Of Ants and Elephants

Asaf Shiloni, Noa Agmon and Gal A. Kaminka
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
{shilona,segaln,galk}@cs.biu.ac.il

ABSTRACT

Investigations of multi-robot systems often make implicit assumptions concerning the computational capabilities of the robots. Despite the lack of explicit attention to the computational capabilities of robots, two *computational classes* of robots emerge as the focal points of recent research: Robot *Ants* and robot *Elephants*. Ants have poor memory and communication capabilities, but are able to communicate using pheromones, in effect turning their work area into a shared memory. By comparison, Elephants are computationally stronger, have large memory, and are equipped with strong sensing and communication capabilities. Unfortunately, not much is known about the relation between the capabilities of these models in terms of the tasks they can address. In this paper, we present formal models of both Ants and Elephant, and investigate if one dominates the other. We present two algorithms: AntEater, which allows Elephant robots to execute ant algorithms; and ElephantGun, which converts elephant algorithms—specified as Turing machines—into ant algorithms. By exploring the computational capabilities of these algorithms, we reach interesting preliminary results regarding the computational power of both models.

1. INTRODUCTION

Investigations of multi-robot systems, from a computational perspective, often focus on algorithms for specific tasks and applications. Such algorithms make explicit their assumptions concerning the sensing and actuation morphologies of the robots. However, more often than not, assumptions as to the computational capabilities of the robots are left implicit. They can be determined by examining the requirements of the algorithms, and the basic set of atomic actions they utilize.

Despite the lack of explicit attention to the computational capabilities of robots, two *computational classes* of robots emerge as the focal points of recent research: Robot *ants* and robot *elephants*.

Robot *ants*, somewhat popular in swarm robotics [6], are usually memory-less (or have severe memory limitations) and have relatively weak sensing abilities, if any [4, 18]. They usually have an ability to communicate through the environment, by leaving behind pheromones which essentially turn the environment into a shared memory. Robot ants have been shown to be able to carry out impressive robotic tasks, such as terrain coverage [4], area patrol [13], and foraging [12].

Robot *elephants* seem—by comparison—significantly stronger from a computational perspective. These have a large amount of memory (as large as needed, for instance, to hold a full map of the work area), and are equipped with strong sensing and communication machinery [2, 3].

However, not much is known about just what the limits of these models are and respectively what problems they can and cannot solve. Although the computational strength of Elephants were never questionable, the strength of Ants, which emerges from implicit communication and local interactions, is unknown. Thus, it is crucial to understand what are the Ants' limits of computational ability, and how do they stand relative to the Elephants' computational abilities.

In this paper, we present formal models of both Ants and Elephant in the environment of a grid, and investigate if one dominates the other, they are equivalent, or rather each has its own advantage over the other and thus, they are incomparable. We present two algorithms: AntEater, which allows Elephant robots to execute ant algorithms; and ElephantGun, which converts elephant algorithms—specified as Turing machines—into ant algorithms. By exploring the computational capabilities of these algorithms, we reach interesting preliminary results regarding the computational power of both models.

2. BACKGROUND

Ant robots are usually described as memoryless or more formally as finite state machines [17,18], i.e., having only a constant amount of internal memory, the size of which is independent of the problem size. Furthermore, they are described as having limited sensing capabilities [4] although in some of the work that considered simple mobile robots, the assumption is that their sensation abilities are all powerful in contrast to their weak computational abilities [1, 15]. Common to all previous work is the assumption that the Ants are not able to use conventional planning methods [19]. Lastly, what distinguishes the Ants from other simple mobile robots is the usage of pheromones to communicate with each other. These pheromones are basically pieces of information that can take any physical from such as chemicals [12], heat [11], markings [4] etc., and are sometimes evaporative [11, 17].

Bruckstein and Wagner have shown algorithms for area coverage by a swarm of Ants, using evaporative [17] and non-evaporative [8] markings. While some of these pheromones are laid by the robots themselves [8], others are a part of the given workspace [18]. They considered simple robots with a bounded amount of memory [18, 20] for their model of ant robots. Their works and additional works by Koenig et al. [5] produced upper bounds for the time it takes to complete a single or a repeated coverage by a swarm of Ants. However, non of the works above prove any concrete boundaries on the Ant model abilities, regardless of the specific task they engage in.

