself. They are tailored during deployment.
   (a) **Global Solution parameters:** Global solution parameters can only be seen and modified from an omniscient viewpoint. For example, in a swarm, the number of agents is a global solution parameter.
   (b) **Agent Solution parameters:** Agent parameters could conceivably be applied to each individual agent and have different values for each agent. These parameters determine the diversity of swarm members.
3. **Environmental Parameters:** these vary the deployment scenario. Examples might include communication delays and computation node failures.

Preferably, as many solution parameters as possible should fall into the Agent category rather than the Global category. The ultimate goal of a fully decentralised system demands no global solution parameters. Table 2 and Table 3 summarise the global and agent solution parameters respectively for ACS.

Table 2. Global Solution Parameters in Ant Colony System

Symbol	**Description**	**Comments**
m	Number of ants	
N/A	Maximum number of ants per city initially	This was not an explicit parameter in the original account of ACS [14]

Note that although the ACS algorithm mentioned a single parameter ρ for use in both local and global pheromone updates, we distinguish between two respective values of ρ in keeping with our research motivation. Note also that all but one of the agent solution parameters is assigned to the Solution agents rather than the Place agents. This is desirable because it keeps the Place agents as general as possible.

Table 3. Agent Solution Parameters for Ant Colony System

Symbol	**Description**	**Comments**
NCmax	Max number of cycles of the algorithm	A Solution agent builds this number of solutions before reporting its best solution found.
α	Trail importance. The relative importance of trail intensity in the transition probability.	This was not explicitly used in Ant Colony System. We include it in this research with a value of 1.0 for consistency with related algorithms like Ant System [27].
Q	Global pheromone deposition constant.	Again, this was not explicitly used in ACS so we set it to a value of 1.0.
β	Visibility importance. The relative importance of visibility in the decision probability.	
q_0	Exploitation probability threshold. A parameter between 0 and 1 that determines the relative importance between exploration and exploitation in the decision rule.	
ρ_{local}	Decay constant used in local pheromone updates.	
ρ_{global}	Decay constant used in global pheromone updates.	
τ_0	The initial intensity of trail set on all edges at the start of a trial.	This is a parameter for the Place agents in the pheromone infrastructure.

4.4. *Protocols*

The use of agents and their associated messaging introduces the need for protocols. Preferably, these protocols should conform to published standards such as those of FIPA.

Since ants in the ACS algorithm interact only with the environment and not with one another, there is no need to add to the protocols provided with the Place agents and their pheromone infrastructure (Section 3.2). This simplicity of protocols is a clear advantage of nature-inspired MASs over MASs that rely on complicated market mechanisms and negotiations to self-organise.

4.5. *Messaging Cost*

The cost of messaging and its effect on MAS dynamics is a real engineering concern. This has been highlighted in related work with a MAS implementation of a genetic algorithm [13].

Table Table 4 and 5 present an estimation of the messaging needed in a MAS implementation of ACS for a problem of size $n = 50$ cities with $m = 20$ Solution agents. This is typical of the type of problem size on which ACS was first tested [14]. Table Table 4 shows the messaging required by the setting up of the pheromone infrastructure. Table 5 is one iteration of the main loop. Furthermore, there will occasionally be a further step where a Solution agent must backtrack to perform the 'global' update.

In the concrete example, setting up the pheromone infrastructure requires 5180 messages. The equivalent of one iteration of the main loop requires 8040 messages. ACS was

Table 4. Estimated Messaging for setting up the pheromone infrastructure

Stage	**Item**	**Protocol**	**Msgs per protocol**	**Total msgs**	**Value**
1	Each Place agent registers with the global DF so that other Place agents can find it.	Request Interaction	2	$2 \times n$	100
2	Each Place agent searches the global DF for its neighbouring Place agents	Request Interaction	2	$2 \times n$	100
3	Each Place agent registers its Place service with its neighbouring Place agents in the topology.	Request Interaction	2	$2 \times n \times (n-1)$	4900
4	Each Solution agent searches the global DF for its starting Place agent.	Request Interaction	2	$2 \times m$	40
5	Each Solution agent registers with its starting Place agent	Request Interaction	2	$2 \times m$	40

