Of Robot 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 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 elephants, 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 conclusions regarding the computational power of both models.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Relations between models*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Theory, Algorithms

Keywords

Ant robotics, Computational Models, Multi-Robot Systems

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.

Cite as: Of Robot Ants and Elephants, Asaf Shiloni, Noa Agmon, Gal A. Kaminka, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 81 – 88

Despite the lack of explicit attention to the formal computational capabilities of robots, two *computational classes* of robots emerge as marking extreme points in recent research: Robot *ants*, which have restricted computational and communication capabilities, but can utilize pheromones to read/write messages in their environment, and robot *elephants*, which have strong computation and communication capabilities, but no pheromones. Other computational classes lie somewhere in between these extremes, e.g., many types of swarm robotics models [1, 12] which often share some computation restrictions with ants, but similarly to elephants, do not have pheromones. We focus here on ants and elephants.

Robot *ants* [18] are usually memory-less (or have severe memory limitations) and have relatively weak sensing abilities, if any [3, 17]. However, they can 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 [3, 17], and foraging [5, 11].

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, computation, and communication machinery. Robot elephants have similarly been shown to work in the same tasks as above (e.g., [2]).

However, while researchers of ant-robotics and elephant-robotics have tackled similar tasks, the actual computational capabilities and limitations of the two models remain an open questions. Empirical comparisons between solutions are difficult and rare, in part due to the different metrics and experiment designs in each community. Moreover, most robots in practice are more limited than the prototypical elephants described above, but less restricted than the robot ants; this makes distinguishing the underlying computational advantages and disadvantages of different robot models even more challenging.

In this paper, we seek to theoretically distinguish the two extreme models and their basic computability capabilities. We present formal models of both ants and elephants in a grid environment, 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 conclusions regarding the computational power of both models. We find that a group of elephant robots can easily simulate every ant algorithm run by a group of ant robots. Moreover, we find that a single ant robot can fully simulate a single

elephant robot, given infinite space. However, we show that there exist problems, which multiple elephants can solve and ants cannot.

2. BACKGROUND

Ant robots are usually described as memoryless or more formally as finite state machines [17], i.e., having only a constant amount of internal memory, the size of which is independent of the problem size. Furthermore, they typically have limited sensing capabilities [3, 18]. 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 form such as chemicals [11], heat [10], markings [3] etc.

Bruckstein and Wagner have shown algorithms for area coverage by a team of ants, using evaporative [16] 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 [17]. They considered simple robots with a bounded amount of memory [17, 19] for their model of ant robots. Their works and additional works by Koenig et al. [4] produced upper bounds for the time it takes to complete a single or a repeated coverage by a swarm of ants. However, none of the works above prove any concrete boundaries on the ant model abilities in general.

It is important to differentiate between ants and other types of swarms robots. All swarm robot models are decentralized and have very limited sensorial and computational abilities [1, 12]. However, ants have the usage of pheromones that can be placed and sensed, and by that transform their environment into a shared memory. Other swarm robot models exist which do not use pheromones, and yet do not have the unbounded communications of robot elephants [1, 12]. We do not investigate these in this paper.

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 [13]. 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.

Several papers investigated classes of semi-synchronous [14] and asynchronous [9] 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.

3. ELEPHANTS IMITATING ANTS

In this section, we provide formal definitions of the ant and elephant models used throughout our work (Section 3.1). We then begin to compare between the computational power of the models, using a first algorithm (Section 3.2) that allows multiple elephants to execute an algorithm for multiple ants. The next section (4) considers a reverse case, where ants execute the algorithm of elephants (under some restrictions).

3.1 Definitions

For simplicity, we will use a grid as the environment in which the ants and elephants interact. Nonetheless, we note that some of the proofs ahead are valid even on continuous domains.

We define the ant model as having a representative subset of properties from the models discussed 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 and Computation. Ants are oblivious in the sense that 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. From a computational point of view, ants are finite state machines.

Communications. Ants have an unlimited amount of pheromones, which are essentially traces that can be read from and be written to space. 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 all have the same capabilities, and run the same algorithm.

Centralization. Ants work in a decentralized fashion.

To focus the comparison between the ant and the elephant models 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. Elephants can move in all directions, sense the same limited radius around them as the ants.

Memory and Computation. Elephants have *unbounded* memory. From a computational point of view, elephants are Turing machines.