AAMAS 2008 Workshop on Formal Models and Methods for Multi-Robot Systems, May 13 2008, Estoril, Portugal.

Several papers investigated classes of semi-synchronous [16] and asynchronous [9,10] mobile robots that have all powerful sensing abilities, such as taking a snapshot of the world, in contrast to their weak memory functionality, no localization, and no sense of direction. Some interesting boundaries to these robots' abilities were found, yet we do not know if those limits stand when these robots are equipped with pheromones.

Unfortunately, model comparisons of robots had not been often discussed. There is, indeed, an extensive theory of computation, which includes a hierarchy of calculating machines from finite state machines to Turing machines [14], but lacks the ability to model machines with ongoing input. O'Kane and LaValle [7] produced a model for comparing the power of robot based on sensory abilities, but did not address computational and memory differences.

3. FORMAL ANTS AND ELEPHANTS

In this section, we provide formal definitions of the Ant and Elephant models used throughout our work (Section 3.1). We then compare between the computational power of the models, using two algorithms. The first (Section 3.2) allows a multiple Elephants to execute an algorithm for multiple ants. The second (Section 3.3) considers a specific reverse case, where a single Ant executes the algorithm of a single Elephant (under some restrictions).

3.1 Definitions

We define the Ant model as having a representative subset of properties from the models portrayed above. The capabilities of the Ant model are defined as follows:

Instruction set. Ants can:

- *Move* in all directions.
- *Sense* a limited radius around them
- *Read* and *write* arbitrary levels of multiple pheromone types.
- *Calculate* any set of values - bounded by their computational power.

Memory. Ants are oblivious in the sense their memory is constant, and is very limited compared to the size of the work area, allowing them to remember only a constant number of moves back.

Communications. Ants have an unlimited amount of pheromones, which are essentially pieces of limited information that can be left in space, read from, and be written to. The pheromones do not evaporate by themselves.

Localization. Ants have no means of localization.

Anonymity. Ants are anonymous, and cannot identify each other.

Homogeneity. Ants are homogenous; they have the same capabilities, and run the same algorithm.

Centralization. Ants work in a decentralized fashion.

We define the Elephant as obtaining the maximal abilities used by known models. However, in order to focus the comparison between the Ant and the Elephant model on issues rather than sensing (already handled by [7]), we assume that the Elephants have the same sensing capabilities as the Ants. The Elephant model's capabilities are defined below. We use emphasized text to denote differences with Ants:

Instruction set. They can move in all directions, sense the same limited radius around them as the Ants.

Memory. They have *unbounded* memory.

Communication. They have *reliable, instantaneous communications* to all others.

Localization. They *can localize themselves on a shared coordination system*.

Anonymity. Elephants have distinct identities, and all know of each other.

Homogeneity. They are homogenous in the sense that they have the same capabilities and run the same algorithm.

Centralization. They work in a decentralized fashion.

In order to investigate an equivalence between the two above models, we first define the measurement in which we compare the two models. We define dominance similar to the definition in [7], as follows:

DEFINITION 1. *let A_N and B_M be models of N and M mobile robots, respectively. Then:*

- *We say that A_N* dominates *B_M and notate it $A_N \trianglerighteq B_M$ if the computational ability of A_N are at least as powerful as those of B_M.*
- *We say that A_N is* equivalent *to B_M and notate it $A_N \equiv B_M$ if $A_N \trianglerighteq B_M$ and $B_M \trianglerighteq A_N$.*
- *We say that A_N is* incomparable *to B_M and notate it $A_N \bowtie B_M$ if $A_N \triangleright B_M$ and $B_M \triangleright A_N$.*

3.2 The Anteater

In this section, we show that N Elephants computationally dominate N Ants in the sense that N Elephants can simulate N Ants, where $N \geq 1$. In order to do that, we use an algorithm AntEater, that is executed by the Elephant, and simulates the behavior of the Ant. We prove that this algorithm transforms the Ants' algorithm, while keeping the characteristics of the original algorithm.

