

Ant-Inspired Dynamic Task Allocation via Gossiping *

Hsin-Hao Su¹, Lili Su², Anna Dornhaus³, and Nancy Lynch⁴

1 CSAIL, MIT, Cambridge MA, USA

2 ECE, University of Illinois at Urbana-Champaign, Urbana IL, USA

3 Ecology and Evolutionary Biology, University of Arizona, Tucson AZ, USA

4 CSAIL, MIT, Cambridge MA, USA

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 ant colonies, we propose several scalable and efficient algorithms to *dynamically* allocate the agents *as the task demands vary*. Similar to ants, in our algorithms, each agent obtains sufficient information to make its local decision by gossiping with the other ants. Our algorithms vary in their advantages and disadvantages, 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.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.2.11 Distributed Artificial Intelligence

Keywords and phrases ant colonies, task allocation, dynamic algorithms, gossip protocol, fault tolerant

Digital Object Identifier 10.4230/LIPIcs...

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 [19]. The drones may learn the demands in each region from a broadcasting ground control station. The demands may vary from time to time. The drones are required to coordinate among themselves, 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 [28]) which require different number of ants. Ant colonies generally do a good job of regulating the assignment of workers to tasks. In this work, we take inspiration from 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.

* Regular paper submission, non-student. Contact author: Hsin-Hao Su, G606 32 Vassar St., Cambridge, MA 02139, USA. Email: hsinhao@csail.mit.edu



© Hsin-Hao Su, Lili Su, Anna Dornhaus, and Nancy Lynch;
licensed under Creative Commons License CC-BY

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To model the task allocation without centralized controllers, we consider randomized *gossip protocols* [6] (sometimes known as *population protocols*¹ [1]) as the underlying method of communication among the agents. In short, randomized gossip protocols consist of *rounds*. In each round, each agent chooses another 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 [14]. The gossip protocol model captures the way they exchange information in a peer-to-peer manner. 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, $ID_a \leq \text{poly}(n)$, and a state $Q_a \in \{1, 2, \dots, k\}$, which indicates the task that it is working on. (Note that in ant colonies, 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 $\vec{d}^{(t)} = (d_1^{(t)}, d_2^{(t)}, \dots, d_k^{(t)})$ from the tasks, where $d_i^{(t)}$ indicates the demand of task i ². Note that 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 fit the size of the input signals). Then, the agents can change their states. Then they proceed to the next round.

Cornejo et al. [5] and Radeva et al. [26] defined a model 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*.

1.2 Problem Formulation

We formulate the task allocation problem similarly to [26] and [5] as follows. Let $A_i^{(t)}$ denote the set of agents working on task i for $1 \leq i \leq k$. Let $\vec{w}^{(t)} = (w_1^{(t)}, w_2^{(t)}, \dots, w_k^{(t)})$ denote the number of workers 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 $i \in \{1, 2, \dots, k\}$. 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.

There are several objectives we consider for an algorithm. First, whenever the demands change, we hope that the allocation recovers 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. When the demands change from \vec{d} to \vec{d}' , it is clear that the switching cost is at least $\text{OPT} \stackrel{\text{def}}{=} |\vec{d} - \vec{d}'|_1 / 2$ (if the work exactly matches

¹ Although gossip-based protocols and population protocols are slightly different. For example, the latter usually have a more restrictive memory constraint.

² The demands should be thought as the work-rates required to keep the tasks satisfied.

■ **Table 1** Comparisons of the Algorithms.

	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		$O(\frac{k}{\epsilon} \log n)$	$O(\frac{k}{\epsilon} \log n)$	$O(\frac{1}{\epsilon} \log^2 n)$	$O(\left(\frac{k}{\epsilon}\right)^2 \log n)$
Realloc. Time	$O(\log^2 n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Switching Cost	OPT	$(k - 1) \cdot \text{OPT}$	OPT	$O(n)$	$O(k \log n) \cdot \text{OPT}$ (or $O(n)$)
Fault Tolerance		transient faults after preproc.		transient faults after preproc.	transient & (crash) no global clock

the demand for each task). 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 crashed, as long as there are enough remaining agents.