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

Localization. Elephants can typically *perfectly localize themselves on a shared coordinate system*. We call these *LF-Ants* (the *L* stands for localized). We also explore a variant of elephants that cannot localize within a global grid, called NF-Ants.

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

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

Centralization. Elephants work in a decentralized fashion.

The difference in computational ability between models is measured by the ability to solve different classes of problems. We define computational *dominance* similarly to the definition in [7]. Dominance is defined 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 is at least as powerful as those of B_M, i.e., if every problem solvable by B_M is also solvable by A_N.*

- *We say that* A_N strictly dominates B_M *and notate it* $A_N \rhd B_M$ *if* $A_N \trianglerighteq B_M$ *is true, and in addition there exists at least one problem solvable by* A_N*, but unsolvable by* 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* $B_M \trianglerighteq A_N$*.*

3.2 The Anteater

In this section, we show that N LF-Ants (elephants with localization) computationally dominate N ants in the sense that N LF-Ants can simulate N ants, where $N \geq 1$. To do this, we use an algorithm AntEater, that is executed by the LF-Ant, 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, map M)

```
1: Initialize pointer p to point to first instruction in A.
2: while A has not stopped do
3:    if step in p is to write pheromone level l in location (x, y) then
4:       write l in M(x, y)
5:    else if step in p is read pheromone level l from location (x, y) then
6:       read value l from M(x, y)
7:    else if Step in p is sense location (x, y) then
8:       Sense location (x, y) in space
9:    else if Step in p is calculate values (z_0, ..., z_n) then
10:      Simulate calculation of (z_0, ..., z_n)
11:   Broadcast M to all r ∈ R
12:   if step in p is move to (x, y) then
13:      Move to location (x, y) in space
14:      Update M with current position from localization device
15:   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. The elephant receives a map M, large enough to contain the work area, with current position from localization device. Whenever $\mathcal{A}$ writes a pheromone value in the environment, AntEater writes it in the internal map kept by each LF-Ant robot. And whenever $\mathcal{A}$ reads a pheromone value, the map is accessed in memory to retrieve the value stored. The LF-Ant robots continuously communicate their map information to each other, thus making sure that their maps are identical—therefore simulated pheromones written in one LF-Ant 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 LF-Ant, 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 LF-Ants 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 $mc = \mathcal{O}(m)$, 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 2, 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:

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

```
1: Let x ← argmin_{q∈R(r,2r,p)} σ(q)
2: if σ(p) ≤ σ(x) then
3:    for all u ∈ D(r, p) do
4:       σ(u) ← σ(x) + 1 {We mark open disk of radius r around p}
5: Move to x
```

Therefore, if we run AntEater with Mark-Ant-Walk as an input on LF-Ants with the same sensing capability yet with direct communication instead of the ability to read and write pheromones, it will behave as follows: First, the LF-Ant will initialize a map with its own location on it and keep updating that map all along its run time with information it receives from other robots. This can be done since LF-Ants have enough memory to create such a map. Then, in each step the LF-Ant 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 all other robots the changes it made to the map, in case there are any. Based on Theorem 1, we maintain the original upper bound of Mark-An-Walk. 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 LF-Ant'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 LF-Ants computationally dominates a group of N ants:

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

PROOF. Following Theorem 1, every algorithm executed by ants can be executed by LF-Ants, while completing the same goal in at most the same computational complexity and while maintaining the same characteristics. Therefore the computational ability of N LF-Ants is at least as strong as the computational ability of N ants. □

3.3 LF-Ants and NF-Ants

The LF-Ants above use a shared coordinated system thanks to their localization devices. This localization within a shared coordinate system is a key component in their dominance over ants. However, localization is not a trivial capability.

We therefore introduce the NF-Ant, which is a weaker version of the LF-Ant model. The NF-Ant model is identical to the LF-Ant model except it does not have a localization and thus, two or more NF-Ants do not necessarily share the same coordinate system.

Hence, we provide a way for NF-Ants to simulate ants, of course, without localizing themselves on a shared coordinated system. This is done by an algorithm called AntEaterSpiral, which is compounded from the following NFantSpiral algorithm and the previous AntEater algorithm.

Algorithm 3 NFantSpiral (list of robots R, map M)