Algorithm 1 AntEater (Ant algorithm $\mathcal{A}$, list of robots R)

```
1: Initialize map M large enough to contain the work area, with
   current position from localization device.
2: Initialize pointer p to point to first instruction in A.
3: while A has not stopped do
4:    if step in p is to write pheromone level l in location (x, y)
      then
5:       write l in M(x, y)
6:    else if step in p is read pheromone level l from location
      (x, y) then
7:       read value l from M(x, y)
8:    else if Step in p is sense location (x, y) then
9:       Sense location (x, y) in space
10:   else if Step in p is calculate values (z_0, ..., z_n) then
11:      Simulate calculation of (z_0, ..., z_n)
12:   Broadcast M to all r ∈ R
13:   if step in p is move to (x, y) then
14:      Move to location (x, y) in space
15:      Update M with current position from localization device
16:   Set p to point to next instruction in A
```

The underlying idea in AntEater is to execute exactly the same movements as the ant algorithm $\mathcal{A}$, but distribute the shared memory created by the use of pheromones. Whenever $\mathcal{A}$ writes a pheromone value in the environment, AntEater writes in in the internal map kept by each elephant robot. And whenever $\mathcal{A}$ reads a pheromone value, the map is accessed in memory to retrieve the value stored. The elephant robots continuously communicate their map information to each other, thus making sure that their maps are identical—thus simulated pheromones written in one elephant robot's memory are readily available to all others for reading. We formally show this in Theorem 1.

THEOREM 1. *Procedure* AntEater*, if executed by an Elephant, achieves the properties of the ant algorithm it is given. Specifically, it guaranties the same* ***a.*** *Task completion,* ***b.*** *Time complexity, and* ***c.*** *Robustness to failures*

PROOF. *Task completion*: Assume that the solution for a given problem is a collection of paths and that this collection is achieved by the ant algorithm at a certain time. Therefore, since AntEater performs the same movements as the original ant algorithm $\mathcal{A}$ and simulates its calculations and pheromones in space, the Elephants will perform the same collection of paths and thus, will solve the given problem.
Time complexity: Let $\mathcal{O}(m)$ be the time complexity of the original ant algorithm $\mathcal{A}$, such that m is the number of steps taken by the Ant. Since in every step AntEater is going over exactly the step that would have been taken by $\mathcal{A}$, its time complexity will be $\mathcal{O}(mc)$, where c is the cost of broadcasting the robot's map and thus, is still a function of the number of robots. This can be achieved because AntEater does not perform any extra actions per step.
Robustness: AntEater preserves $\mathcal{A}$'s original robustness, for they eventually behave exactly the same. Lastly, as it emerges from line 3, AntEater assures termination in case the original ant algorithm itself terminates. □

We will use a coverage algorithm for ant robots called Mark-Ant-Walk, proposed by Osherovich et. al. [8], in order to exemplify the above theorem. The Mark-Ant-Walk algorithm is intended for one or more memoryless robots who use pheromones as indirect communication to perform a coverage task of an area. As advertised, Mark-Ant-Walk guaranties full coverage of a continuous area within $n\left\lceil\frac{d}{r}\right\rceil + 1$ steps, where n is the number of cells in the domain, d is the diameter of the domain, and r is the radius of the robot effector (although, the above algorithm does not know when to stop). Also, it promises immunity to noise and robustness to robot death: As long as at least one robot is alive, the area will be complete.

The Mark-Ant-Walk algorithm is given below (Algorithm 2). This algorithm is called continuously by each ant robots, with p given as the current location (whose coordinates are unknown to the robot). $R(r, 2r, p)$ denotes the robot's ability to sense pheromone level at its current position p and in a closed ring of radii r and $2r$ around p. $D(r, p)$ denotes the open disk radius r around the robot in which it can set the pheromone level, and $\sigma(a)$ denotes the pheromone level at point a:

Therefore, if we run AntEater with Mark-Ant-Walk as an input on Elephants with the same sensing capability yet with wireless communication instead of the ability to read and write pheromones, it will behave as follows: First, the Elephant will initialize a map with its own location on it and keep updating that map all along its run time from information it receives from other robots. This can be done since Elephants have enough memory to create such a map. Then, in each step the Elephant will move exactly as the Ant would have, use its effector just as the Ant would have, but instead of placing pheromones, it will update their value in its own map. Also, instead of sensing for pheromones it will fetch the pheromone level from its own map. Eventually, after completing a step, it will broadcast the changes it made to the map to all other robots, in case there are any.

