Ant-Inspired Dynamic Task Allocation via Gossiping

Hsin-Hao Su[1], Lili Su[1], Anna Dornhaus[2], and Nancy Lynch[1]

[1] CSAIL, MIT
[2] Department of Ecology and Evolutionary Biology, University of Arizona

Abstract. We study the distributed task allocation problem in multi-agent systems, where each agent selects a task in such a way that, collectively, they achieve a proper global task allocation. In this paper, inspired by specialization on division of labor in ant colonies, we propose several scalable and efficient algorithms to *dynamically* allocate the agents *as the task demands change*. The algorithms have their own pros and cons, with respect to (1) how fast they react to dynamic demands change, (2) how many agents need to switch tasks, (3) whether extra agents are needed, and (4) whether they are resilient to faults.

1 Introduction

In a multi-agent system, different tasks may need to be performed. The task allocation problem is to find an allocation of agents such that there are enough agents working on each task. This is often done in a distributed manner in many applications. For instance, drone package delivery for one city may consist of deliveries for several different regions [20]. The drones may learn the demands in each region from a broadcasting ground control station. The demands may change from time to time. The drones are required to coordinate among themselves (upon receiving the same signal), without central control, to ensure that there are enough individuals working in each region.

The problem of task allocation also occurs in the ant world. In ant colonies, there are several different tasks (brood care, foraging, nest maintenance, defense [29]) which require different numbers of ants. Ant colonies generally do a good job of regulating the assignment of workers to tasks. In this work, we take inspiration from specialization in ants to develop several algorithms that are efficient and robust for the task allocation problem. Conversely, we hope our work can shed some light on questions about collective insect behavior.

To model the task allocation without centralized controllers, we consider randomized *gossip protocols* [6] (which are similar to *population protocols* [1]) as the underlying method of communication among the agents. In short, randomized gossip protocols consist of *rounds*. In each round, each agent chooses an agent uniformly at random to contact, and then the pair exchanges messages. Gossip-based protocols capture a common method of communication in biological systems. For example, in ant colonies, two ants communicate by touching each other with their antennae [15]. The gossip protocol also captures any methods of peer-to-peer information exchange, including indirect communication such as one agent leaving marks for another agent. Not only are gossip-based protocols natural communication mechanisms in biological systems, the algorithms in such protocols are usually simple, easily scalable, and resilient to failures.

1.1 The Model

We assume there are n agents and k tasks. Each agent a is associated with a unique identifier, $\mathrm{ID}_a \leq \mathrm{poly}(n)$, and a state $Q_a \in \{1, 2, \ldots, k\}$, which indicates the task that it is working on. In ant colonies, the ID can be thought as an encoding of features of an ant such as age, body size, genetic backgrounds, or spatial fidelity zones [30]. Also, in such settings, k is usually a small constant less than 20. The scenario proceeds in synchronized rounds. In the beginning of round t, each agent receives the demand signals $\boldsymbol{d}^{(t)} = (d_1^{(t)}, d_2^{(t)}, \ldots, d_k^{(t)})$ from the tasks, where $d_i^{(t)}$ indicates the demand of task i. [3] Note that the demands should be thought as the work-rates required to keep the tasks satisfied. The demands may change arbitrarily in every round. Each agent a chooses another agent a' uniformly at random and then they can exchange messages of $O(k \log n)$ bits (which are just enough fit the size of the input signals, $d^{(t)}$). Then, the agents can change their states. Then they proceed to the next round.

Cornejo et al. [5] and Radeva et al. [28] defined models for the task allocation problem in ant colonies. In their work, when the ants receive heterogeneous feedback from the environment, there could be information flow from one ant to another. In our model, *the information flow happens only through gossiping.* Many models inspired by insect colonies have restrictive memory constraints on each agent. However, since evidence shows that insects can remember and learn things (such as a path) very well [8], we decide not to impose constraints on memory, but on communication.

1.2 Problem Formulation

We formulate the task allocation problem similarly to [28] and [5] as follows. Let $A_i^{(t)}$ denote the set of agents working on task i for $1 \leq i \leq k$. Let $\boldsymbol{w}^{(t)} = (w_1^{(t)}, w_2^{(t)}, \ldots, w_k^{(t)})$ denote the number of agents working on the k tasks ($w_i = |A_i|$). We say the allocation at round t is a *proper allocation* if $w_i^{(t)} \geq d_i^{(t)}$ for all $i \in \{1, 2, \ldots, k\}$. For convenience, assume that the total demand $D = \sum_{i=1}^{k} d_i$ is fixed. We can assume this without loss of generality, since we can let task k denote the dummy task for idle agents. We often omit the superscripts (t) to denote the quantities of the current round.

[3] Although the assumption that every agent knows all the demands seems to be strong, as long as each demand is known by some agent, all the demands can be propagated to everyone in $O(\log n)$ rounds by gossiping (see broadcasting in Section 2).

	Mv. and Fill	Tkn. Pass I	Tkn. Pass II	Ranking I	Ranking II
#Agents	D	$(1+\epsilon)D$	$(1+\epsilon)D$	$(1+\epsilon)D$	$(1+\epsilon)D$
Preproc. Time		$O(\frac{k}{\epsilon}\log n)$	$O(\frac{k}{\epsilon}\log n)$	$O(\frac{1}{\epsilon}\log^2 n)$	$O((\frac{k}{\epsilon})^2\log n)$
Realloc. Time	$O(\log^2 n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Switching Cost	OPT	$(k-1)\cdot$ OPT	OPT	$O(n)$	$O(k\log n)\cdot$ OPT (or $O(n)$)
Fault Tolerance		transient faults after preproc.		transient faults after preproc.	transient & (crash) no global clock

Table 1: The time complexities are provided as the rounds needed for the algorithms to succeed with high probability, that is, with probability $1 - 1/\operatorname{poly}(n)$.

