Dynamic Task Assignment in Robot Swarms

James McLurkin
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Lab
Cambridge, MA 02139 USA
Email: jamesm@csail.mit.edu

Daniel Yamins
Harvard University
Department of Mathematics
Cambridge, MA 02138 USA
Email: yamins@fas.harvard.edu

***Abstract*— A large group of robots will often be partitioned into subgroups, each subgroup performing a different task. This paper presents four distributed algorithms for assigning swarms of homogenous robots to subgroups to meet a specified global task distribution. Algorithm Random-Choice selects tasks randomly, but runs in constant time. Algorithm Extreme-Comm compiles a complete inventory of all the robots on every robot, runs quickly, but uses a great deal of communication. The Card-Dealer's algorithm assigns tasks to individual robots sequentially, using minimal communications but a great deal of time. The Tree-Recolor algorithm is a compromise between Extreme-Comm and Card-Dealer's, balancing communications use and running time. The three deterministic algorithms drive the system towards the desired assignment of subtasks with high accuracy. We implement the algorithms on a group of 25 iRobot SwarmBots, and collect and analyze performance data.**

I. Introduction

The dynamic assignment of tasks in multi-robot systems has many applications. When ants forage, different workers must simultaneously scout food sources, lay trails, and transport prey back to the nest [1]–[3]. In a search-and-rescue mission of 40 robots, a preferred task assignment might be 30 robots to explore the environment, 2 to mark resources, and 8 to maintain a communications network [4], [6], [7]. This distribution, $(75\%, 5\%, 20\%)$, should be maintained as reinforcements arrive, or robots leave to recharge their batteries. When an entire subgroup is removed, the remaining robots should reassign themselves to preserve the global distribution. The system should also be responsive to new global distribution inputs from a user or a task-allocation algorithm.

The task allocation problem in multi-robot systems therefore has two components: 1) deciding how to divide a group of robots into subgroups, with each subgroup performing a separate task; and 2) achieving the desired subgroup assignment in a distributed system. The first component consists of determining an optimal number of subgroups into which to divide the robot population, and an optimal distribution for the relative sizes of these subgroups. The second consists in having robots determine, through local interactions, which group to join to achieve the desired global task distribution. This paper addresses the second component, assuming a "black-box" solution to the first. We describe four distributed dynamic task assignment algorithms, analyze them mathematically, and implement them in a swarm of 25 autonomous mobile robots.

All robots are given the target distribution by a centralized source in the form of a vector of normalized relative subgroup sizes, e.g. the tuple $(1/6, 1/3, 1/2)$ for a system with three subgroups. The algorithms are designed to work on a swarm of homogeneous robots, so any robot can perform any task. The robots share limited information about their state with their immediate neighbors and switch their tasks when needed. The four algorithms span a spectrum of trade-offs between temporal efficiency, communications complexity, and accuracy.

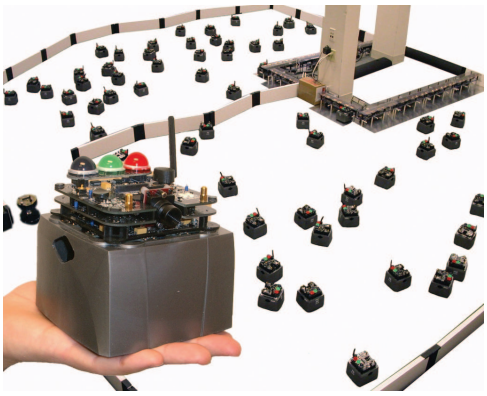

Fig. 1. The iRobot Swarm is comprised of over 100 SwarmBots, charging stations, and navigational beacons. Each SwarmBot measures 5" on a side and has a suite of sensors, communications hardware, and human interface devices. Hands-free operation is essential with this number of robots, and the Swarm supports remote downloading and autonomous charging.

Algorithm **Random-Choice** is the simplest solution: robots choose a given task with probability equal to the relative size of that task subgroup in the target distribution. This algorithm requires no communication and completes immediately. However, there is a high probability that it will fail to achieve the target distribution in small to medium-sized swarms (10-50 robots). Algorithm **Extreme-Comm** is at the opposite end of the spectrum. Each robot uses local interactions to build a complete list of all other robots in the swarm, and uses this list to determine its task. Though fast and accurate, the algorithm requires a large amount of inter-robot communications. The **Card-Dealer's** algorithm uses minimal communications by sequentializing the task-assignment problem into a series of stages; its flaw, thereby, is its long execution time. Finally, the **Tree-Recolor** algorithm is a compromise between **Extreme-Comm** and **Card-Dealer's**, balancing communications use

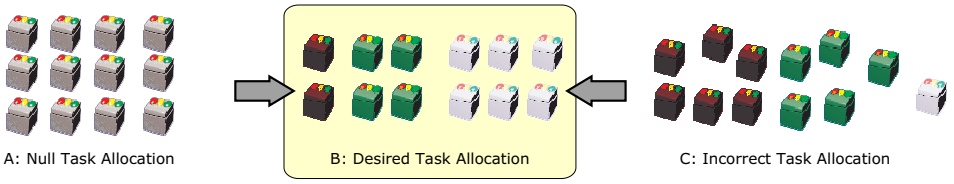

Fig. 2. Illustration of task assignment for a group of 12 robots. The desired task assignment is given in percentages of red(dark grey), green(medium grey), and blue(light grey), $(1/6, 1/3, 1/2)$. In the left panel, none of the robots have selected a task. In the right panel, the task assignment is incorrect. A task assignment algorithm will drive the system to the distribution illustrated in the center panel with 2 reds, 4 greens, and 6 blues. Note that robots with similar tasks are placed near each other for illustrative purposes - this is neither a result of nor a requirement for algorithm execution.

and running time by using gradient trees. All of the algorithms are robust to the removal or addition of robots at any time, which is critical for implementation in any real multi-robot system.