```
1: if ID == 0 then
2:    Set current location o as point of origin on M
3:    r ← 1 { The number of robots traveling in the group }
4:    while r < |R| − 1 do
5:       Move within a clockwise spiral { recording movements }
6:       if There exist a robot r_i in point p then
7:          Send robot r_i point −p
8:          r ← r + 1
9:    Return to o
10: else
11:    if received position p then
12:       Set p as point of origin on M
```

The main idea in the above NFantSpiral algorithm is for one robot to search for all other robots, update the new origin of their coordinate systems as it own origin, and then return to it own starting point.

To do so, all robots will receive a map large enough to contain the work area and then will elect the robot with the lowest ID as the leader (zero ID w.l.o.g). The leader will then start moving in a spiral around its original position until it finds another robot. It will then send the difference between its own origin and the robot position as the robot's new origin. The leader will continue searching for other robots and will stop only if the group size equals the size of the list of robots given as input, when it will then return to its own origin. Lastly, in order to show that NF-Ants can simulate ants, they run the following AntEaterSpiral algorithm that first calls NFantSpiral and then calls AntEater as before, except now the map's origin is not given, but decided upon spiraling.

Therefore, we can show that a group of N NF-Ants computationally dominate a group of N ants:

THEOREM 3. *Let ANT_N and $NF - ANT_N$ be the models discussed above, where N is the number of robots, then $NF - ANT_N \trianglerighteq ANT_N$ for $N \geq 1$.*

Algorithm 4 AntEaterSpiral (Ant algorithm $\mathcal{A}$, list of robots R, map M)