There are several objectives we consider for an algorithm. First, whenever the demands change, we would like the allocation to recover to a proper one as soon as possible. The reallocation time is defined to be the number of rounds needed for the algorithm to find a proper allocation, after the demand stabilizes. Algorithms are allowed to have a preprocessing phase, so that the reallocation can be done faster after that. Second, when the demands change, we hope the number of task switches is as small as possible, since task changing may incur some overheads. We define the switching cost to be the number of agents who switched tasks until a proper allocation is achieved. Suppose the number of agents equals to the total demand. When the demands change from $\boldsymbol{d}$ to $\boldsymbol{d}'$, it is clear that the switching cost is at least $\text{OPT} \overset{\text{def}}{=} |\boldsymbol{d} - \boldsymbol{d}'|_1/2$. Third, we study the number of agents needed for the algorithm. Clearly, all algorithms that behave correctly need to have at least D agents. However, the question is whether extra agents can help us in designing more efficient algorithms.

Finally, we consider two types of faults: **transient faults** and **crash faults**. A transient fault means an agent temporary malfunctions but later recovers. For example, an agent might not receive the most recent demands for some reason (perhaps due to the propagation delay). We say an algorithm tolerates transient faults if the agents adapt to a proper allocation after all the agents recover from the faults. A crash fault is when an agent malfunctions permanently (and it will no longer be contacted by other agents). We say an algorithm tolerates crash faults if the agents adapt to a proper allocation after some of them have crashed, as long as there are enough remaining agents.

1.3 Our Contribution

We explore different possibilities that can be achieved by a(ge)nts under the gossip model where information exchanges are limited. We give several algorithms for the task allocation problem. Some algorithms are inspired by ants, where their intrinsic difference is used to facilitate symmetry breaking. The algorithms are incomparable in the sense that no one dominates the other on all the objectives (see Table 1). Our first algorithm, the **move-and-fill algorithm**, is a straightforward algorithm, where the excess ants working on over-satisfied tasks leave the tasks and switch to the unsatisfied tasks. We show that this can be done in $O(\log^2 n)$ rounds in our model w.h.p.[4] using the gossip-based counting and selection algorithms developed in [19] . The main advantage of the move-and-fill algorithm over the other two is that the number of agents needed is exactly D. Moreover, the switching cost is optimal. The drawback of the algorithm is that whenever the demands change, the re-allocation time is $O(\log^2 n)$ rounds. If the demands change more frequent than once every $O(\log^2 n)$ rounds, the allocation will not be able to catch up to the demands. In reality, the demands may change very frequently due to both internal factors (consumable tasks where the demands decrease when they are done) and external factors (sudden changes in the environment).

The ant inspiration for the next two algorithms. Consider ant colonies, where ants receive the demand signals from the tasks. In reality, the signals can be the temperature or the production of chemicals. Biologists have conjectured that different ants have different response thresholds to the signals [4]. The question is whether such a design could help in task allocation. Consider the following simple example, where $n = k = 2$. Suppose that the first ant a_1 is more sensitive to the signal of task 1 than a_2. Then, when task 1 and task 2 have both 1 unit of demand, it is possible that a_1 goes to work on task 1 and a_2 goes to work on task 2. The main inspiration here is that if the ants have different responses to the signals, then they can take advantage of the difference to facilitate task allocation. Each ant can decide where to go based on the demand signals, independent of the other ants' actions. Therefore, the reallocation can be done very quickly.

Both our **token passing algorithm** and **ranking algorithm** are based on this idea. Both algorithms consist of a preprocessing phase, where each ant a computes a value X_a. After X_a is computed, they will allocate themselves according to X_a and the vector of demands, so that when the demands change, each ant can reallocate itself instantaneously. The drawback compared to the first algorithm is that they both need extra agents. After the X_a-values are computed, the allocation is done in a very simple way. In a high level sense, we divide the range of X_a-values into k disjoint intervals such that the length of i'th interval is proportional to the demand of task i (with additional slacks, see Algorithm 2). Every agent will go to the task whose interval contains its X_a-value. In general, we hope that the X_a-values of the agents are well-spread so that an interval of length proportional to d_i would contain d_i agents whose X_a-values lie in the interval.

In the **token passing algorithm**, each agent is assigned a unique token X_a from $\{1, 2, \ldots, n + \lfloor \frac{\epsilon}{k} \cdot D \rfloor\}$ in the preprocessing phase, where $0 < \epsilon < 1$ is a fixed parameter. This is done by using the loose renaming procedure developed by Giakkoupis, Kermarrec, and Woelfel [14]. When there are $(1 + \epsilon) \cdot D$ agents, the preprocessing phase takes $O(\frac{k}{\epsilon} \log n)$ rounds. After that, each agent can determine its role based on X_a and the demand vector in $O(1)$ rounds. There are two variants of the algorithms that reallocate in different ways when the demands

[4] With high probability, which means with probability at least $1 - 1/\operatorname{poly}(n)$. Note that if there are $\operatorname{poly}(n)$ events and each one holds w.h.p., then all of them simultaneously hold w.h.p. by an union bound argument.

change. The first is that every agent keeps the X_a-value the same and then reallocates according to that. In that algorithm, the switching cost is bounded by $(k-1)\cdot \mathrm{OPT}$. In the second variant, the X_a-values are also reallocated. This achieves the optimal switching cost and the reassignments of X_a-values can be done instantaneously. However, unlike the first variant it does not tolerate transient faults after the preprocessing phase.

We define the notion of *stable algorithms* which capture the type of algorithms where each agent's decision only depends on the current input signals. As long as the agents run Algorithm 2 with fixed X_a-values (like the first variant), the resulting algorithm is stable. The stable algorithms are resilient to transient faults, because as long as each agent functions normally and receives the current input signals, the allocation is proper. We show that for this type of algorithms, the switching cost is at least $2\cdot \mathrm{OPT}$ when $n = D$. In comparison, our first variant achieves a switching cost of $(k-1)\cdot \mathrm{OPT}$. We discuss the possibility to close the gap in Section 5.