We implemented the algorithms on a group of iRobot SwarmBots (Figure 1). The algorithms generally performed to predicted levels of communications usage, temporal efficiency, and assignment accuracy, but problems arose due to bandwidth limitations in the communication hardware of the SwarmBot system.

A. Related Work

The first stage of the task allocation problem – the determination of the optimal group structure and size distribution for a variety of distribution problems – has been studied extensively by Mataric and others [8], [9], [11]. The problem we study here, that of achieving a target distribution, has been addressed using probabilistic threshold models [12], [3], [13], [14]. But these techniques do not provide precise and variable control over global task distribution, and provide no guarantee for success in small groups. Also, when the number of tasks exceeds a small number (often 2 or 3), probabilistic threshold methods often have multiple steady states which can trap systems away from a desired task distribution [12]. What is needed is an algorithm which deterministically drives the task distribution to the desired global proportions, independent of other system dynamics.

B. Problem Description

We model a swarm of n robots $\{x_{ID} | 1 \leq ID \leq MaxRobots\}$ for some large constant $MaxRobots$. Each robot has a unique hard-coded identification number, a processor, and enough memory to store state proportional to n. Each robot has a communications system that enables it to communicate with its *neighbors*, robots that are within a circle of some fixed radius determined by the communications hardware. These communication neighborhoods define a directed graph $\mathcal{G}$ whose nodes correspond to robots and whose edges correspond to communications links between neighboring robots. We require this graph to be connected at all times. Each robot selects an integer task between 1 and m, and a robot can change their task assignment at will.

Every robot has a local execution clock which regulates algorithm execution and message broadcast, and every robot's execution clock increments at the same rate. This rate is much slower than the robot's actual processor clock, so events can be modeled as occurring instantaneously on each "tick" of the execution clock. Therefore, algorithms will be executed at the same rate on each robot, even though no two robots are likely to execute the same instruction simultaneously. Therefore, the swarm as a whole has a well-defined *execution cycle*, the period during which all robots have collected messages and executed an algorithm exactly once, though different robots will not be at the same point in the cycle at the same time. We can therefore model the swarm as a *synchronous dynamic network*; i.e., as if there were one global clock to which all events synchronize, even when robots enter or depart. The algorithms described in this paper do not depend on global synchronization, only on the fact that duration of the execution cycle is the same for all robots.

At any given moment t, the system possesses a task distribution vector $dv(t) = (n_1/n, \ldots, n_m/n)$ in which n_i is the number of robots in task-group i.[1] The global task assignment problem is to find a distributed algorithm such that given a target distribution vector $\mathbf{p} = (p_1, \ldots, p_m)$, the distribution $dv(t)$ converges as close to $\mathbf{p}$ as possible. That is,

$$\lim_{t \to \infty} dv(t) = argmin_v \{||v - \mathbf{p}||_2 \quad \text{for} |v| = n\}.$$

Additionally, we want the algorithm to be insensitive to arbitrary changes in the total number of robots n, or the topology of graph $\mathcal{G}$, as long as the network remains connected.

II. Four Algorithms

A. The *Random-Choice* Algorithm

In the **Random-Choice** algorithm each robot draws a random number $x \in [0, 1]$ uniformly and *bins* it with respect to $\mathbf{p}$: if $x \in [p_1 + \ldots + p_{i-1},\ p_1 + \ldots + p_i]$ then the robot enters task-group i. Its running time is $O(1)$, depending only on the time it takes to choose and bin x. No inter-robot communication is required.

Let X_i be the random variable defined by the relative percentage of robots in task i after n robots have chosen as above, so that $X = (X_1, \ldots, X_m)$ is a random m-dimensional column vector whose entries sum to 1. The variable $n \cdot X$ is distributed according to the $(m-1)$-dimensional multinomial distribution with sample size n and mean $\mathbf{E}[X] = n \cdot \mathbf{p}$. Hence, X itself is distributed according to a $(m-1)$-dimensional

[1] Since $n = \sum_{i=1}^{C} n_i$, the elements of $\mathbf{p}$ sum to unity.

multinomial distribution in which the domain has been normalized by n, the number of robots. X has expected value $\mathbf{E}[X] = \mathbf{p}$ and covariance matrix

$$\Sigma_{\mathbf{p}} = \mathbf{E}[(X - \mathbf{E}[X])(X - \mathbf{E}[X])^T] = \frac{1}{n}(\mathbf{D}_{\mathbf{p}} - \mathbf{p} \cdot \mathbf{p}^T),$$

where $\mathbf{p}^T$ is the transpose of $\mathbf{p}$, a m-dimensional row-vector, and $\mathbf{D}_{\mathbf{p}}$ is the diagonal matrix whose i-th diagonal element is p_i. Hence, accuracy improves quickly as n increases. However, in a group of 40 robots (a large number by today's standards), a task requiring 5% of the total robots stands a nearly 13% chance of receiving no assignment. If this task is required for the global application, the entire mission will fail. Many practical systems are too small for such probabilistic methods to be reliable.