```
1: Run NFantSpiral on (R, M) to assure a common coordinate system
2: Run AntEater on (A, R, M)
```

PROOF. By applying AntEaterSpiral, a group of N NF-Ants first agree upon the origin of their map, using NFantSpiral. From that moment on, they are equivalent to a group of N LF-Ants, which we have shown in theorem 2 to simulate any Ant algorithm they are given. Thus, by running AntEater the group of N NF-Ants simulates the group of N ants, and therefore $NF - ANT_N \trianglerighteq ANT_N$. □

4. ANTS SIMULATING ELEPHANTS

So, we know that a group of N LF-Ants that are communicating explicitly among themselves dominate a group of N ants, But, even one LF-Ant dominates one Ant. That raises the question, whether one Ant dominates one LF-Ant (Section 4.1), and whether N ants dominate N LF-Ants (Sections 4.2 and 4.3).

4.1 A single ant

We have established the fact that a group of N LF-Ants dominates a group of N ants for $N \geq 1$. This is strongly based on the communication between the LF-Ants. Therefore the question that arises is whether a single LF-Ant still dominates a single Ant. In other words, after neutralizing the communication factor, is an LF-Ant computationally stronger than an Ant.

We consider the subset of the general LF-Ant model - the NF-Ant model, in which the LF-Ants have no localization abilities. In the following, we prove that, surprisingly, for NF-Ants the answer is that one Ant is equivalent to one NF-Ant.

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. Hence, in the proof we use a finite state machine and a Turing machine as the Ant's and NF-Ant's computational mechanisms respectively.

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

To solve that, we will keep track of two virtual 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: left, right, back, forth, and here. So, instead of pheromones in the size of the original NF-Ant alphabet Γ, we will use pheromones in the size of $5 \times 5 \times |\Gamma| = 25|\Gamma|$ 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 pheromones can take the form of all four basic directions as well as a symbol for pointing out that the ant is located exactly where the head is. Note that we restrict ourselves here to movements 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 NF-Ant, it will move in space, acting as a a physical Turing machine. But, if

interrupted by a movement of the NF-Ant 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, an NF-Ant is a Turing machine Elephant such that:

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

where Q is the set of states, Σ is the input's alphabet, b is the blank symbol, Γ is the tape's alphabet, δ is the transition function, s is the starting state, and F is the set of accepting states. We will define the finite state machine ElephantGun as a Turing machine without a tape (since both models are equivalent [13]), such that:

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

ElephantGun will have the states $Q' = Q \cup Q$",$s' = s$,$F' = F$ where Q" is a set of additional states that are specified below, and transitions δ' that are also specified below In addition it will have an infinite amount of pheromones. Nevertheless, these pheromones will be from a finite number of types, such that the symbols in Γ' corresponds to all of the $|\Gamma| \times 5 \times 5$ combinations of the triplet of pheromones 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 $\forall x \in \{L, R, B, F, H\}, \overline{L} = R, \overline{R} = L, \overline{B} = F, \overline{F} = B, \overline{H} = H$ where $L = left$, $R = right$, $B = back$, $F = forth$, and $H = here$. Lastly, the input alphabet stays the same and hence, $\Sigma' = \Sigma$.

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

- $q_{setmem(R)}$- 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 $Z \in \{L, R, B, F, H\}$

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 δ' such that every transition from the form $(q, a) \rightarrow (q", b, R)$ will be replaced by the following transitions, where $y \in \{L, R, B, F, H\}$, $z \in \{L, R, B, F, H\}$, 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 in 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 NF-Ant's, it is 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 $\forall i, i = 1..n, x_i \rightarrow x_{i+1} \ mod \ n$ (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, break 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. □

The former two lemmas therefore assure us that such consistent movement between the memory head and physical head can be achieved and thus, we can proceed towards constructing such simulation of the NF-Ant by the ant. Thus, we reach the following important conclusions: The first is that the ant's finite state machine with the assistance of the workspace is equivalent to the NF-Ant's Turing machine. This implies that the ant model is as least as strong as the NF-Ant when involving only one robot (lemma 3).

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 *$(NF - Ant)$.*

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, 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 4. *Let ANT_1 and $NF - ANT_1$ be the models portrayed above correspondingly. Then, $ANT_1 \trianglerighteq NF - ANT_1$.*

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

When combining Theorem 2 and Theorem 4, we get the following conclusion for a single ant and a single NF-Ant.

COROLLARY 1. $ANT_1 \equiv NF - ANT_1$.

PROOF. Since we have shown in Theorem 2 that $NF - ANT_N \trianglerighteq ANT_N$ for $N \geq 1$, then $NF - ANT_1 \trianglerighteq ANT_1$. Also, we have shown in Theorem 4 that $ANT_1 \trianglerighteq NF - ANT_1$. Therefore, $ANT_1 \equiv NF - ANT_1$. □

4.2 Multiple ants

When investigating the problems involving N robots it seems like we could easily find ones which are solvable by a group of N LF-Ants, but unsolvable by a group of N ants. However, looking more closely, we find that many of these problems are indeed solvable by a group of N ants, usually at the price of additional time complexity. For instance, let the problem Meeting be defined as follows.

DEFINITION 2. Meeting *Given two mobile robots r_1 and r_2, which are positioned on a 2-dimensional grid in positions (x_1, y_1) and (x_2, y_2) respectively, we say that an algorithm $\mathcal{A}$ running on both robots succeeds if and only if for every pair of points (x_1, y_1) and (x_2, y_2), r_1 and r_2 meet within a finite time.*

It is easy to construct an algorithm for two LF-Ants that can solve Meeting like the following.

Algorithm 5 Manhattan (robot r)

1: broadcast initial position (x_i, y_i)
2: receive other robot's position (x_{1-i}, y_{1-i})
3: calculate mid point p of the manhattan distance between (x_i, y_i) and (x_{1-i}, y_{1-i})
4: move towards p

Since LF-Ants can communicate directly, the first two steps are possible and so is the rest of the algorithm. The time complexity for Manhattan is exactly the midpoint of the manhattan distance $\left\lceil \frac{|y_{1-i}-y_i|+|x_{1-i}-x_i|}{2} \right\rceil$, which is optimal in a grid. We will denote that time complexity as $O(d)$ such that $d = \left\lceil \frac{|y_{1-i}-y_i|+|x_{1-i}-x_i|}{2} \right\rceil$. Moreover, we have shown (Algorithm 3) that an NF-Ant can also solve Meeting.

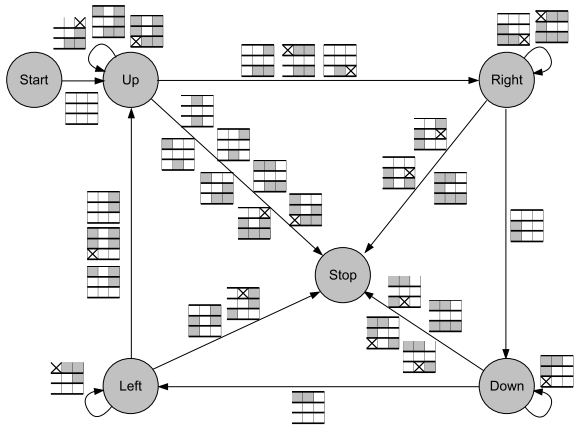

Figure 1: Meeting finite state machine. Current position is on the middle square. Gray squares contain a pheromone, white squares do not, and "X" squares signify a "don't care". The FSM halts when another ant is on one of the squares.

Nevertheless, there is also an ant algorithm which solves Meeting, like the following Spiral algorithm, while using a state machine to create a spiral and to meet the other ant (see Figure 1).

Algorithm 6 Spiral