Algorithm 2 Mark-Ant-Walk (current location p)

1: Let $x \leftarrow \text{argmin}_{q \in R(r,2r,p)}\, \sigma(q)$
2: **if** $\sigma(p) \leq \sigma(x)$ **then**
3: **for all** $u \in D(r, p)$ **do**
4: $\sigma(u) \leftarrow \sigma(x) + 1$ {We mark open disk of radius r around p}
5: Move to x

Again, we analyze AntEater's performance in the criteria of time complexity, complete coverage, and robustness.

- *Complete coverage*: Since the original ant algorithms guaranties complete coverage, and since the AntEater algorithm goes over each step originated by the ant algorithm, we can infer that an Elephant with same dimensions and effector capabilities will achieve complete overage as well.
- *Time complexity*: Assuming robots with same dimensions and effector capabilities, our AntEater algorithm guaranties the same time complexity, because the Elephant is moving only when the original Ant was supposed to move. Also, had the Mark-Ant-Walk algorithm have a stopping point, AntEater would have stopped exactly then. Since there are no additional actions, we can conclude that the AntEater guaranties complete coverage also within at most $O(c(n\left\lceil\frac{d}{r}\right\rceil + 1))$, where c is again the cost in time of broadcasting the map to all the other robots.
- *Robustness*: According to Osherovich et al. [8], the Mark-Ant-Walk algorithm is first and foremost an algorithm for a single Ant, thus if it succeeds proving robustness, then so does AntEater. The reason is that in the worst case scenario, when only one robot remains, it will cover the remaining area using the information gathered until then. Since in AntEater there are no additional actions that depend on other robots, it will not damage the ant algorithm's original robustness.

Moreover, we claim that not only does the AntEater preserve the original ant algorithm, but with some additions which are built specifically for a certain ant algorithm, we can improve its run time, efficiency, and/or robustness. As an example, the above Mark-Ant-Walk algorithm does not know when to stop. This is due to its bounded memory, which is not a function of the problem size and thus, cannot count steps to know to stop after $n\left\lceil\frac{d}{r}\right\rceil + 1$ steps, when it is assured that the area is covered. However, our Elephant's memory is not bounded and therefore, an addition to the algorithm of counting steps and a condition to stop after $n\left\lceil\frac{d}{r}\right\rceil + 1$ improves the original algorithm.

Indeed, we show (Theorem 2) that a group of N Elephants computationally dominates a group of N Ants:

THEOREM 2. *Let ANT_N and $ELEPHANT_N$ be the models portrayed in Subsection 3.1, where N is the number of robots, then $ELEPHANT_N \trianglerighteq ANT_N$ for $N \geq 1$.*

PROOF. Following Theorem 1, every algorithm executed by ants can be executed by elephants, while completing the same goal

in at most the same computational complexity and while maintaining the same characteristics. Therefore the computational ability of N elephants is at least as strong as the computational ability of N ants. □

3.3 Elephant Gun

We have established the fact that a group of N Elephants dominates a group of N Ants for $N \geq 1$. This is strongly based on the communication between the Elephants. Therefore the question that arises is whether a single Elephant still dominates a single Ant. In other words, after neutralizing the communication factor, is an Elephant computationally stronger than an Ant. We consider a subset of the general Elephant model, in which the Elephants have no localization abilities. That is, the Elephant's instruction set is the same as before, but in this model it cannot fetch its own position. In the following, we prove that, surprisingly, for this specific subset of Elephants the answer is that an Ant is equivalent to the Elephant model.