However, **Random-Choice** is optimally accurate if inter-robot communication is prohibited. One direction for improvement is represented by the probabilistic threshold models developed Deneubourg et al. and others [3], [12]–[14]. After initializing via the **Random-Choice** algorithm, each robot would query its neighors' task states: the further the local distribution is from optimal, the more likely the robot would change its task to rectify the imbalance. This kind of algorithm will converge to a near-optimal distribution in most circumstances. However, these algorithms still have a non-trivial probability of failure for distributions with small relative assignments, or in small-to-medium sized systems. The threshold models are not usable in applications where robots performing the same task need to be near each other.

*B. The **Extreme-Comm** Algorithm*

In the **Extreme-Comm** algorithm, each robot constructs a list of the IDs of all the robots in the network. Each robot selects its task based on its relative position in this list. The algorithm runs quickly, but as its name suggests, it requires a large amount of inter-robot communication.

At every execution cycle, each robot broadcasts a $RobotID$ message to its neighbors containing the tuple *(MyID, timestamp)* of its own ID and its current execution clock value. During each cycle, the robot receives new $RobotID$ messages from its neighbors, and compiles them into a list. In the next cycle, it rebroadcasts this list of messages, along with a new $RobotID$ message of its own with an updated timestamp. The algorithm runs continuously, so that $RobotID$ messages from each robot propagate throughout the network. After $T = Diam(\mathcal{G})$ cycles, each robot will have an accurate list of all the robots in $\mathcal{G}$. A new robot added to the network will propagate messages as above, and its presence will be known to all other robots in at most T cycles. Once a messages has been relayed, it is stored on a separate list for a "refractory period" P before being deleted. The robot does not rebroadcast messages from this list, or any copies of them received from neighbors, a second time. If a robot is removed, its absence will be noted within T execution cycles by all robots, since it will no longer be present to propagate new copies of messages bearing its ID.

Since each robot maintains a list of the IDs of all the other robots in the network it can determine its relative position in that list. Each robot then selects a task by taking its position in the list and binning it with respect to $\mathbf{p}$, just as in the **Random-Choice** algorithm. Note that the algorithm does not require an unbounded timestamp in the $RobotID$ messages, and can use one which resets to 0 after reaching some maximum value, as long as the maximum value is larger than the refractory period P. The largest value of P required is $Diam(\mathcal{G})$, but P can be made as short as 2 cycles, though smaller values make the algorithm less robust to network topology changes. The algorithm is self-stabilizing and converges deterministically to the correct task distribution within T cycles of initialization or $T + P$ cycles of any perturbation.

The communications complexity per-robot per-cycle scales as n, since each robot must send one $RobotID$ message for every robot in the network. The expected total number of messages sent by all robots during convergence scales as $Diam(\mathcal{G}) \times n^2$, which could make this algorithm impractical for a swarm with modest communication bandwidth. The algorithm is also unattractive aesthetically because it collects a large about of global information on each robot that it does not use to solve the problem.

*C. The **Card-Dealer's** Algorithm*

The **Card-Dealer's** algorithm breaks the task assignment problem into a series of stages. At each stage, a robot is "dealt" a task as a function of the stage number, just as a card player is dealt a card as a function of her position around a gaming table. There are as many stages as robots in the system, and the algorithm completes each stage before moving on to the next. This causes the algorithm to run slowly. But unlike **Extreme-Comm**, **Card-Dealer's** never calculates or stores any global quantities, thereby minimizing inter-robot communication and memory requirements.

In every stage of the **Card-Dealer's** algorithm, a "competitive suppression" technique is used to identify the robot in the network with smallest ID. Each robot broadcasts its own ID to its neighbors repeatedly, unless it receives a similar broadcast message indicating that a robot with smaller ID is present. It then switches to rebroadcasting this smaller ID, until it receives a message indicating that a robot with an even smaller ID is present, at which point it switches to rebroadcasting that ID instead. This procedure continues until, after at most $T = Diam(\mathcal{G})$ steps, the smallest ID in the network, say LID_1, has won out and is known to all robots. After waiting for T cycles during which LID_1 has remained stable, the robot with this smallest ID, x_{LID_1}, selects its task according to the rule described below. This robot then declares the beginning of stage 2. This begins another round of competitive suppression, which identifies the second-smallest ID in the network. During this period the first robot relays messages but does not actively participate in the competitive suppression procedure. Once the new minimum ID stabilizes, the selected robot x_{LID_2} selects its task, declares stage 3, and sets itself to be inactive. This procedure continues until the robot with highest ID in the

network selects its task and becomes inactive. When no new stage has been announced for more than T steps, all robots reset the stage counter and reactivate. The whole procedure repeats, running continuously, to allocate robots that have been newly added to or removed from the network.

The **Card-Dealer's** algorithm requires each robot to obtain an estimate of the diameter of the network in order to know how long to wait at each stage. This is to ensure that robots wait long enough to identify the unique minimal ID at a given stage before moving on. Description of a diameter estimation algorithm is given in subsection II-E.

The order of task choice in the **Card-Dealer's** algorithm is based on the fact that any distribution $\mathbf{p} \in \mathbb{Q}^m$ possesses a *minimal representation*, the sequence of integers $\mathbf{v} = (v_1, \ldots, v_m)$ of minimal sum $V = \sum_i v_i$ such that $\mathbf{p} = \mathbf{v}/V$. The simplest ordering would be for the robot at stage s to pick task i if $s \ mod \ V$ is between $v_1 + \ldots + v_{i-1}$ and $v_1 + \ldots + v_i$; this "bins" s with respect to the minimal representation. If the size of the population n is divisible by V, this achieves optimal accuracy. However, if $n \ mod \ V \neq 0$, the last $n \ mod \ V$ robots are part of an incomplete minimal representation. Depending on the order in which the preceding robots have been retasked, the resulting distribution may not be a closest-possible approximation. Because the **Card-Dealer's** algorithm never calculates the total number of robots in the system, it cannot explicitly take advantage of knowing n, and therefore $n \ mod \ V$, to compensate.