```
1: walk in a spiral according to the state machine in Figure 1
2: if Other robot is within radius then
3:     Move towards other robot
```

In this algorithm, the ant decides upon the next step according to a finite state machine that considers the pheromone's locations within the eight squares surrounding it. In each step, the ant places a pheromone in its own location and then move according to the FSM. Whenever it senses another ant within its sensory radius, it moves towards the other ant.

As we can see, in the worst case both ants will meet after creating a spiral with a $\left\lceil \frac{|y_{1-i}-y_i|+|x_{1-i}-x_i|}{2} \right\rceil$ radius. This spiral has a total distance of $\left\lceil \frac{(|y_{1-i}-y_i|+|x_{1-i}-x_i|)^2}{4} \right\rceil$ and thus, $O(d^2)$ is its time complexity. Note that this time complexity is only quadratic relative to the time complexity of Manhattan.

So, are there any problems which cannot be solved by a group of N ants?

4.3 Tragedy of the Common Ant

In this section we will prove by a counter example that a group of N ants cannot fully simulate a group of N NF-Ants. As a result, since N LF-Ants dominate N NF-Ants, then N LF-Ants dominate N ants. In order to do this, we will define the following problem we call LimitedKServer.

DEFINITION 3. LimitedKServer *Let $R = r_1, ..., r_N$ be a set of mobile robots with sensing radiuses of $\frac{M^2}{2N}$ each, all are positioned on a finite $M \times M$ grid such that all sensing radiuses are disjoint and their union covers the whole grid. Let $C = c_1, ..., c_x$ be a set of calls and y be a positive integer such that $y \leq x < 2yM$ where y is known, but neither C nor x is known. Assume that within a finite period of time t, x calls are made such that no two calls are made in parallel. Assume that every robot $r_i \in R$ can answer a call immediately only within its sensing radius and that each call lasts for one time cycle only. We say that an algorithm $\mathcal{A}$ succeeds if and only if for every sequence of x calls, $\mathcal{A}$ answers exactly y. Otherwise, $\mathcal{A}$ fails.*

Note that the above problem is a variant of a physical k-server [6] problem where there are a finite number of calls and the servers need to collectively answer only an exact smaller amount of these calls. This problem represents an abstraction of problems related to the tragedy of the commons such as overfishing [15]. For our purpose of showing that NF-Ants dominate ants we will first show that there exist an algorithm called Fisherman for NF-Ants which solves LimitedKServer. Following that, we will prove that there exist no algorithm for ants which solves that same problem.

Algorithm 7 Fisherman (list of robots R, call limit y)