In the **ranking algorithm**, X_a is an estimate of the normalized rank (i.e. $\mathrm{rank}(a)/n$) of a, where $\mathrm{rank}(a)$ is the rank of a's ID over all the agents. In fact, the algorithm is similar to ants' behavior. The ID of each agent can be thought as some features of the agents. In ant colonies, ants allocate themselves to the tasks based on their individual traits. For example, there are tendencies for older ants to forage while younger ants work on tasks within or near the nests [31]. The ranking algorithms follow this general strategy by allocating every agent based on the value of its trait (assuming the traits are comparable).

We propose two different ways for estimating the normalized ranks. The first is a rounding-based algorithm that runs in $O(\frac{1}{\epsilon}\log^2 n)$ rounds while the second is a lightweight sampling-based algorithm runs in $O((\frac{k}{\epsilon})^2 \log n)$ rounds. The advantage of the first one is that the estimate does not depend on the execution of the algorithm and so that an agent always gets the same estimate. The advantage of the second algorithm is that it can tolerate both transient and crash faults. Moreover, each agent is allowed to keep its own clock–there is no global clock. Also, it is a **self-stabilizing algorithm**, which always converges to a proper allocation given arbitrary initial states of the agents. The drawback is that the algorithms may have a fairly large switching cost (e.g., task switching can happen even when the demands are stabilized). However, for the second variant we may sacrifice the crash fault tolerance for a bounded switching cost of $O(k\log n)\cdot \mathrm{OPT}$ by fixing the X_a-values after the agents get accurate enough estimates.

1.4 Related Work

The task allocation problem in ant colonies has been studied extensively in biology literature. Empirical works suggest that the task an ant chooses to work on depends on various factors, including its age [29], body size [32], genetic background [18], position in the nest [30], social interaction [17], and internal response threshold [3]. There are also works that formulate the task allocation problems using mathematical models [2, 3, 16, 25, 26].

Cornejo et al. [5] was the first to model the ant task allocation problem from a distributed computing perspective. Then, [28] studied how extra agents can speed up the task allocation process. In the ant-colony task allocation models of [28] and [5], they assumed the signals the agents received from the tasks are the deficit (i.e. $d_i - w_i$) or whether the tasks need more work (i.e. $\mathrm{sgn}(w_i - d_i)$). It is not clear what the signals the ants are actually receiving in reality, and they may depend on the type of the tasks.

In computer science, task allocation problems have also been well-studied under various contexts, including scheduling of multiprocessors [7, 22], robotics [23, 12], and communication complexity [9]. The major difference between their problems and ours is that we consider the case where the tasks are understood as the task types, where each task (type) is accomplished by many agents collaboratively. As a result, the number of tasks is usually much smaller than the number of agents. While in their cases each task is a single instance that will be handled by a single agent. Their goal is to study how the tasks should be assigned to the agents such that some objectives are optimized (possibly under some constraints).

Recently, there has been a rising number of work to model collective insect behavior from a distributed computing perspective. This includes the studies for the foraging problem [21, 10], the house-hunting problem [13], and density estimation problem [24]. See [27] for a more comprehensive survey.

1.5 Organization

In Section 2, we present the move-and-fill algorithm. In Section 3, we present the token-passing algorithm. In Section 4, we present the ranking algorithm. Finally, in Section 5, we propose open problems inspired by this work.

2 The Move-and-Fill Algorithm

We first present an algorithm that does not require extra (more than D) agents and achieves an optimal switching cost. Before describing the algorithm, we review a few elementary subroutines that can be achieved by gossip protocols.

- **Broadcasting**. Suppose that a message m is initiated at a. Suppose that in each round, each agent a that holds m forwards it to a', the agent being contacted by a. Frieze and Grimmett [11, Theorem 5.2] showed that in $O(\log n)$ rounds, w.h.p. all the agents receive the message.
- **Counting**. For any $1 \leq i \leq k$, the number of agents working on task i can be counted in $O(\log n)$ rounds w.h.p. Suppose that each node (or agent, in our case) is associated with an integer. The push-sum algorithm of [19, Algorithm 1] approximates the summation up to a $(1 \pm \epsilon)$ factor via gossiping in $O(\log n + \log(1/\epsilon) + \log(1/\delta))$ rounds with probability at least $1 - \delta$. Let A_i denote the set of agents working on task i for $1 \leq i \leq k$. To count the number of agents in A_i, we let the agents in A_i initiate the values to 1 and

agents not in A_i set their values to 0. Then, by approximating the summation using the algorithm with $\epsilon = 1/(2n+1)$ and $\delta = 1/\operatorname{poly}(n)$, we can count the exact number of agents working on task i in $O(\log n)$ rounds w.h.p. Also, since our bandwidth on the messages is $O(k \log n)$, we can run k executions of the algorithms in parallel in $O(\log n)$ rounds to count the number of agents working on every task.

- **Selection and Rank Testing.** Let $A' \subseteq A$ be a set of agents and let r be an integer. Suppose that $a \in A'$, let $\operatorname{rank}_{A'}(a)$ denote the rank of a in the set A' ordered by IDs, *beginning with 0*. We explain that in $O(\log^2 n)$ rounds, w.h.p. each agent a in A' can determine whether $\operatorname{rank}_{A'}(a)$ is at least r or not. Kempe, Dobra, and Gehrke [19, Theorem 4.2] gave an algorithm for computing the t'th smallest element in $O(\log^2 n)$ rounds w.h.p. We can set $t = r+1$ and use their algorithm to find out the ID of the agent a with $\operatorname{rank}_{A'}(a) = r$. Then, it will broadcast its ID to every agent in $O(\log n)$ rounds. Therefore, all agents can compare their own IDs with the received ID to determine whether its rank is at least r. Again, we can run k copies of the algorithms simultaneously, since each copy uses only $O(\log n)$ message size.