However, a better ordering can be chosen, so that regardless of the value of $n \ mod \ V$, the resulting distribution is optimal. The key fact is that the closest approximation to a given $\mathbf{p} = (p_1, \ldots, p_n)$ by an integer sequence of length k, say, $(p_{1,k}, \ldots, p_{n,k})$, is equal to the closest approximation generated from $(p_{1,k-1}, \ldots, p_{n,k-1})$ by adding 1 robot to one of the task types. This can be seen by noting that

$$\begin{aligned} &min_{\sum_i n_i = k} \left\{ \sum_1^m \left(p_i - \tfrac{n_i}{k} \right)^2 \right\} \\ &\quad = min_j \left\{ \left(p_j - \tfrac{p_{j,k-1}(k-1)+1}{k} \right)^2 \right. \\ &\qquad \left. + \sum_{i \neq j} \left(p_i - \tfrac{p_{i,k-1}(k-1)}{k} \right)^2 \right\}. \end{aligned}$$

If we let

$$\begin{aligned} j_k^* \quad &= argmin_j \left\{ \left(p_j - \tfrac{p_{j,k-1}(k-1)+1}{k} \right)^2 \right. \\ &\quad \left. + \sum_{i \neq j} \left(p_i - \tfrac{p_{i,k-1}(k-1)}{k} \right)^2 \right\} \end{aligned}$$

then the sequence $s_k = j_k^*$ is the optimal sequence for each k. Further analysis shows that j^* can be evaluated via a simple non-recursive procedure.

The **Card-Dealer's** algorithm is self-stabilizing under changes in the robot population. First, consider a robot x_{ID} to be added in to the system at time t. This robot will be active when added. If its ID is smaller than that of any of the active robots, it will win the competitive suppression and select its task in that stage. If not, then it will wait for its turn as if it had been present since the beginning. Now consider a robot x_{ID} that is removed from the network. If x_{ID} is still active and there exists an active robot $x_{ID'}$ in the network with $ID' < ID$, then the algorithm will simply skip over the lack of x_{ID} as if it never were present in the first place. If x_{ID} has already selected its task, then the gap will be rectified after all other robots have selected their tasks and the stage counter is reset. The only case in which care must be taken is if x_{ID} is the robot with smallest ID active in the network at that instant. In this case, the network might already have propagated x_{ID} as the lowest active robot, and other robots in the system will have stored this as their value for the LID variable. If x_{ID} is removed before retasking and sending the signal for the next stage, then the algorithm will encounter a liveness fault, because the system will not know to move on and identify the robot with the next largest ID as a replacement for x_{ID}. This problem is solved by adding a liveness-error check. If the value that a robot stores in LID does not change for a longer time than could arise if x_{LID} were present, robots clear LID, wait for $2Diam(\mathcal{G})$ cycles, and return to searching for the robot with the lowest ID.

The **Card-Dealer's** algorithm is as slow as the **Extreme-Comm** algorithm is communications-intensive. The expected running time scales as $O(Diam(\mathcal{G}) \times n)$. However, the per-robot per-cycle communications complexity is constant. Hence, the total number of messages passed between all robots during convergence scales as $Diam(\mathcal{G}) \times n^2$, just as in **Extreme-Comm**.

*D. The **Tree-Recolor** Algorithm*

The **Tree-Recolor** algorithm elects a robot by competitive suppression as in the **Card-Dealer's** algorithm, and uses gradients to construct a spanning tree of the network with that robot as root. By standard tree-summing algorithms, the task distribution in each subtree is computed and propagated up one level, so that the root eventually receives the global distribution. The root uses this information to compute and distribute "retasking" messages that cause robots below it in the tree to switch to the correct task. The **Tree-Recolor** algorithm uses gradient trees to balance communications usage and running time.

A gradient is a multi-hop messaging procedure used in many routing protocols to find optimal routes through *ad hoc* networks [16]. A source robot creates a *gradient message* that is broadcast to its neighbors. Each robot rebroadcasts received gradient messages, which propagate through the network. Whenever a non-source robot receives several gradient messages in the same execution cycle, it relays the one with the lowest hop-count, adding 1 to account for itself. This constructs a tree on the graph $\mathcal{G}$ with the source robot as root. Other robots have a parent and (possibly several) children in this tree. The tree is continually rebuilt from the source outward; if the source is removed, the gradient dies out in a controlled fashion [6]. This dynamic maintenance is essential for applications in swarms of mobile robots. Many elegant algorithms are defined on trees, and by constructing a spanning

tree via gradients, the swarm is able to take advantage of all these algorithms.

In **Tree-Recolor**, each robot begins as a source for the gradient. The gradient propagation algorithm from above is modified so that a message from a source with a lower ID is selected for rebroadcast, even if it has a higher hop-count than a message from another source. Messages from the same source are evaluated as described above. This produces a competitive suppression of higher source IDs (as in the **Card-Dealer's** algorithm), and the robot x_{min} with the minimal ID in the network eventually wins out as the unique source, becoming the root of a spanning tree $\mathcal{T}$ of $\mathcal{G}$. Leaves, *level 1* robots, send messages containing their task to their parents, the *level 2* robots. The level 2 robots sum these messages, computing the number of children they have in each task. They then pass this information up the gradient to their parents, taking care to add 1 for their own task. This repeats on each execution cycle at all levels of the tree, so that the source receives an accurate picture of the current global task distribution, as long as $\mathcal{G}$ remains stable for at least $T = 2 \times Depth(\mathcal{T})$ cycles. In addition, each robot learns what level it is in the tree by taking the maximum of the levels of its children, adding 1, and passing this up to its parent. The source thus learns $Depth(\mathcal{T})$ as well.