Most literature treats Ants as computationally equivalent to finite state machines. Similarly, the computational power of an Elephant is of a Turing machine. However, when we measure both models' computability power, we have to also take into consideration the space in which they operate in. Regarding the Elephants, there is no added computational power, since space can be modeled as additional tapes to a Turing machine, which is known to add no power. However, that is not necessarily true for the Ant. Recall that a finite state machine is equivalent to a Turing machine with no tape. Thus, the addition of space to the model is equivalent to the addition of a Turing machine tape, and with the assumption of an infinite amount of pheromones as the ability to read and write on that tape, we claim that the two models are of an equivalent power.

The intuition is that while an Ant has constant limited memory (making it equivalent to a finite state machine), it can use its own pheromones in space to give the Ant the external storage needed to have the strength of a Turing machine, given it has an infinite space to work in. Therefore, we will use the equivalent finite state machine to the Ant model we defined and similarly the Turing machine equivalent to the Elephant model.

However, the ant robot will need to move in space for two independent purposes: First, to simulate the Elephant's movements in space. And second, to utilize pheromones for storage. Thus it will need to remember if it is simulating movement or conducting a calculation.

To solve that, we will keep track of two Turing machine heads: The memory head, which moves during a calculation, and the movement head, which moves during a movement of the physical robot. Also, we will add information to the pheromones, which will point to the directions of each head. So, if instead of pheromones in the size of the original Elephant alphabet $|\Sigma|$, we will use pheromones in the size of $5 \times 5 \times |\Sigma| = 25|\Sigma|$ due to a pointer with the direction to memory head, a pointer with the direction to physical robot head, and the original alphabet. Each of the first two consists all directions and a symbol pointed that the Ant is on the pursued head. Note that we restrict ourselves here to movement on a grid, and thus all directions include the four basic movements on a grid: left, right, back, and forth (where the robot moves left and right without actually turning).

Thus, when the Ant simulates a calculation done by the Elephant, it will move left and right in space, acting as a a physical Turing machine. But, if interrupted by a movement of the robot it will first follow its own trail to find the physical robot head and once reaching the head, it will move the head to the desired location. Similarly, when needed to continue a calculation, the Ant will follow the trail to the memory head and once reaching the head, it will continue the calculation, changing the trail to point to the new head location.

However, in order to accomplish the above routine, the Ant will need to be careful not to create loops of pointers or rather not to follow old trails that lead nowhere. Therefore, when the Ant moves the memory head, it will both create a pointer to the memory head in every step, even if there is already a pointer there, and create a pointer to the movement head opposite of its own movement, except when there is already a pointer there. On the other hand, when the ant moves towards the memory head it will not change any pointers, but follow the pointers that already exist.

More formally, given an Elephant Turing machine Elephant such that:

$$\mathsf{Elephant} = (Q, \Sigma, b, \Gamma, \delta, s, F)$$

we will define the finite state machine ElephantGun as a Turing Machine without a tape, since both models are equivalent [14], such that:

$$\mathsf{ElephantGun} = (Q", \Sigma', b, \Gamma, \delta', s', F')$$

ElephantGun will have the states $Q" = Q \cup Q'$,$s' = s$,$F' = F$ where Q' is a set of additional states that will be specified ahead, and transitions $\delta' = \delta"$, where again $\delta"$ will be specified ahead, and in addition it will have an infinite amount of pheromones. Nevertheless, these pheromones will be from a finite number of types, such that the number of types $|\Sigma'|$ corresponds to $|\Gamma| \times 5 \times 5$. This is the size of the triplet mentioned above, where the first element represents the original alphabet Γ, the second points to the memory head, i.e. *left*, *right*, *back*, *forth*, or *here*, and the third points to the physical robot location with the same 5 options. Let us also define the operator $\overline{x}$ such that $\forall x \in L, R, B, F | \overline{L} = R, \overline{R} = L, \overline{B} = F, \overline{F} = B$ where $L = left$, $R = right$, $B = back$, and $F = forth$.