Algorithm 1 The Move-and-Fill Algorithm

```
Obtain $w_1, \ldots, w_k$ by counting the number of agents working on each task.
For $1 \le i \le k$, for each $a \in A_i$, include $a$ in $A'$ if $\operatorname{rank}_{A_i}(a) \ge d_i$.
Let $\phi_i = \max(d_i - w_i, 0)$ be the deficit of task $i$ for $1 \le i \le k$.
For $1 \le i \le k$, let $\Phi_i = \sum_{j=1}^{i} \phi_i$ be the prefix sum of the deficit and let $\Phi_0 = 0$.
Let $I_i = [\Phi_{i-1}, \Phi_i)$ be the half-open intervals for $1 \le i \le k$.
For each $a \in A'$, go to task $i(a)$, where $I_{i(a)}$ is the interval that contains $\operatorname{rank}_{A'}(a)$.
```

We assume $n = D$. The **move-and-fill algorithm** is described in Algorithm 1. The algorithm is similar to Radeva et al.'s algorithm [28], where the excess agents at each task pop out and move to the unsatisfied tasks. We use A' to denote the set of excess agents. Agents in A' will reassign themselves to the unsatisfied tasks according to the deficits and their ranks in A'. To determine whether $a \in A'$, a does so by testing whether $\operatorname{rank}_{A_{Q_a}}(a) \ge d_{Q_a}$. Such a test can be done for every agent by running k rank testing algorithms (one for each task) in parallel w.h.p. For task i, $\max(0, w_i - d_i)$ agents will be in A'. Since the number of agents is equal to the total demand, the number of excess agents must be equal to the total deficits of the tasks, Φ_k. We partition the interval of length Φ_k into k intervals, each with length $\Phi_i - \Phi_{i-1} = \phi_i$, so that there will be exactly ϕ_i agents whose $\operatorname{rank}_{A'}(a)$ lie in the interval I_i. Thus, ϕ_i agents will go to task i. This implies all the tasks become satisfied. Again, w.h.p. such a test can be done for every agent by running k rank testing algorithm to test if $\operatorname{rank}_{A'}(a) \ge \Phi_i$ for each i.

Therefore, the number of rounds needed to get to a proper allocation is $O(\log^2 n)$ w.h.p. Suppose that the demands change from $\boldsymbol{d}$ to $\boldsymbol{d}'$ and then do not change for $\Omega(\log^2 n)$ rounds. It is clear that the algorithm achieves an optimal switching cost of $\mathrm{OPT} = |\boldsymbol{d} - \boldsymbol{d}'|_1/2$, since the number of agents who switched tasks is $\sum_{i=1}^{k} \max(0, d'_i - w_i) = \sum_{i=1}^{k} \min(0, d'_i - d_i) = |\boldsymbol{d}' - \boldsymbol{d}|_1/2$.

3 The Token Passing Algorithm

While the move-and-fill algorithm achieves an optimal switching cost, it requires a significant amount of re-computation whenever the demands change. In situations where the demands change more frequent than $O(\log^2 n)$, the algorithm may fail. In this section, we present an algorithm that reallocates in $O(1)$ rounds whenever the demands change. However, the algorithm requires some extra agents in addition to the total demand.

We assume that $n = \lceil (1+\epsilon)D \rceil - \lfloor \frac{\epsilon}{k} \cdot D \rfloor$, which is slightly less than $\lceil (1+\epsilon)D \rceil$. In the preprocessing phase, we assign each agent a a token TK_a from $\{0, \ldots, \lceil (1+\epsilon)D \rceil - 1\}$ such that each token is assigned to at most one agent. Giakkoupis et al. [14] gave an algorithm for the renaming problem in the gossip model that assigns a name from the name space $\{1, 2, \ldots, (1+\epsilon')n\}$ to each node in $O(\frac{1}{\epsilon'} \log n)$ rounds, where n is the number of nodes. This can be used to assign the tokens for the agents in our case, where we have $n \leq (1+\epsilon)D - \frac{\epsilon}{k} \cdot D + 2$ agents and at least $(1+\epsilon)D$ tokens and so $\epsilon' \geq \frac{(1+\epsilon)D}{n} - 1 = \frac{(1+\epsilon)D}{(1+\epsilon)D - \frac{\epsilon}{k} \cdot D + 2} - 1 = \Omega(\epsilon/k)$, provided $D = \Omega(k/\epsilon)$. Therefore, in $O(\frac{k}{\epsilon} \cdot \log n)$ rounds, the agents will get a token from $\{0, \ldots, \lceil (1+\epsilon) \rceil D - 1\}$.

Once the agents are assigned tokens, each agent compares its token with the demand vector to determine which task it is going to. Define the error $\epsilon_i = i \cdot \lfloor D \cdot \frac{\epsilon}{k} \rfloor$ for $0 \leq i \leq k$. Define the interval $I_j = [\frac{D_{j-1}+\epsilon_{j-1}}{N}, \frac{D_j+\epsilon_j}{N})$ for $1 \leq j \leq k$, where $D_j = \sum_{i=1}^{j} d_j$ and $N = D + \epsilon_k$ is a normalization term (which is actually not necessary for the token passing algorithm, but will be needed for the ranking algorithm). These intervals form a disjoint partition of $[0, 1)$. Let TK_a denote the token assigned to agent a. Let $X_a = \mathrm{TK}_a / N$. We show that if each agent a executes allocate_task(a, X_a) described in Algorithm 2, where a goes to task $j(a)$ such that $I_{j(a)}$ contains X_a, then the allocation is proper.