When the values of the distribution sums seen by the source are stable for more than $Diam(\mathcal{G})$ cycles, the source calculates a "retasking-array" R, a matrix in which the (i, j)-th entry is the number of robots in task-group i that should switch to task-group j to achieve the goal distribution. The source changes tasks itself, if appropriate, and then calculates and transmits "subtree retasking arrays", $R_1, \ldots, R_k$, one to each of its k children. Care is taken to ensure that each of these arrays contain no more retasking commands for a given task than robots in that subtree. The children process their retasking arrays, changing their own task if necessary, then calculate and transmit subtree retasking arrays to their own children. The retasking commands spread as a wavefront down the tree. When they reach the leaves, the goal distribution is achieved. The source waits T cycles before broadcasting a new retasking array, ensuring that the previous arrays have completed their propagation and the new task sums reflecting the changes have accumulated back at the source. Transmitting new retasking arrays earlier than this could cause the global task assignment to become unstable, and goal distribution to never to be achieved.

Since $Depth(\mathcal{T}) \leq Diam(\mathcal{G})$, the run time of the **Tree-Recolor** algorithm is $O(Diam(\mathcal{G}))$, comparable to that of the **Extreme-Comm** algorithm. However, the per-robot per-cycle communications usage scales as $O(km)$, where k is the maximum allowed number of retasking messages transmitted per cycle, plus one gradient message. This maximum is determined by the hardware implementation. Communications usage is significantly better than **Extreme-Comm**, as m, the number of different tasks, is usually smaller than n. The total number of messages passed throughout one full run of the algorithm scales as $O(m \times n \times Diam(\mathcal{G}))$ which is one factor less in n than the previous two algorithms.

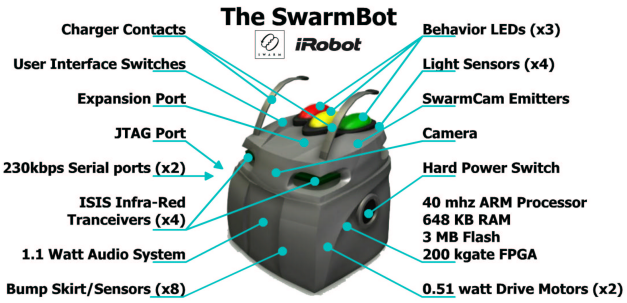

Fig. 4. The iRobot SwarmBot is designed for *in vivo* distributed algorithm development. Each SwarmBot has four IR transceivers, one in each corner, allowing nearby robots to communicate and determine the bearing, orientation, and range of their neighbors. An omnidirectional bump skirt provides robust low-level obstacle avoidance. The 40 MHz 32-bit microprocessor provides more than enough processing power for our algorithms.

E. Diameter Estimation

The **Card-Dealer's** and **Tree-Recolor** algorithms need to synchronize robot state transitions to within a diameter period across the network. To achieve this, they must estimate the worst-case time that a message needs to propagate throughout the network. This is determined by the diameter $Diam(\mathcal{G})$, the maximum distance between any two nodes in $\mathcal{G}$.

The diameter can be estimated by starting a gradient tree from any node $a \in \mathcal{G}$, and noting that $Diam(\mathcal{G}) \leq 2D$ where D is the depth of the gradient tree. The **Card-Dealer's** algorithm runs a diameter estimation algorithm from the robot in the system with smallest ID. Maximum depth estimates are propagated up the tree to the root, which then broadcasts the global maximum to the rest of the robots via the gradient.[2] The **Tree-Recolor** algorithm exactly replicates this calculation when it determines the tree depth as described above.

III. Implementation

A. Hardware Description

We implemented the algorithms on a group of iRobot SwarmBots. Each SwarmBot (Figure 4) is mobile and contains a suite of sensors, inter-robot communication and localization, and a microprocessor.

Each robot transmits its public state at the end of every execution cycle. The time for each cycle is the same for all robots, which ensures that each robot will receive only one set of messages from each of its neighbors during any cycle, enforcing the synchronicity model from section B. The cycle period is 250 ms, short enough for smooth robot motion control based on neighbor positions, but long enough to preserve communications bandwidth. The link layer protocol is similar to the Aloha [17] protocol, in that each robot

[2]When the algorithm initializes, before identifying LID_1, each robot uses $MaxRobots$ as its (over-)estimate for the tree diameter.

Algorithm	running time	Per Robot Comm. Rate	Total Comm.	Error Variance
Random-Choice	$O(1)$	0	0	$O(1/n)$
Extreme-Comm	$O(Diam(\mathcal{G}))$	$O(n)$	$O(n^2 Diam(\mathcal{G}))$	0
Card-Dealer's	$O(n \times Diam(\mathcal{G})$	$O(1)$	$O(n^2 Diam(\mathcal{G}))$	0
Tree-Recolor	$O(Diam(\mathcal{G}))$	$O(m)$	$O(nmDiam(\mathcal{G}))$	0

Fig. 3. Comparison of the asymptotic theoretical upper bounds on the four algorithms' running time, communications usage, and accuracy. The notation n, m, and $\mathcal{G}$ respectively denote the total number of robots in the network, the number of tasks in the goal distribution, and the network graph. Running time is defined as the number of execution cycles between stabilization of the robot population and stabilization of the final task distribution. The per-robot per-cycle communications rate is defined as the number of messages sent by a single robot in a single execution cycle. (There are four execution cycles per second.) The total communications burden on the network is defined to be the sum over all robots of the per-cycle per-robot communications rate until the algorithm converges. Accuracy is measured by the variance of the theoretical error distribution between final stabilized state and actual target distribution.

transmits with minimal [3] checking to prevent collisions with a neighboring robot's transmission. Care must be taken to not saturate the communications channel, as this can cause network "crystallization" and catastrophic communication failures. With 10 neighbors, each robot has sufficient bandwidth to receive about 18 messages per neighbor per cycle, a very tight constraint.