1.3 Our Contribution

The main contribution of our work is to explore different possibilities that can be achieved by a(ge)nts under the gossip model where information exchanges are limited. We give three different types of algorithms for the task allocation problem. They 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 similar to the algorithm of Radeva et al. [26], 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.³ using the gossip-based counting and selection algorithms developed in [18]. The main advantage of the 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 $O(\log^2 n)$ 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 stimulation of chemicals. Biologist have conjectured that different ants have different response thresholds to the signals [3]. 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

³ With high probability, which means with probability at least $1 - 1/\text{poly}(n)$.

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 lying in the interval.

In the **token passing algorithm**, each agent is assigned a unique token X_a from $\{1, 2, \dots, n + \lfloor \frac{\epsilon}{k} \cdot D \rfloor\}$ in the preprocessing phase. This is done by using the loose renaming procedure developed by Giakkoupis et al. [13]. 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 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 $(k - 1) \cdot \text{OPT}$. 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 $2 \cdot \text{OPT}$. In comparison, our first variant achieves a switching cost of $(k - 1) \cdot \text{OPT}$. 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. $\text{rank}(a)/n$) of a , where $\text{rank}(a)$ is the rank of a 's ID over all the agents. In fact, the algorithm is more similar to ants' behavior. The ID of each agent can be thought as some features of the agents. In ant colonies, ants allocate the tasks based on their individual traits such as age [28, 30], body size [31], genetic background [17]. For example, there are tendencies for older ants to go foraging and for younger ants to work on the tasks that are closer to the nests [30]. The ranking algorithms mimic this in the way that if the demands are fixed, then the allocation for every agent is determined by the relative position 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 light-weight 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. 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 $O(k \log n) \cdot \text{OPT}$ 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 [28], body size [31], genetic background [17], position in the nest [29], social interaction [16], and internal response thresholds [4]. There are also works that formulate the task allocation problems using mathematical models [2, 4, 15, 24, 25]. Most of the works focus on the dynamics of workers and studied the convergence of the dynamics rather than on the individual algorithmic mechanisms.

Cornejo et al. [5] was the first to model the ant task allocation problem from a distributed computing perspective. Then, [26] studied how extra agents can speed up the task allocation process. In the ant-colony task allocation models of [26] 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. $\text{sgn}(w_i - d_i)$). Biologists claimed that it is not clear what the actually signals the ants are receiving in reality, and they may depend on the type of the tasks. The reason we use the demands as signals rather than the (functions of) deficits is because that it gives a clear separation between the environment and the communication. For example, if agent a learns the deficit of task i , there is an information flow from the agents working on task i to a . In our model, such an information flow only happens through explicit communication. Moreover, since the agents can do counting (c.f. Section 2), the demands can be converted to the deficits (and vice versa) with an $O(\log n)$ delay in our model.

In computer science, task allocation problems have also been well-studied under various contexts, including scheduling of multiprocessors [7, 21], robotics [22, 11], and communication complexity [8]. 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 as 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 [20, 9], the house-hunting problem [12], and density estimation problem [23]. 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 fundamental open problems inspired by this work.

2 The Move-and-Fill Algorithm

We first present an algorithm that does not require extra 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 [10, 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 [18, Algorithm 1] approximates the summation up to $(1 \pm \epsilon)$ 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 $\delta = 1/\text{poly}(n)$, 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 $\text{rank}_{A'}(a)$ denote the rank of a in the set A' , *beginning with 0*. We explain that in $O(\log^2 n)$ rounds, w.h.p. each agent a in A' can determine whether $\text{rank}_{A'}(a)$ is at least r or not. In Kempe et al. [18, Theorem 4.2], they 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 $\text{rank}_{A'}(a) = r$. 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, \dots, w_k by counting the number of agents working on each task.