Table 5. Estimated Messaging for the equivalent of a single run of the 'main loop'

1	Each Solution agent queries its current Place agent for representation of the local pheromone infrastructure.	Query Interaction	2	$2 \times m \times n$	2000
2	Each Solution agent begins its move by deregistering from its current Place agent.	Request Interaction	2	$2 \times m \times n$	2000
3	Each Solution agent finishes its move by registering with its destination Place agent	Request Interaction	2	$2 \times m \times n$	2000
4	Each Solution agent performs a local pheromone update.	Request Interaction	2	$2 \times m \times n$	2000
5	Each Solution agent reports its tour and receives the length of the best tour.	Request Interaction	2	$2 \times m$	40

run for iterations of the order of 1200. For the given problem example, this would require approximately 9.6 million messages. This is clearly a heavy messaging load. Some of our exploratory studies of messaging in JADE [28], a popular MAS platform, on a desktop machine[2] yielded an average message rate of 550 per second. This would mean an equivalent MAS runtime of about 5 hours. . This is the discouraging reality of a direct MAS implementation of a nature-inspired algorithm when restricted to a single machine. We emphasise that we are investigate nature-inspired systems that will run on large numbers of machines at a scale that can harness the power of real ant colonies. There is no way to predict whether such an implementation will require either a smaller or greater number of 'runs' to achieve a similar solution quality to its algorithm equivalent. This justifies our motivation for studying implementations in truly parallel, asynchronous and decentralised environments.

It is also worth noting the effect of not using the standard Request Interaction protocol. If Solution agents did not require an agree return message then the number of messages per protocol would drop to 1 in steps 2-4 of Table 5. This reduces the corre-

[2] Intel Pentium 4, 2.80 GHz, 512 Mb RAM

sponding total message load to 6 million and the run time to 3 hours. There is a cost to conforming to standards.

4.6. Termination

A fundamental difference between an algorithm and a MAS is the idea of open-endedness. An algorithm typically steps through a number of instructions and either terminates or loops. Termination criteria include executing a certain number of iterations or reaching an acceptable level of solution quality. MASs, by contrast, are intended to be continually running systems. This poses the question of how we can know when our MAS has done an equivalent amount of work to the algorithm on which it was based.

This is easily solved in ACS by having each Solution Agent track the number of tours it has built and by giving the agent a maximum number of tours to build as an Agent solution parameter. Agents then know when to report their result. Agents deactivate after performing the required number of runs. This is, of course, a contrived situation so that a valid comparison can be made with the original ant algorithm. The ultimate goal is to realise a continuously running and adapting system. In such circumstances, it would be more realistic for agents to report their best solutions at some regular interval.

5. Related Work

There has been some related work on the transfer of nature-inspired algorithms to MAS-type platforms, infrastructures for nature-inspired MASs, and the issues when approximating parallelism in sequential and synchronous systems.

Several authors have investigated parallel implementations of Ant Colony algorithms [29,30]. However, these were in the vein of a master/slave type approach. A similar tactic is well established in the Evolutionary Computation community's use of the 'Island' genetic algorithm. Clearly, the master node in these implementations is a centralised component with global knowledge. This approach does not make sense in the context of the Peer-to-Peer type systems for which we are developing.