The new states Q', will be composed as follows for each $q \in Q$ and $Z \in L, R, B, F$:

- $q_{setmem(R)}$- an intermediate state to update the current slot as the memory head
- $q_{setmem(L)}$- an intermediate state to update the current slot as the memory head
- $q_{setloc(Z)}$- an intermediate state to update the current slot as the robot's location
- q_{find}- an intermediate state to find the robot's location
- $q_{find(Z)}$- an intermediate state to find the robot's location and move one slot to $Z \in L, R, B, F$

Also, for each $q \in Q, q' \in Q, a \in \Gamma, b \in \Gamma$:

- $q_{a,b,q',R}$- an intermediate state to find the memory head's location and perform the $(q, a) \rightarrow (q', b, R)$ transition
- $q_{a,b,q',L}$- an intermediate state to find the memory head's location and perform the $(q, a) \rightarrow (q', b, L)$ transition

In addition, we will replace the transitions δ by the new set of transitions $\delta"$ such that every transition from the form $(q, a) \rightarrow (q', b, R)$ will be replaced by the following transitions, where $y \in L, R, B, F$, $z \in L, R, B, F, H$, and $\sqcup$ is the empty pheromone:

- $(q, (a, y, z)) \rightarrow (q_{a,b,q',R}, (a, y, z), y)$ - for the case that the ant is not on the memory head

- $(q,(a,H,z)) \rightarrow (q'_{setmem(R)},(b,R,z),R)$ - for the case that the ant is on the memory head

Also, we will add the following transitions, where S stands for no movement:

- $(q_{a,b,q',R},(a,y,z)) \rightarrow (q_{a,b,q',R},(a,y,z),y)$ - continue searching the memory head in the pointed direction
- $(q_{a,b,q',R},(a,H,z)) \rightarrow (q'_{setmem(R)},(b,R,L),R)$ - found memory head, process transition, and move to the right
- $(q_{setmem(R)},(a,\sqcup,\sqcup)) \rightarrow (q,(a,H,L),S)$ - update memory head pointer to "here" and pointer to physical head
- $(q_{setmem(R)},(a,y,z)) \rightarrow (q,(a,H,z),S)$ - update memory head pointer to "here"

Likewise, every transition from the form $(q,a) \rightarrow (q',b,L)$ will be replaced by the following transitions:

- $(q,(a,y,z)) \rightarrow (q_{a,b,q',L},(a,y,z),y)$ - for the case that the ant is not on the memory head
- $(q,(a,H,z)) \rightarrow (q'_{setmem(L)},(a,L,z),L)$ - for the case that the ant is on the memory head

Also, we will add the following transitions:

- $(q_{a,b,q',L},(a,y,z)) \rightarrow (q_{a,b,q',L},(a,y,z),y)$ - continue searching the memory head in the pointed direction
- $(q_{a,b,q',L},(a,H,z)) \rightarrow (q'_{setmem(L)},(b,L,R),R)$ - found memory head, process transition, and move to the right
- $(q_{setmem(L)},(a,\sqcup,\sqcup)) \rightarrow (q,(a,H,R),S)$ - update memory head pointer to "here" and pointer to physical head
- $(q_{setmem(L)},(a,y,z)) \rightarrow (q,(a,H,z),S)$ - update memory head pointer to "here"

However, for every movement $Z \in L,R,B,F$ and every $z \in L,R,B,F$,$y \in L,R,B,F,H$ of the physical robot, the ant will have the following new transitions:

- $(q,(a,y,z)) \rightarrow (q_{find(Z)},(a,y,z),z)$- for the case that the ant is not on the physical robot head
- $(q,(a,y,H)) \rightarrow (q_{setloc(Z)},(a,y,Z),Z)$- for the case that the ant is on the physical robot head

Together with the following new transitions:

- $(q_{find(Z)},(a,y,z)) \rightarrow (q_{find(Z)},(a,y,z),z)$ - continue searching the physical robot head in the pointed direction
- $(q_{find(Z)},(a,y,H)) \rightarrow (q_{setloc(Z)},(a,y,Z),Z)$ - found physical robot head, update pointer, and move to the desired direction $Z \in L,R,B,F$
- $(q_{setloc(Z)},(a,\sqcup,\sqcup)) \rightarrow (q,(a,\overline{Z},H),S)$ - update physical robot head pointer to "here" and memory head pointer to where you came from
- $(q_{setloc(Z)},(a,y,z)) \rightarrow (q,(a,y,H),S)$ - update physical robot head pointer to "here"