Each SwarmBot is adorned with large red, green, and blue "Behavior LEDs" and has a MIDI audio system. We use these lights and sounds to monitor the internal state of the robots. In particular, the task a robot has selected is represented by illuminating one of these LEDs, so we limited our experiments to three task groups – red, green and blue – to allow the use of a standard video camera for data collection.

B. Experiments

We conducted three sets of experiments to measure each algorithm's assignment accuracy and convergence time, running time as a function of total number of robots, and stability to external disturbances.

Convergence and Accuracy: The first experiment was designed to measure convergence time and accuracy. Convergence error is defined as the distance from the current distribution vector and a fixed goal distribution $\mathbf{p} = (1/6, 1/3, 1/2)$: $e = ||dv(t) - \mathbf{p}||_2$. The results are shown in Figure 5. All three algorithms were highly accurate, with **Extreme-Comm** and **Card-Dealer's** producing final assignments with no errors. The **Extreme-Comm** algorithm converged the fastest, but **Card-Dealer's** outperformed **Tree-Recolor** on average, ignoring the theoretical results that predict otherwise. The **Card-Dealer's** algorithm uses little communications, and ran exceptionally well on the Swarm. But, even for this small swarm, the average convergence time was 53 seconds, making this algorithm impractical for large swarms. The **Tree-Recolor** algorithm relies on gradient trees to propagate partial sums of the current distribution back to the root robot. Messages dropped during this process were common, which caused incorrect summations, leading to an incorrect distribution accumulating at the root, and then incorrect retasking arrays sent back down the tree. Some of the experimental runs showed the promise of the **Tree-Recolor** algorithm, with the correct assignment being achieved in two retasking cycles (21 seconds), only twice the minimum running time. There are techniques in the literature to help stabilize the tree computations (taking the max over windows of time, encoding packet routing history to deal with topology changes [15], etc.), some of which might bring the convergence time closer to the theoretical limits. It is interesting to note that **Extreme-Comm** and **Tree-Recolor** both measure the current distribution and use feedback to converge monotonically toward the goal, while **Card-Dealer's** makes uninformed task assignments at each round, causing the global error to increase while running.

Running Time: The second experiment measured the convergence time as a function of the number of robots in the network. We progressed from 4 to 25 robots, which is close to the bandwidth limit for **Extreme-Comm**. We arranged the robots so that the diameter of the network was known *a priori* and the minimum running time could be computed directly. For each group size, several runs from a random initial distribution to an assignment of $\mathbf{p} = (1/6, 1/3, 1/2)$ were timed. The execution cycle of 250 ms and the algorithm design were used to compute a lower bound on the running time, but actual running time depended on communications errors, as lost messages impeded performance. This can be modeled by extending the execution cycle and calculating expected running times. The expected execution cycle is $\sim$ 37% longer than the minimum [6]. The results are shown in Figure 6.

The **Extreme-Comm** algorithm performed well, but as the communications burden increased, the measured running times moved further from the expected running times, possibly because the communications usage was outside of the usage patterns used to characterize the system. The **Card-Dealer's** algorithm ran perfectly each time, generating data with almost no variance. Again, the **Tree-Recolor** algorithm ran slower than expected, with a large variance in convergence time. The source of errors was the same as the previous experiment - incorrect sums caused by fragile gradient trees.

Disturbance Rejection and Self-Stabilization: The third set of experiments measured the disturbance rejection properties of the three algorithms. Four disturbances were introduced in sequence: 1. Initial transition from $\mathbf{p} = null$ to $\mathbf{p} = (1, 0, 0)$, 2. transition from $\mathbf{p} = (1, 0, 0)$ to $\mathbf{p} = (0, 1/2, 1/2)$, 3. transition from $n = 24$ to $n = 12$ while keeping $\mathbf{p}$ constant, 4. transition from $n = 12$ to $n = 24$ while keeping $\mathbf{p}$ constant. A centralized radio network was used to broadcast the new distribution and population commands to a group of 24 robots. Figure 7 shows the results, with grey lines indicating when

[3] If a robot is receiving a message, it will wait to transmit, but will not adjust the transmit time for the next neighbor cycle. This ensures synchronicity, but can cause repeated collisions when the robots are stationary.

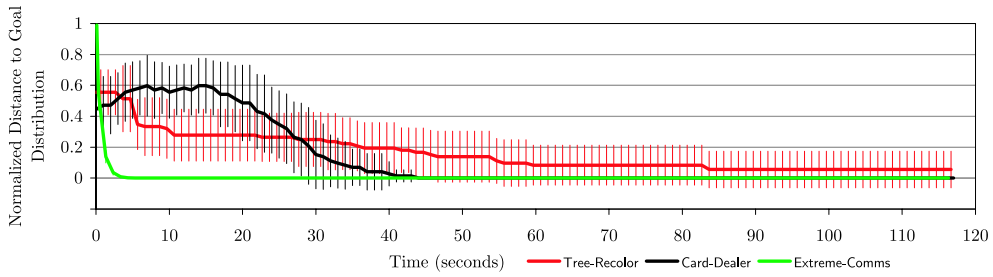

Fig. 5. The convergence properties were measured by running each algorithm 8 times on a group of 18 robots, and measuring the normalized error as a function of time. In all runs, the robots started from an random initial assignment, and converged towards $\mathbf{p} = (1/6, 1/3, 1/2)$. The **Extreme-Comm** and **Card-Dealer's** algorithms' temporal performance was close to the theoretical limits, and both converged to the correct distribution in all runs with no error. The **Tree-Recolor** algorithm struggled with communication errors which extended its running time well beyond the theory, but it still converged to the desired distribution.