Smith *et al* [13] have investigated an implementation of a genetic algorithm using IBM's Aglets platform[3]. This approach is different from the vast majority of evolutionary computation (EC) work. Generally, EC algorithms are a centralised program storing individuals as data structures and imposing genetic operations to generate successive populations. In an agent system, there is no concept of a generation since agents interact asynchronously and in parallel. In Smith *et al*'s framework, agents exchange genetic information with messaging and can perform their own internal genetic operations on that information. The authors implemented equivalents of tournament selection and an elitist strategy. The authors tested their framework on the OneMax problem—a simple standard problem in the EC field. Counter to EC intuition, the elitist scheme converged much more slowly than the tournament scheme in real time due to the overhead of the greater number of message exchanges required. We have seen such a message overhead in our calculations on a MAS implementation of ACS (Section 4.5). This highlighted the general point that established conclusions of traditional EC research might not necessarily transfer to a MAS implementation.

[3] www.trl.ibm.com/aglets/

Brueckner's PhD dissertation proposed the original idea of *pheromone infrastructures* [20]. A decentralised MAS for manufacturing control was built on top of this infrastructure. Although there was a clear mention of a yellow pages service and certain ontological concepts, these were not related to any agent standards.

Cornforth has experimented with different update schemes in cellular automata (CA) [8,31]. Cellular automata can be considered as highly abstract MASs. A cellular automaton is a lattice of discrete sites such that each site has a finite set of possible values. Sites change in discrete time steps according to the same deterministic rules and these rules incorporate information from a limited neighbourhood of sites. Cornforth investigated 6 update schemes for site changes in a 1 dimensional cellular automaton. These schemes were: synchronous, random independent, random order, cyclic, clocked and self-sync. Each scheme produced dramatically different results in the cellular automaton. This emphasised that the update scheme in a sequential system is a factor in the system's performance and raises the question of which, if any, of the schemes is the best at approximating a truly parallel and asynchronous system.

6. Conclusions

We have argued the need for disciplined scientific and engineering research into how the successes of nature-inspired algorithms might be transferred to emerging computing environments such as the Grid, Peer-to-Peer systems, Autonomic Computing and the Semantic Web. Such research is of benefit to two communities. On the one hand, researchers of nature-inspired algorithms might see some performance benefits in the parallelism, asynchronicity, decentralisation and distribution of such environments. Furthermore, it gives these researchers a new and very relevant application domain for their algorithms. On the other hand, researchers looking for robust, decentralised, self-organising systems to operate in emerging computing environments should be interested in the promise of nature-inspired methods. These methods have been dramatically successful in their original application domains.

We have framed the idea of pheromone infrastructures within maturing agent standards, making specific reference to the protocols proposed by FIPA [15].

We then outlined a general methodology for transferring nature-inspired algorithms to emerging computing environments using pheromone infrastructures. This involved 6 steps that were illustrated by reference to a suggested MAS implementation of Ant Colony System. The Solution agents that operate on the pheromone infrastructure are identified. The global components of the original algorithm are removed or replaced. The parameters of the algorithm are categorised as solution parameters, problem parameters or environment parameters. We proposed a refinement of the solution parameter category to distinguish between agent and global solution parameters. The necessary protocols are chosen, preferable from suitable existing standard protocols. The cost of messaging is estimated. Some decision is made on when the MAS would have done an 'equivalent' amount of work to its algorithm counterpart. The estimate of the cost of messaging highlighted the price to be paid for the use of standard protocols for interoperability.

7. Current and Future Work

We are currently verifying our program code for our exploratory studies of the ACS algorithm. These experiments begin shortly. We expect these studies to reveal parallelism and asynchronicity as factors in the algorithm's performance and to provide quantitative data that will justify testing a fully-fledged MAS implementation.

Our longer-term goal is to develop a FIPA compliant set of Place agents that can support a pheromone infrastructure using standard protocols and ontologies. We would like to develop ant colony systems to run on this infrastructure and make well defined and statistically grounded assessments of the benefits and costs of such implementations. This will determine bounds on the usefulness of established results from the ant colony algorithm community when transferred to emerging computing environments.

References

[1] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts, USA, 1995.

[2] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[3] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence*. Oxford University Press, 1999.