Algorithm 2 allocate_task(a, X_a)

```
Let $I_j = [\frac{D_{j-1}+\epsilon_{j-1}}{N}, \frac{D_j+\epsilon_j}{N})$, for $1 \leq j \leq k$, where $D_j = \sum_{i=1}^{j} d_j$,$\epsilon_j = j\lfloor \frac{\epsilon}{k} \cdot D \rfloor$, and $N = D + \epsilon_k$.
Let $I_{j(a)}$ be the interval that contains $X_a$.
Go to task $j(a)$.
```

Lemma 1. *Suppose that $n = \lceil (1+\epsilon)D \rceil - \lfloor \frac{\epsilon}{k} \cdot D \rfloor$ and each agent is assigned a unique token from $\{0, 1, \ldots, \lceil (1+\epsilon) \rceil D - 1\}$. If each agent a goes to work on task $j(a)$, where the interval $I_{j(a)}$ contains TK_a, then the allocation is proper.*

Proof. Since the length of the interval I_j is $\frac{d_j+\lfloor\frac{\epsilon}{k}\cdot D\rfloor}{N}$, I_j contains at least $d_j + \lfloor\frac{\epsilon}{k}\cdot D\rfloor$ tokens. However, since at most $\lceil(1+\epsilon)D\rceil - n = \lfloor\frac{\epsilon}{k}\cdot D\rfloor$ tokens are not taken by the agents, it contains at least $d_j + \lfloor\frac{\epsilon}{k}\cdot D\rfloor - \lfloor\frac{\epsilon}{k}\cdot D\rfloor = d_j$ tokens used by the agents. Therefore, at least d_j agents are working on task i.

When the demands change from $\boldsymbol{d}$ to $\boldsymbol{d}'$, there are two variants of the algorithms to deal with that. The first one is to continue to run allocate_task(a, X_a) with the same X_a. The second one is to update the token and X_a and then run allocate_task(a, X_a). We will show that the first one has a $(k-1)$-optimal switching cost, while the second one gives an optimal switching cost. However, the second one requires all the agents receive the same demand vectors in a consistent order. Therefore, unlike the first variant, it does not tolerate transient faults, since when an agent temporarily malfunctions, it is not able to receive input signals.

3.1 The First Variant

We will bound the switching cost when all the agents keep running Algorithm 2 without changing the values of X_a. The lemma is stated in a more general way so that we can also apply it later in the next section. The proof is deferred to the full version.

Lemma 2. *Suppose that the demands change from* $\boldsymbol{d}$ *to* $\boldsymbol{d}'$ *and the* X_a*-values of all the agents are fixed. Let* $\mathcal{X} = \{X_a\}_{a\in A}$ *be the multi-set that consists of all the* X_a*-values of the agents. Let* $\gamma(\mathcal{X}) = sup_{0\leq i\leq N-1}|X \cap [\frac{i}{N}, \frac{i+1}{N})|$ *denote the maximum number of agents whose* X_a*-value lie in the interval over all intervals of length* $\frac{1}{N}$*. The switching cost is bounded by* $\gamma\cdot(k-1)\cdot\mathrm{OPT}$*, where* $\mathrm{OPT} = |\boldsymbol{d}-\boldsymbol{d}'|_1/2$*.*

Since X_a is defined to be the token value divided by N and the token values are integers, we must have $\gamma(\mathcal{X}) = 1$. Therefore, the switching cost is at most $(k-1)\cdot\mathrm{OPT}$. Algorithm 2 is capped at this bound for the switching cost. An interesting question is whether there exists another scheme for achieving a better switching cost, perhaps by partitioning the $[0,1)$ interval in a better way. The algorithms where the X_a-values do not change can be captured by the following definition of stable algorithms. We will show that stable algorithms cannot achieve the optimal switching cost when $n = D$. (Note that Algorithm 2 does not fit into this case, because it uses more agents than D. See discussions in Section 5.)

Definition 1. *A stable task allocation algorithm is where each agent* a *is associated with a function* $f_a(d_1, d_2, \ldots, d_k)$ *such that agent* a *goes to* $(d_1, \ldots, d_k)$ *when the demand vector is* $(d_1, d_2, \ldots, d_k)$*.*

We show that the stable algorithms that achieve proper allocations must incur a switching cost of at least $2\cdot\mathrm{OPT}$ when $n = D$.

Lemma 3. *Suppose that* $n = D$ *and an algorithm is stable. Then, there exist demands* $\boldsymbol{d}$ *and* $\boldsymbol{d}'$ *such that the algorithm uses at least* $|d'-d|_1$ *switching cost when the demands change from* $\boldsymbol{d}$ *to* $\boldsymbol{d}'$*.*

Proof. Suppose there are 3 agents, a_1, a_2, a_3. Suppose to the contrary that $f_{a_1}, f_{a_2}, f_{a_3}$ are functions for a_1, a_2, and a_3 that achieve the optimal movements when there are 3 tasks with total demand 3. Suppose that the initial demand is $\boldsymbol{d}_1 = (1,1,1)$. Without loss of generality, suppose that $(f_{a_1}(\boldsymbol{d}_1), f_{a_2}(\boldsymbol{d}_1), f_{a_3}(\boldsymbol{d}_1)) = (1,2,3)$. When the demands change to $\boldsymbol{d}_2 = (1,2,0)$, since we assume the strategy achieves the optimal movement, we must have $(f_{a_1}(\boldsymbol{d}_1), f_{a_2}(\boldsymbol{d}_1), f_{a_3}(\boldsymbol{d}_1)) = (1,2,2)$. If the demands again change to $\boldsymbol{d}_3 = (0,2,1)$, then we must have $(f_{a_1}(\boldsymbol{d}_1), f_{a_2}(\boldsymbol{d}_1), f_{a_3}(\boldsymbol{d}_1)) = (3,2,2)$ by the same reasoning. Finally, if the demands again change back to $\boldsymbol{d}_1 = (1,1,1)$, then by the same reasoning, we have either $(f_{a_1}(\boldsymbol{d}_1), f_{a_2}(\boldsymbol{d}_1), f_{a_3}(\boldsymbol{d}_1)) = (3,2,1)$ or $(f_{a_1}(\boldsymbol{d}_1), f_{a_2}(\boldsymbol{d}_1), f_{a_3}(\boldsymbol{d}_1)) = (3,1,2)$, contradicting with the fact that $f_{a_1}, f_{a_2}, f_{a_3}$ are functions.

3.2 The Second Variant

If we do not require the algorithm to be stable, then it is still possible to achieve an optimal switching cost. In the second variant, we will reassign TK_a (and set $X_a = \mathrm{TK}_a/N$) when the demands change. We will pretend there are dummy agents such that all tokens are used up. The agents reassign the tokens according to the following rules.