For each $a \in A_i$, include a in A' if $\text{rank}_{A_i}(a) \geq d_i$.

Let $\phi_i = \max(d_i - w_i, 0)$ be the deficit of task i .

For $1 \leq i \leq k$, let $\Phi_i = \sum_{j=1}^i \phi_j$ 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 $1 \leq i \leq k$.

For each $a \in A'$, go to task $i(a)$, where $I_{i(a)}$ is the interval that contains $\text{rank}_{A'}(a)$.

We assume $n = D$. The **move-and-fill algorithm** is described in Algorithm 1. The algorithm is similar to Radeva et al.'s algorithm [26], 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 $\text{rank}_{A_{Q_a}}(a) \geq d_{Q_a}$. Such a test can be done for every agent by running k rank testing algorithms (one for each task) in parallel. 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 $\text{rank}_{A'}(a)$ lying in the interval I_i . Thus, ϕ_i agents will go to task i . This implies all the tasks become satisfied. Again, such a test can be done for every agent by running k rank testing algorithm to test if $\text{rank}_{A'}(a) \geq \Phi_i$ for each i .

The number of rounds needed to get to a proper allocation is $O(\log^2 n)$. Suppose that the demands change from \vec{d} to \vec{d}' . It is clear that the algorithm achieves an optimal switching cost of $\text{OPT} = |\vec{d} - \vec{d}'|_1/2$, since the number of agents who switched tasks is $\sum_{i=1}^k \max(0, d'_i - w_i) = \sum_{i=1}^k \min(0, d'_i - d_i) = |\vec{d}' - \vec{d}|_1/2$.

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 $\{0, \dots, \lceil(1 + \epsilon)D\rceil - 1\}$ such that each token is assigned to at most one agent. Giakkoupis et al. [13] gave an algorithm for the renaming problem in the gossip model that assigns a name from the name space $\{1, 2, \dots, (1 + \epsilon')n\}$ 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 $\{0, \dots, \lceil(1 + \epsilon)D\rceil - 1\}$.

Once the agents are assigned tokens, the allocation is obtained by comparing its token with the demand vector. 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 $N = D + \epsilon_k$ 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 $X_a = \text{TK}_a / N$. We show that if each agent a executes $\text{allocate_task}(a, X_a)$ 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 $\text{allocate_task}(a, X_a)$

Let $I_j = [\frac{D_{j-1} + \epsilon_{j-1}}{N}, \frac{D_j + \epsilon_j}{N})$, for $1 \leq j \leq k$, where $D_j = \sum_{i=1}^j d_i, \epsilon_j = j \lfloor \frac{\epsilon}{k} \cdot D \rfloor$, and $N = D + \epsilon_k$.

Let $I_{j(a)}$ be the interval where X_a is.

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 $\{0, 1, \dots, \lceil(1 + \epsilon)D\rceil - 1\}$. 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. The length of the interval I_j is $\frac{d_j + \lfloor\frac{\epsilon}{k} \cdot D\rfloor}{N}$. Therefore, it supposedly 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 \vec{d} to \vec{d}' , there are two variants of the algorithms to deal with that. The first one is to continue to run $\text{allocate_task}(a, X_a)$ with the same X_a . The second one is to update the token and X_a and then run $\text{allocate_task}(a, X_a)$. 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 tolerant 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.

► **Lemma 2.** *Suppose that the demands change from \vec{d} to \vec{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 $\gamma(\mathcal{X}) = \sup_{0 \leq i \leq N-1} \sum_{X_a \in \mathcal{X}} |X_a \cap [\frac{i}{N}, \frac{i+1}{N})|$ 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 $\gamma \cdot (k - 1) \cdot \text{OPT}$, where $\text{OPT} = |\vec{d} - \vec{d}'|_1/2$.*

Proof. First, let $m = |\vec{d} - \vec{d}'|_1/2$. It is not hard to see that there exists a sequence of demand vectors $\vec{d}^{(0)} = \vec{d}, \vec{d}^{(1)}, \vec{d}^{(2)}, \dots, \vec{d}^{(m)} = \vec{d}'$ such that for $1 \leq i \leq m$, $|\vec{d}^{(i-1)} - \vec{d}^{(i)}|_1 = 2$.