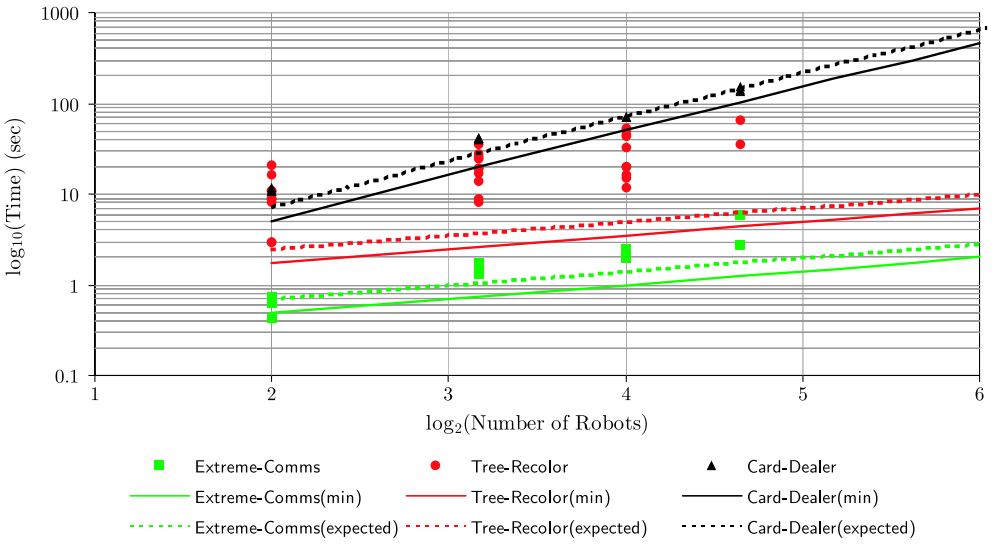

Fig. 6. The running time of all three algorithms scale with the total number of robots in the network. This graph shows time plotted against the number of robots on log-log axes. Configurations containing between 4 and 25 robots were tested. Solid lines show the predicted running time with no communications errors, dashed lines show the expected running time with typical errors. Dots are data points from individual experiments. The **Card-Dealer's** and **Extreme-Comm** algorithms performance was close to the expected running time, while the **Tree-Recolor** algorithm was far from expected. This was caused by incorrect summation information propagating back to the root of the gradient tree, making the subsequent retasking commands incorrect.

each of the disturbances occurred.

The **Extreme-Comm** algorithm should have easily handled all of these disturbances. Unfortunately, problems with the robots' radios corrupted many of the data sets. Attempts were made to separate errors caused by the experimental setup from errors caused by inter-robot communications, but unexplained variances are still present. The distribution change at $t = 5$ occurs very quickly, because each robot has already compiled the list of all the other robots, and can select a new task in $O(1)$ time. Removing half of the robots produced a plateau in the error from $t = 10$ to $t = 12.5$ because the timestamp in each $RobotID$ message must become invalid before that message can be removed from a list of robots. This is in contrast to the next disturbance at $t = 20$ when the 12 robots are turned back on. The algorithm adds robots as new messages arrive, quickly driving the error towards zero.

Because the **Card-Dealer's** algorithm does not measure the current distribution, its worst case response to each disturbance is the same: $2 \times n \times Diam(\mathcal{G})$. The initial assignment and the distribution change show that **Card-Dealer's** only changes the task of one robot at a time. The small error after the population change from 24 to 12 robots is misleading, as removing random robots did not have a large effect on the global distribution, but caused **Card-Dealer's** to change the tasks on many of the remaining robots.

The **Tree-Recolor** algorithm was hindered by the same experimental setup as **Extreme-Comm** as well as the gradient-tree summation errors mentioned earlier. As before, there was a large variance on convergence time, pointing to room for improvement.

IV. Conclusion and Future Work

The four solutions presented here exhibit different scaling properties and communications requirements. All four could find a place in the distributed algorithm designer's toolbox