Suppose that a is working on a task $j(a)$. If $d_{j(a)} > d'_{j(a)}$ and a is an agent holding one of the largest $d_{j(a)} - d'_{j(a)}$ tokens among the agents working on $j(a)$, then a will switch tasks. Let A' denote the set of all agents that belong to this case. Each $a \in A'$ will use $\mathrm{rank}_{A'}(a)$ and the deficits of the tasks to update its token and switch to the corresponding task. Otherwise, if a does not belong to the case described above, it will not switch tasks. However, it will also reassign its token to avoid conflict. The details are postponed to the full version, where we will also show that after updating the tokens, each token in $\{0, 1, \ldots, \lceil (1+\epsilon)D \rceil - 1\}$ is assigned to at most one agent.

All the tokens held by the agents (including dummy agents) are distinct after updating. Now if we delete the dummy agents, all the tokens are still distinct. By Lemma 1, we conclude that the allocation obtained is proper. Furthermore, for task i, $d_i - d'_i$ agents switch tasks if $d_i > d'_i$. The switching cost is $\sum_i \max(0, d_i - d'_i) = |d' - d|_1/2 = \mathrm{OPT}$.

4 The Ranking Algorithm

In this section, we assume that $\lceil (1+\epsilon) \cdot D \rceil \le n \le 2D$ for some $0 < \epsilon < 1$. Let $\mathrm{rank}(a)$ denote the rank of ID_a among all the ID of other agents. We assume the rank begins with 0, i.e. the rank of the agent with the smallest ID is 0. Let $\mathrm{nrank}(a) = \mathrm{rank}(a)/n$ denote the *normalized rank* of a.

We give two variants of algorithms for approximating the normalized ranks of the agents. By setting X_a to be the estimated normalized rank of a and running allocate_task(a, X_a) described in Algorithm 2, we show that the allocation is proper if the estimates are accurate enough. Recall that in Algorithm 2, we partition the entire working space $[0,1)$ into half-open intervals $I_j = [\frac{D_{j-1}+\epsilon_{j-1}}{N}, \frac{D_j+\epsilon_j}{N})$, for $1 \le j \le k$, where $D_j = \sum_{i=1}^{j} d_j$, $\epsilon_j = j \cdot \lfloor \frac{\epsilon}{k} \cdot D \rfloor$, and $N = D + \epsilon_k$. If X_a lies in the interval $I_{j(a)}$, then it will go to task $j(a)$. We show that if each agent has a sufficiently good estimate of its own rank, then the allocation obtained is proper.

Lemma 4. *Suppose that* $X_a \in [\mathrm{nrank}(a) - \epsilon/(6k), \mathrm{nrank}(a) + \epsilon/(6k)]$ *for each* a*, then for each task* j*, there are at least* d_j *agents working on it. That is, the allocation is proper.*

Proof. Consider the interval $I_j = [\frac{D_{j-1}+\epsilon_{j-1}}{N}, \frac{D_j+\epsilon_j}{N})$. The length of the interval is $(d_j + \lfloor \frac{\epsilon}{k} \cdot D \rfloor)/N$. Consider the half-open interval $I'_j \subseteq I_j$ obtained by removing the first $\epsilon/(6k)$ and the last $\epsilon/(6k)$ of I_j. The length of I'_j is at least

$$\begin{aligned}
\frac{d_j}{N} + \left\lfloor \frac{\epsilon D}{k} \right\rfloor \cdot \frac{1}{N} - \frac{\epsilon}{3k} &\ge \frac{d_j}{N} + \left(\frac{\epsilon D}{k} - 1\right) \cdot \frac{1}{N} - \frac{\epsilon}{3k} \\
&\ge \frac{d_j}{N} + \frac{\epsilon}{2k} - \frac{1}{N} - \frac{\epsilon}{3k} && N \le 2D \\
&\ge \frac{d_j}{N} + \frac{\epsilon}{6k} - \frac{1}{N} \ge \frac{d_j}{N} + \frac{1}{N} && D \ge \frac{12k}{\epsilon}
\end{aligned}$$

Since the smallest normalized rank in I'_j must appear in the first $1/N$ segment from the beginning of the interval, the number of agents whose normalized rank lie in I'_j is at least $n \cdot (\frac{d_j}{N} + \frac{1}{N}) - 1 \ge d_j + 1 - 1 \ge d_j$, since $n \ge N$. Moreover, if $\mathrm{nrank}(a) \in I'_j$, then its estimate X_a must be in I_j, since $X_a \in [\mathrm{nrank}(a) - \epsilon/(6k), \mathrm{nrank}(a) + \epsilon/(6k)]$. Because there are at least d_j agents whose normalized rank lie in I'_j, there are at least d_j agents whose X_a-values lie in I_j. Therefore, the number of agents working on task j is at least d_j.

4.1 The First Variant

In this section, we show how to approximate the normalized rank up to an additive $\pm\epsilon/(6k)$ factor in $O((\log^2 n)/\epsilon)$ rounds. Moreover, w.h.p. the estimated rank for each agent is the same for different executions of the algorithm. The resulting task allocation algorithm is therefore stable.

The algorithm works as follow: First, count the total number of agents, n, in $O(\log n)$ rounds. Then, identify the $O(k/\epsilon)$ *pivot* agents whose ranks are $0, \lfloor \frac{\epsilon n}{6k} \rfloor, 2 \cdot \lfloor \frac{\epsilon n}{6k} \rfloor, \ldots, \lceil \frac{6k}{\epsilon} \rceil \cdot \lfloor \frac{\epsilon n}{6k} \rfloor$. This can be done in $O((\log^2 n)/\epsilon)$ rounds by running $O(k)$ selection algorithms of [14] in parallel, using $O(k \log n)$ message size. Then, the pivot agents broadcast their IDs and the normalized ranks to everyone in $O(\log n)$ rounds. Each agent a sets its estimate X_a to be the normalized rank of the pivot agent with the largest ID smaller than its own (that is, rounding down).