Let $\mathcal{I}^{(i-1)} = \{I_j^{(i-1)}\}_{1 \leq j \leq k}$ (and $\mathcal{I}^{(i)} = \{I_j^{(i)}\}_{1 \leq j \leq k}$) be the set of intervals defined in Algorithm 2 with respect to the demand $\vec{d}^{(i-1)}$ (and $\vec{d}^{(i)}$, respectively). Let $\Delta(\mathcal{I}^{(i-1)}, \mathcal{I}^{(i)}) = \bigcup_{j=1}^k (I_j^{(i-1)} \setminus I_j^{(i)}) \cup (I_j^{(i)} \setminus I_j^{(i-1)})$. Note that only those agents whose X_a -values lie in $\Delta(\mathcal{I}^{(i-1)}, \mathcal{I}^{(i)})$ will switch tasks. It is not hard to see that $\Delta(\mathcal{I}^{(i-1)}, \mathcal{I}^{(i)})$ is the union of at most $(k - 1)$ half-open intervals of length $1/n$, since $|\vec{d}^{(i-1)} - \vec{d}^{(i)}|_1 = 2$. By definition of γ , there are at most $\gamma \cdot (k - 1)$ agents whose X_a -values lie in $\Delta(\mathcal{I}^{(i-1)}, \mathcal{I}^{(i)})$. Therefore, the switching cost is $\gamma \cdot (k - 1) \cdot |\vec{d}^{(i-1)} - \vec{d}^{(i)}|_1/2$ if the demand changes from $\vec{d}^{(i-1)}$ to $\vec{d}^{(i)}$. Therefore, the total switching cost is $\gamma \cdot (k - 1) \cdot |\vec{d} - \vec{d}'|_1/2 = \gamma \cdot (k - 1) \cdot \text{OPT}$. ◀

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 $(k - 1) \cdot \text{OPT}$. 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 $n = D$.

► **Definition 3.** A stable task allocation algorithm is where each agent a is associated with a function $f_a(d_1, d_2, \dots, d_k)$ such that agent a goes to (d_1, \dots, d_k) when the demand vector is (d_1, d_2, \dots, d_k) .

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

► **Lemma 4.** *Suppose that an algorithm is stable. Then there exists demands \vec{d} and \vec{d}' such that the algorithm uses at least $|\vec{d} - \vec{d}'|_1$ switching cost when the demands change from \vec{d} to \vec{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 $\vec{d}_1 = (1, 1, 1)$. Without loss of generality, suppose that $(f_{a_1}(\vec{d}_1), f_{a_2}(\vec{d}_1), f_{a_3}(\vec{d}_1)) = (1, 2, 3)$. When the demands change to $\vec{d}_2 = (1, 2, 0)$, since we assume the strategy achieves the optimal movement, we must have

$(f_{a_1}(\vec{d}_1), f_{a_2}(\vec{d}_1), f_{a_3}(\vec{d}_1)) = (1, 2, 2)$. If the demands again change to $\vec{d}_3 = (0, 2, 1)$, then we must have $(f_{a_1}(\vec{d}_1), f_{a_2}(\vec{d}_1), f_{a_3}(\vec{d}_1)) = (3, 2, 2)$ by the same reasoning. Finally, if the demands again change back to $\vec{d}_1 = (1, 1, 1)$, then by the same reasoning, we have either $(f_{a_1}(\vec{d}_1), f_{a_2}(\vec{d}_1), f_{a_3}(\vec{d}_1)) = (3, 2, 1)$ or $(f_{a_1}(\vec{d}_1), f_{a_2}(\vec{d}_1), f_{a_3}(\vec{d}_1)) = (3, 1, 2)$, contradicting with the fact that $f_{a_1}, f_{a_2}, f_{a_3}$ are functions. \blacktriangleleft

While we have shown that stable algorithms cannot achieve the optimal switching cost. However, it is still unknown how good can the switching cost be for stable algorithms. In fact, even showing the switching cost is $\omega(1) \cdot \text{OPT}$ or $o(k) \cdot \text{OPT}$ seems to be non-trivial.