And, lastly for any action or sensing need to be done while the ant is in its own physical location:

- $(q,(a,y,z)) \rightarrow (q_{find},(a,y,z),z)$ - for the case that the ant is not on the physical robot head
- $(q,(a,y,H)) \rightarrow (q,(a,y,H),S)$ - for the case that the ant is on the physical robot head

Together with the following new transitions:

- $(q_{find},(a,y,z)) \rightarrow (q_{find},(a,y,z),z)$ - continue searching the physical robot head in the pointed direction
- $(q_{find},(a,y,H)) \rightarrow (q,(a,y,H),S)$ - found physical robot head, the ant can sense or act

It is important to note that although it seems like the Ant's computational time will be huge relative to the Elephant's, it irrelevant in our case since we are proving computability. There may be different methods, which will be more optimal in the sense of time and/or space.

In order to be sure that the procedure of moving between the memory head and the physical robot head does not include any loops or dead ends, we prove the following two lemmas.

LEMMA 1. *No loop of pointers can be created by* ElephantGun.

PROOF. Assume, towards contradiction, that there is a set of pointers $(x_1,x_2,...,x_n)$ pointing to head a such that $\forall i \in 1..n, x_i \rightarrow x_{i+1} mod n$ (w.l.o.g), forming a loop. Thus, there exist no pointer x_i which points to the inside nor the outside of the loop. Also, none of x_i is the head itself, otherwise it would not be a loop. If the loop was created by head a itself, then the last pointer in the loop will replace the first one and will point towards head a and thus, breaking the loop. Otherwise, if the loop was created by head b, the first pointer in the loop will not be replaced and will still point towards head a. But head a is not a part of the loop, leading to a contradiction. □

LEMMA 2. *At every instance in* ElephantGun *there is a path of pointers between the two heads in each direction (not necessarily the same path).*

PROOF. Assume, towards contradiction, that there is no path of pointers from head a to head b (w.l.o.g). Thus, there exist at least one pointer in the path of pointers from a to b that does not lead to b. Since the two heads start from the same place and since there is no action of erasing, we can deduce that there was a path until the above pointer was replaced, and not by b. But, although both heads can add pointers, each of them can only replace its own pointers, leading to a contradiction. □

LEMMA 3. *The finite state machine* ElephantGun*, when equipped with infinite amount of pheromones and being ran on an infinite grid, is equivalent to the Turing machine* Elephant.

PROOF. It is easy to see that one can construct such a finite state machine. The new transitions, states, and alphabet, though each is larger then the original, they are still finite and thus, can be constructed to simulate the Turing machine. Furthermore, once created, the ElephantGun machine uses its infinite amount of pheromones as the Turing machine's alphabet and the grid it is located in as the Turing machine's tape to write in and read from. Lastly, the extra transitions added allow the ElephantGun machine to simulate both the Elephant's movements and calculations independently. □

THEOREM 3. *Let ANT_1 and $ELEPHANT_1$ be the models portrayed above correspondingly. Then, $ANT_1 \trianglerighteq ELEPHANT_1$.*

PROOF. Following Lemma 3, we can construct an Ant that simulates the Elephant model which has no localization. Therefore, each problem that can be solved by $ELEPHANT_1$ can be solved by ANT_1. Thus, $ANT_1 \trianglerighteq ELEPHANT_1$. □

When combining Theorem 2 and Theorem 3, we get the following conclusion for a single Ant and a single Elephant, which has no means of localization.

COROLLARY 1. *$ANT_1 \equiv ELEPHANT_1$ for the subset of the Elephant model which has no means of localization.*

PROOF. Since we have shown in 2 that $ELEPHANT_N \trianglerighteq ANT_N$ for $N \geq 1$, then $ELEPHANT_1 \trianglerighteq ANT_1$. Also, we have shown in 3 that $ANT_1 \trianglerighteq ELEPHANT_1$ for the subset of the Elephant model which has no means of localization. Therefore, $ANT_1 \equiv ELEPHANT_1$.