Lemma 5. *After running the algorithm described above, we have* $X_a \in [\mathrm{nrank}(a) - \epsilon/(6k), \mathrm{nrank}(a)]$.

Proof. The normalized ranks between two consecutive pivot agents is $\lfloor \frac{n}{6k} \rfloor$. Therefore, we must have $X_a \ge \mathrm{nrank}(a) - \lfloor \frac{\epsilon n}{6k} \rfloor / n \ge \mathrm{nrank}(a) - \frac{\epsilon}{6k}$.

For the switching cost of this algorithm, it is not hard to see that $\gamma(\mathcal{X}) = \lfloor \frac{\epsilon n}{6k} \rfloor$ (where $\gamma(\mathcal{X})$ is defined in Lemma 2). Therefore, by Lemma 2, the switching cost is at most $O(\epsilon n) \cdot \mathrm{OPT}$. This implies the switching cost can be pretty large in this algorithm. Note that the algorithm is robust to transient fault after X_a-values are computed. Because like the first variant of the token passing algorithm, once the X_a-values are computed, as long as the agent gets the correct demand vector, it can allocate itself to the right task without any communication.

4.2 The Second Variant

In this section, we present a fault-tolerant algorithm, based on a simple approach to approximate the ranks. The algorithm can be implemented in an asynchronous manner, where each agent maintains its own clock. In that setting, the t'th global round is defined to be the earliest time when every agent completed its t'th round.

Algorithm 3 Sampling-Based Rank Estimation

```
for each round $t$ do
  for each agent $a$ do
    Let $a'$ be the agent $a$ met during round $t$.
    Let $X_t \leftarrow \begin{cases} 1, \text{ if } \mathrm{ID}_{a'} < \mathrm{ID}_a . \\ 0, \text{ otherwise.} \end{cases}$ .
    Let $T = \Theta((\frac{k}{\epsilon})^2 \log n)$.
    Let $X_a = \sum_{i=t-T+1}^{t} X_i / T$.
    allocate_task($a$, $X_a$)
  end for
end for
```

Lemma 6. *Suppose that agent a finishes* $T = \Theta((\frac{k}{\epsilon})^2 \log n)$ *rounds after its last transient fault. Then, w.h.p.* $X_a \in [\mathrm{nrank}(a) - \frac{\epsilon}{6k}, \mathrm{nrank}(a) + \frac{\epsilon}{6k}]$.

Therefore, by Lemma 6 and Lemma 4, after every agent finishes its T'th round, the allocation is proper. However, since they keep updating their X_a-values after T'th round in order to cope with the crash faults (see the next subsection), the switching cost can fairly large ($\tilde{\Omega}(n)$) even if the demands do not change. In the following, we show that if the agents stop updating the X_a-values after T'th round, then they can achieve a bounded switching cost of $O(k \log n) \cdot \mathrm{OPT}$. By Lemma 2, it suffices to show $\gamma(\mathcal{X}) = O(\log n)$.

Lemma 7. *After* $\Omega((\frac{k}{\epsilon})^2 \log n)$ *rounds, w.h.p.* $\gamma(\mathcal{X}) = O(\log n)$ *(Recall that* $\gamma(\mathcal{X}) = sup_{0 \le x \le n-1} \sum_{X_a \in \mathcal{X}} |X_a \cap [\frac{x}{N}, \frac{x+1}{N})|$*).*

Proof. For any $0 \le x \le N-1$, consider the interval $[\frac{x}{N}, \frac{x+1}{N}) \subseteq [0, 1)$. For $-D \le i \le D$, let A_i be the set of agents whose normalized ranks lies in $[\frac{x+i}{N}, \frac{x+i+1}{N})$

(note that there are at most n/N such agents). Let Y_a be the indicator random variable denoting whether a lies in $[\frac{x}{N}, \frac{x+1}{N})$. Let $Y = \sum_{-D \leq i \leq D} Y_a$. For $a \in A_i$, it is not hard to see that $\Pr(Y_a = 1) \leq 1/(i+1)$. Therefore,

$$\mathbb{E}[Y] = \sum_{-D \leq i \leq D} \sum_{a \in A_i} \mathbb{E}[Y_a] \leq 2 \cdot \sum_{0 \leq i \leq D} \frac{n}{N} \cdot \frac{1}{(i+1)} = O(\log n)$$

Since Y is a sum of independent variables, by Chernoff Bound, for some constant $K > 0$, $\Pr(Y \geq K \cdot \log n) \leq 1/\operatorname{poly}(n)$. By taking an union bound the intervals $[\frac{x}{N}, \frac{x+1}{N})$ for $x = 0, 1, \ldots, N-1$, we conclude that w.h.p. $\gamma(\mathcal{X}) < 2K \log n$.

Fault Tolerance for Crash Faults We show that our algorithm is resilient to crash failures. Now suppose there are at most f agents who died in the previous T rounds. We will show that if f is sufficiently small, then the current allocation is a proper allocation w.h.p. On the other hand, if f is large, then we give a bound on the number of rounds needed to recover to a proper allocation. The proofs of the following two lemmas are postponed to the full version.