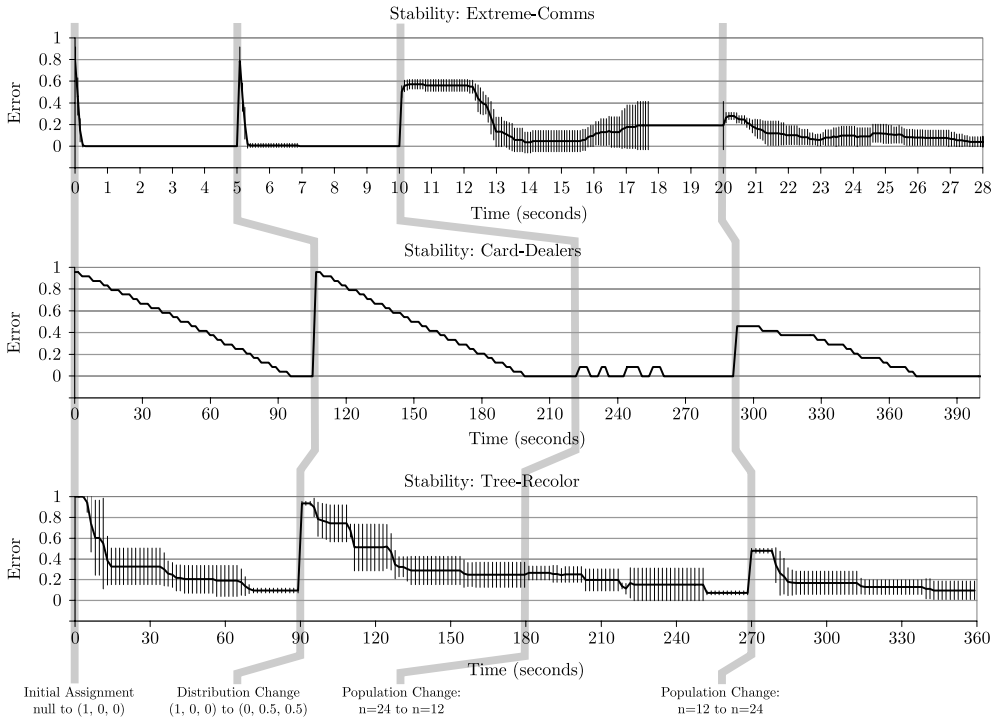

Fig. 7. All of the algorithms are self-stabilizing when subject to external disturbances. This trio of graphs displays the recovery of the algorithms when subject to four different external disturbances; the initial convergence, a goal distribution change, removal of half of the robots, and replacement of the removed robots. Normalized error is plotted against time in all graphs, but the absolute times in each graph are different. Eight runs of **Extreme-Comm** produced the data on top. The algorithm performed well, but the data is partially corrupted by errors in our experimental setup. Some of this can be seen after t = 16, as the error begins to rise, even though there is no disturbance (compare with Fig. 5). The plot of **Card-Dealer's** shows the algorithm's steady progress. Multiple runs produced identical results, so only one example is shown here. The magnitude of the error after the first population change is small, but many robots needed to switch tasks, which can be very disruptive. The **Tree-Recolor** algorithm was hindered by the same experimental setup as **Extreme-Comm** as well as the gradient-tree summation errors mentioned in the text. It converged accurately, but not rapidly.

for practical dynamic task assignment. A useful enhancement would be the ability to specify absolute requirements on the minimum or maximum number of robots assigned to a task. One of the most intriguing research directions suggested by these algorithms is exploration of resource trade-offs that are minimized by the most efficient algorithms, and are bounded below by some "conserved quantity" associated with the dynamic task assignment problem. Another important question is to understand how to combine results from this work with approaches to locally determining the optimal task distribution. Future work could follow up on this theoretical question, in addition to implementing and optimizing specific solutions.

ACKNOWLEDGEMENTS

Support for J. McLurkin provided in part by Boeing and DARPA IPTO under contracts DASG60-02-C-0028 and N66001-99-C-8513. Support for D. Yamins provided in part by a National Science Foundation Graduate Research Fellowship. The authors would like to thank L. Kaelbling, D. Rus, and D. Bourne for invaluable help during the preparation of this paper.

REFERENCES

[1] S Camazine, et al. *Self-Organizing Biological Systems* Princeton Univ. Press, 2001.

[2] E.O. Wilson and G. Oster. *Caste in Social Insects*. Princeton Univ. Press, 1979.

[3] M Dorigo et al. *New Ideas in Optimization* ACM Dig. Lib. 1999

[4] P Ogren, E Fiorelli, NE Leonard. *Formations with a Mission*, Proc. MTNS, 2002.

[5] WJ Butera, *Progamming a Paintable Computer*, Ph.D. Thesis MIT 2002.

[6] J McLurkin, *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*, S.M Thesis. MIT 2004.

[7] JM Kahn et al., *Next Century Challenges: Mobile Networking for "Smart-Dust"*, Proc. 5th ACM/IEEE CMCN. 1999.

[8] MJ Mataric, GS Sukhatme, EH Ostergaard. *Distributed Multi-robot Task Allocation for Emergency Handling*, Autonomous Robots 14, 2003.

[9] BP Gerkey, MJ Mataric. *Multi-robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures*, Int'l J. of Robotics Research 23 (9), 2004.

[10] O. Shehory, S. Kraus. *Methods for Task Allocation via Agent Coalition Formation*, Artificial Intelligence 101, 1998.

[11] J. Valk and C. Witteveen. *Lecture Notes in AI 52417*, 2002.

[12] E Bonabeau, G Theraulaz, JL Deneubourg. *Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in Insect Societies*, Bull. Math. Bio, 60, 1998.

[13] MJB Krieger, JB Billeter, L Keller. *Ant-like Task Allocation and Recruitment in Cooperative Robots*, Nature 406 (31) 2000.

[14] S Nouyan. *Agent-Based Approach to Dynamic Task Allocation* , ANTS 2002 (Lec. Notes in CS 2463), 2002.

[15] S Nath, PB Gibbons, S Seshan, Z Anderson. *Synopsis Diffusion for Robust Aggregation in Sensor Networks*, ACM SenSys 2004.

[16] C Intanagonwiwat, R Govindan, D Estrin. *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks*, Proc. 6th Ann. Conf. on Mobile Computing and Networks, 2000.

[17] N Abramson, *The Aloha System – Another Alternative for Computer Communications*, Proc. Fall Joint Comput. Conf., AFIPS Conf., 1970.