□

4. CONCLUSIONS

It was proposed that Ant robots can perform difficult computational tasks despite their weak computational abilities [18]. However, the computational limits of this model are still not known. We defined Elephant, as the most used model of robots with strong computational and sensing abilities and investigated the computational relationship between the two models.

We have shown that assuming reliable, instantaneous communication, Elephant robots can simulate any task done by Ant robots and therefore, are at least as computationally strong as Ants. This result is not surprising, as Elephants are by definition stronger. However, more surprisingly, we have also shown that given a large enough space and infinite amount of pheromones, a single Ant can simulate any task done by a single Elephant that has no localization abilities.

It is still not clear if a swarm of Ants can achieve the computational abilities of an Elephants herd. Furthermore, since in our proof the Ant uses its environment as memory, we cannot evaluate the Ant's computational abilities upon restriction on its environment or rather on its movement in the area.

The most straightforward future direction will be to wrap the equivalence between the two models, and that is to find out if N Ants can simulate N Elephants. Another direction will be to find out exactly how many Ants are needed to simulate an Elephant in minimal time and space overhead.

5. REFERENCES

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM J. Comput.*, 36(1):56–82, 2006.

[2] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. In *ICRA*, 2007.

[3] N. Hazon, F. Mieli, and G. A. Kaminka. Towards robust on-line multi-robot coverage. In *ICRA*, 2006.

[4] S. Koenig and Y. Liu. Terrain coverage with ant robots: a simulation study. In *AGENTS*, pages 600–607, New York, NY, USA, 2001. ACM.

[5] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):41–76, 2001.

[6] T. H. Labella, M. Dorigo, and J.-L. Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Transactions on Autonomous Adaptive Systems*, 1(1):4–25, 2006.

[7] J. M. O'Kane and S. M. LaValle. On comparing the power of robots. *International Journal of Robotics Research*, 27(1):5–23, January 2008.

[8] E. Osherovich, A. M. Bruckstein, and V. Yanovski. Covering a continuous domain by distributed, limited robots. In *ANTS Workshop*, pages 144–155, 2006.

[9] G. Prencipe. CORDA: Distributed coordination of a set of atonomous mobile robots. In *ERSADS*, pages 185–190, May 2001.

[10] G. Prencipe. Instantaneous actions vs. full asynchronicity: Controlling and coordinating a set of autonomous mobile robots. In *ICTCS*, pages 185–190, October 2001.

[11] R. Russell. Heat trails as short-lived navigational markers for mobile robots. In *ICRA*, volume 4, pages 3534–3539, 1997.

[12] R. Russell. Ant trails: An example for robots to follow? In *ICRA*, volume 4, pages 2698–2703, 1999.

[13] A. Sempe, F.; Drogoul. Adaptive patrol for a group of robots. In *IROS*, volume 3, pages 2865–2869, 2003.

[14] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[15] I. Suzuki and M. Yamashita. Agreement on a common x-y coordinate system by a group of mobile robots. In *In proceedings of the 1996 Dagstuhl Workshop on Intelligent Robots: Sensing, Modeling and Planning*, pages 305–321. World Scientific Press, 1997.

[16] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28:1347–1363, 1999.

[17] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation,*, 15(5):918–933, 1999.

[18] I. A. Wagner, Y. Altshuler, V. Yanovski, and A. M. Bruckstein. Cooperative cleaners: A study in ant robotics. *International Journal of Robotics Research*, 27(1):127–151, 2008.

[19] I. A. Wagner and A. M. Bruckstein. From ants to a(ge)nts: A special issue on ant-robotics (editorial). *Annals of Mathematics and Artificial Intelligence*, 31(1–4):1–5, 2001.

[20] V. Yanovski, I. A. Wagner, and A. M. Bruckstein. Vertex-ant-walk: A robust method for efficient exploration of faulty graphs. *Annals of Mathematics and Artificial Intelligence*, 31(1–4):99–112, 2001.