```
1: initialize local value calls to zero
2: while calls < y do
3:     if a message 'CALL' has been received then
4:         increment calls
5:         if calls = y then
6:             break
7:     else if there exist a call c within the robot's call answering
       radius then
8:         broadcast 'CALL' to all r ∈ R
9:         increment calls
10:        answer call
```

THEOREM 5. *Algorithm* Fisherman *solves* LimitedKServer *when running N NF-Ants.*

PROOF. Let X be the finite set of calls. Since the N NF-Ants cover the $M \times M$ entirely, there exist no call that is overlooked by all of the NF-Ants. Also, since each NF-Ant reigns over a disjoint territory, no call is being answered by two NF-Ants. In addition, since an NF-Ant is only occupied for one cycle per call and no two calls are made in parallel, there is no situation in which an NF-Ant is occupied and cannot answer a new call. Thus, after y calls the herd of NF-Ants have answered exactly y calls and therefore, each NF-Ant will break from the while loop upon receiving the y-th 'CALL' message. Lastly, there is no usage of localization along the messages transferred by the NF-Ants and thus, the NF-Ants will have no problem processing the algorithm. □

Therefore, we can infer the following corollary directly.

COROLLARY 2. LimitedKServer *can be solved by N NF-Ants.*

Now, if we inspect the ant behavior within LimitedKServer we can reach the following lemmas:

LEMMA 4. *There is no ant algorithm which solves* LimitedKServer *when running N ants and at least one ant moves from its initial position.*

PROOF. Assume $y = x$. Therefore, if an ant algorithm involves an ant moving from its initial position p at time t, there can always exist a sequence with a call at time t at position p which will be missed and thus, the algorithm will fail. □

LEMMA 5. *Every ant algorithm $\mathcal{A}$ solving* LimitedKServer *requires at least one Ant to move from its position during its execution.*

PROOF. Since ants do not have any direct communication, the only way they can propagate information among themselves is by leaving pheromones over the grid or moving towards each other, both involve at least one ant moving. Now, suppose that there is an ant algorithm $\mathcal{A}$, which attempts to solve LimitedKServer without any movement by any of the ants. Then, there is no way an ant could know whether or not to answer the $(y+1)$-th call for it cannot know that there were y calls before. □

Based on Lemmas 4 and 5 we can now conclude:

THEOREM 6. *There is no ant algorithm which solves* LimitedKServer *when running N ants.*

PROOF. Recall that we assume that the ant's sensing radius is identical to the LF-Ant's sensing radius. Since ants do not have explicit communication beyond their sensing radius, we can extract from the above lemma that no information can be transferred from one ant to another when trying to solve LimitedKServer. So, in order to solve LimitedKServer, any ant algorithm would need to know which calls to answer in advance, and since this information is not available we conclude that there is no ant algorithm which solves LimitedKServer when running N ants. □

Tragedy on an Infinite Grid Now, we wish to investigate if an infinite working space would assist ants to solve all the problems LF-Ants can. Unfortunately, we discover that the computational gap between the two models still holds. We show this by a small refinement to the former LimitedKServer problem. We define InfiniteKServer to be identical to LimitedKServer only that this time the robots' working space is infinite while all calls are still made in the predefined $M \times M$ area. We claim that the Fisherman algorithm solves InfiniteKServer when running N NF-Ants, since Fisherman ignores the additional infinite working space and behaves exactly like in theorem 5.

However, similarly to Lemma 4, there is no ant algorithm which solves InfiniteKServer when running N ants and at least one ant moves from its initial position. This is because every ant algorithm $\mathcal{A}$ solving LimitedKServer requires at least one ant to move from its position during its execution, as we have seen in Lemma 5. Therefore, $NF - ANT_N \rhd ANT_N$, regardless of the size of the working space.

In other words, we reach the conclusion that not the memory deficiency, but the lack of instant communication is what ultimately differs the ants from the NF-Ants. Thus, even an infinite workspace is not sufficient for ants to simulate NF-Ants in certain problems.

5. CONCLUSIONS

It was proposed that ant robots can perform difficult computational tasks despite their weak computational abilities [17]. However, the computational limits of this model are still not known. We defined elephant, as the most used model of robots with strong computational, sensing, and communication 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. Unfortunately, this stops with multiple ants or elephants, as we show that there exist some problems that can be solved by N elephants, but not with N ants.

Now that the basic computability differences between these models are known, we hope to extend the analysis to more realistic robots, which for the most part are in-between the two computational extremes discussed above. We also seek to combine the analysis with sensing models (e.g., as in [7]), determining complexity tradeoffs for the subset of problems that are solvable by both models, or finding out exactly how many ants are needed to simulate an elephant in minimal time and space overhead.

Acknowledgements. We thank Alfred "Freddy" Bruckstein, Israel Wagner, and Manuela Veloso for useful discussions. This research was supported in part by ISF grant #1357/07.

6. REFERENCES

[1] E. Şahin. Swarm robotics: From sources of inspiration to domains of application. In *Swarm Robotics: SAB 2004 International Workshop*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer, 2005.

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

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

[4] 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.

[5] 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.

[6] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[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 autonomous mobile robots. In *ERSADS*, pages 185–190, May 2001.

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

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

[12] A. J. Sharkey. Robots, insects and swarm intelligence. *Artificial Intelligence Review*, 26(4):255–268, 2006.

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

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

[15] R. M. Turner. The tragedy of the commons and distributed AI systems. In *in Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 379–390, 1993.

[16] 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.

[17] 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.

[18] 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.

[19] 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.