Lemma 8. *Let $T = \Theta((\frac{k}{\epsilon})^2 \log n)$. Let f denote the number of agents who crashed during rounds $t - T + 1, t - T + 2, \ldots, t$. Suppose that $f = O(\epsilon n/k)$, then w.h.p. $X_a \in [\operatorname{nrank}'(a) - \frac{\epsilon}{12k}, \operatorname{nrank}'(a) + \frac{\epsilon}{12k}]$, where $\operatorname{nrank}'(a)$ denote the normalized rank of a at round t.*

Lemma 9. *Let $T = \Theta((k/\epsilon)^2 \log n)$. Let f denote the number of agents who died during the past T rounds. Suppose that $f = \Omega(\epsilon n/k)$, then the allocation becomes proper after $(1 - \frac{\epsilon n}{12fk}) \cdot T$ rounds, if no more failures happen.*

5 Open Problems Motivated by This Work

- **The role of extra agents.** In our token passing algorithm and ranking algorithm, extra agents are needed to achieve a logarithmic running time. An interesting question is whether the extra agents are really necessary to achieve that. Also, in stable algorithms, although we showed that the switching cost is at least 2-optimal when $n = D$, the existence of extra agents helps reducing the switching cost. For example, when there are kD agents, we can achieve 0 switching cost by allocating D agents to every task. Studying the trade-off between the number of agents and the switching cost seems to be an interesting direction.
- **The switching cost gap of stable algorithms.** We showed that stable algorithms cannot achieve the optimal switching cost (they must be at least 2-optimal). On the other hand, if all agents have their X_a-values properly assigned, then Algorithm 2 can achieve a switching cost that is $(k-1)$-optimal. There is still a large gap between a factor of $(k-1)$ and a factor 2. Closing this gap is a very interesting open problem. Our bounds are tight when the number of tasks is three. Our partition scheme shows that $2 \cdot \mathrm{OPT}$ is

achievable, while our lower bound shows that this is the best possible. In fact, we have partial results showing that for $D \leq 6$, we can achieve a switching cost of $2 \cdot \mathrm{OPT}$. For $D > 6$, we could not generalize the pattern and therefore it is yet to be investigated.

References

1. D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
2. S. N. Beshers and J. H. Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46(1):413–440, 2001.
3. E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proc. of the Royal Society of London B: Biological Sciences*, 263(1376):1565–1569, 1996.
4. E. Bonabeau, G. Theraulaz, and J.-L. Deneubourg. Fixed response thresholds and the regulation of division of labor in insect societies. *Bulletins of Mathematical Biology*, 60:753–807, 1998.
5. A. Cornejo, A. R. Dornhaus, N. A. Lynch, and R. Nagpal. Task allocation in ant colonies. In *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 46–60, 2014.
6. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. 6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.
7. M. L. Dertouzos and A. K. Mok. Multiprocessor online scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.
8. A. Dornhaus and N. Franks. Individual and collective cognition in ants and other insects (hymenoptera: Formicidae). *Myrmecological News*, 11:215–226.
9. A. Drucker, F. Kuhn, and R. Oshman. The communication complexity of distributed task allocation. In *Proc. 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 67–76, 2012.
10. O. Feinerman and A. Korman. The ANTS problem. *Distributed Computing. to appear*. Extended abstracts appeared in PODC 2012 (together with Z. Lotker and J.S. Sereni) and in DISC 2012.
11. A. M. Frieze and G. R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.
12. B. P. Gerkey and M. J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.
13. M. Ghaffari, C. Musco, T. Radeva, and N. A. Lynch. Distributed house-hunting in ant colonies. In *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 57–66, 2015.
14. G. Giakkoupis, A. Kermarrec, and P. Woelfel. Gossip protocols for renaming and sorting. In *Proc. 27th Symposium on Distributed Computing (DISC)*, pages 194–208, 2013.
15. D. M. Gordon. The organization of work in social insect colonies. *Complexity*, 8(1):43–46, 2002.

16. D. M. Gordon, B. C. Goodwin, and L. Trainor. A parallel distributed model of the behaviour of ant colonies. *J. of Theo. Biology*, 156(3):293–307, 1992.
17. M. J. Greene and D. M. Gordon. Interaction rate informs harvester ant task decisions. *Behavioral Ecology*, 18(2):451–455, 2007.
18. W. O. Hughes, S. Sumner, S. V. Borm, and J. J. Boomsma. Worker caste polymorphism has a genetic basis in acromyrmex leafcutting ants. *Proceedings of the National Academy of Sciences*, 100(16):9394–9397, 2003.
19. D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *IEEE 44th Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.
20. S. Kozub. Amazons new drone delivery plan includes package parachutes. *The Verge*, 2017.
21. T. Langner, D. Stolz, J. Uitto, and R. Wattenhofer. Fault-Tolerant ANTS. In *28th International Symposium on Distributed Computing (DISC)*, pages 31–45, 2014.
22. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.
23. L. Liu and D. A. Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33(3):291–307, 2012.
24. C. Musco, H. Su, and N. A. Lynch. Ant-inspired density estimation via random walks: Extended abstract. In *Proc. 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 469–478, 2016.
25. S. W. Pacala, D. M. Gordon, and H. C. J. Godfray. Effects of social group size on information transfer and task allocation. *Evolutionary Ecology*, 10(2):127–165, 1996.
26. H. M. Pereira and D. M. Gordon. A trade-off in task allocation between sensitivity to the environment and response time. *J. of Theo. Bio.*, 208(2):165–184, 2001.
27. T. Radeva. *A Symbiotic Perspective on Distributed Algorithms and Social Insects*. Dissertation, Massachusetts Institute of Technology, 2017.
28. T. Radeva, A. Dornhaus, N. Lynch, R. Nagpal, and H.-H. Su. Costs of task allocation with local feedback: Effects of colony size and extra workers in social insects and other multi-agent systems. *submitted*. Preliminary version appeared as a brief announcement in *Proc. 28th Symposium on Distributed Computing (DISC)*, pages 657–658, 2014.
29. G. E. Robinson. Regulation of division of labor in insect societies. *Annual Review of Entomology*, 37(1):637–665, 1992.
30. A. B. Sendova-Franks and N. R. Franks. Spatial relationships within nests of the ant leptothorax unifasciatus (latr.) and their implications for the division of labour. *Animal Behaviour*, 50(1):121–136, 1995.
31. F. Tripet and P. Nonacs. Foraging for work and age-based polyethism: The roles of age and previous experience on task choice in ants. *Ethology*, 110(11):863–877, 2004.
32. E. O. Wilson. Caste and division of labor in leaf-cutter ants (hymenoptera: Formicidae: Atta). *Behavioral Ecology and Sociobiology*, 7(2):157–165, 1980.