[4] IBM. Autonomic Computing: IBMs Perspective on the State of Information Technology. Technical report, IBM Research, October 2001.

[5] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[6] Dieter Fensel, Wolfgang Wahlster, Henry Lieberman, and James Hendler, editors. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. The MIT Press, 2002.

[7] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The M.I.T. Press, Cambridge, Massachusetts, U.S.A.

[8] David Cornforth, David G. Green, and David Newth. Ordered asynchronous processes in multi-agent systems. *Physica D*, 2005.

[9] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, Massachusetts, 1995.

[10] H. Van Dyke Parunak, Sven A. Brueckner, Robert Matthews, and John Sauter. Pheromone Learning for Self-Organizing Agents. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 35(3):316– 326, 2005.

[11] Tucker Balch. Hierarchic Social Entropy: An Information Theoretic Measure of Robot Group Diversity. *Autonomous Robots*, 8:209–237, 2000.

[12] Krzysztof Chmiel, Dominik Tomiak, Maciej Gawinecki, Pawel Karczmarek, Michal Szymczak, and Marcin Paprzycki. Testing the Efficiency of JADE Agent Platform. In *Proceedings of the Third International Symposium on Parallel and Distributed Computing*, pages 49–56. IEEE Computer Society, 2004.

[13] Robert E. Smith, Claudio Bonacina, Paul Kearney, and Walter Merlat. Embodiment of Evolutionary Computation in General Agents. *Evolutionary Computation*, 8(4), 2000.

[14] Marco Dorigo and Luca Maria Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[15] The Foundation for Intelligent Physical Agents, 2005. Available from: `www.fipa.org/`.

[16] Marco Dorigo and Thomas Stutzle. *Ant Colony Optimization*. The MIT Press, Massachusetts, USA, 2004.

[17] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. John Wiley and Sons, New York, USA.

[18] Gerhard Reinelt. TSPLIB - A traveling salesman problem library. *ORSA Journal of Computing*, 3:376–384, 1991.

[19] Michael Luck, Peter McBurney, Onn Shehory, and Steve Willmott. Agent Technology Roadmap: Overview and Consultation Report. Technical report, AgentLink, December 2004.

[20] Sven Brueckner. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Phd, Humboldt University, 2000.

[21] H. Van Dyke Parunak, Sven A. Brueckner, Robert Matthews, and John Sauter. How to Calm Hyperactive Agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1092–1093. ACM Press, 2003.

[22] FIPA Agent Management Specification. FIPA Specification SC00023K, Foundation for Intelligent Physical Agents, 18 March 2004.

[23] FIPA. FIPA Request Interaction Protocol Specification. Technical report, Foundation for Intelligent Physical Agents, 2002.

[24] FIPA. FIPA Query Interaction Protocol Specification. Technical report, Foundation for Intelligent Physical Agents, 2002.

[25] H. Van Dyke Parunak and Sven A. Brueckner. Engineering Swarming Systems. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Kluwer, 2004.

[26] Sven A. Brueckner and H. Van Dyke Parunak. Information-driven Phase Changes in Multi-Agent Coordination. In *Poster in the Proceedings of the second international joint conference on Autonomous Agents and Multi-Agent Systems*, pages 950–951. ACM Press, New York, NY, USA, 2003.

[27] Marco Dorigo and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 26(1):1–13, 1996.

[28] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE: a FIPA2000 compliant agent development environment. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 216–217. ACM Press, New York, NY, USA, 2001.

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

[30] Thomas Stutzle. Parallelization Strategies for Ant Colony Optimization. In A. E. Eiben, Thomas Back, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, page 722. 1498 edition, 1998.

[31] David Cornforth, David G. Green, David Newth, and Michael Kirley. Do Artificial Ants March In Step? Ordered Asynchronous Processes and Modularity in Biological Systems. In Standish, Bedau, and Abbass, editors, *Proceedings of the Eighth International Conference on Artificial Life*, pages 28–32. 2002.