3.2 The Second Variant

If we consider outside the scope of stable algorithms, then it is still possible to achieve an optimal switching cost. In the second variant, we will reassign TK_a (and set $X_a = \text{TK}_a / N$) when the demands change. In the following, we will pretend as if there are dummy agents such that all tokens are used up. The agents reassign the tokens according to the following rules. There are two cases, depending on whether agent a is going to switch task or not.

- Suppose that a is working on a task $j(a)$ such that the demand of $j(a)$ decreases. Then $d_{j(a)} - d'_{j(a)}$ agents will pop out and go to the other tasks. If X_a belongs to the last $(d_{j(a)} - d'_{j(a)})/N$ segment of $I_{j(a)}$ (i.e. $\text{TK}_a \in [\sum_{i=1}^{j(a)-1} d_i + d'_{j(a)} + \epsilon_{j(a)}, \sum_{i=1}^{j(a)} d_i + \epsilon_{j(a)})$), then it will pop out.

Let A' denote the set of all agents belong to this category. In fact, a can figure out what $\text{rank}_{A'}(a)$ is by using only the information from \vec{d} and \vec{d}' . In particular,

$$\text{rank}_{A'}(a) = \sum_{i=1}^{j(a)-1} \max(0, d_i - d'_i) + \text{TK}_a - \left(\left(\sum_{i=1}^{j(a)-1} d_i \right) + d'_{j(a)} + \epsilon_{j(a)} \right)$$

Then, they will use $\text{rank}_{A'}(a)$ and the deficits of the tasks $\phi_i \stackrel{\text{def}}{=} \max(0, d'_i - d_i)$ to update its token. Define the interval $J_i = [\sum_{j=1}^{i-1} \phi_j, \sum_{j=1}^i \phi_j]$. Let $j'(a)$ be the interval such that $\text{rank}_{A'}(a) \in J_{j'(a)}$. It will update its token to be

$$\text{TK}_a \leftarrow \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d_{j'(a)} + \epsilon_{j'(a)} + \text{rank}_{A'}(a) - \left(\sum_{i=1}^{j'(a)-1} \phi_i \right)$$

- Otherwise, a will not switch task. However, a still needs to reassign its token. For $1 \leq i \leq k$, define the offset $\Delta_i = \sum_{j=1}^{i-1} (d'_j - d_j)$. If agent a is working on task i , it will set $\text{TK}'_a \leftarrow \text{TK}_a + \Delta_i$.

We show that after updating the tokens, for each token in $\{0, 1, \dots, \lceil (1 + \epsilon)D \rceil - 1\}$, it is assigned to at most one agent. The proof of the following lemma is deferred to the appendix.

► **Lemma 5.** *If $\text{TK}_a \neq \text{TK}_{a'}$ for any $a \neq a'$, then $\text{TK}'_a \neq \text{TK}'_{a'}$ for any $a \neq a'$. Moreover, $\text{TK}'_a \in \{0, 1, \dots, \lceil (1 + \epsilon)D \rceil - 1\}$.*

By Lemma 5, 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, the switching cost is $\sum_i \max(0, d_i - d'_i) = |d' - d|_1 / 2 = \text{OPT}$.

4 The Ranking Algorithm

In this section, we assume that $\lceil (1 + \epsilon) \cdot D \rceil \leq n \leq 2D$ for some $0 < \epsilon < 1$. Let $\text{rank}(a)$ denote the order 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 $\text{nank}(a) = \text{rank}(a)/n$ 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, Xa)` 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 $1 \leq j \leq k$, where $D_j = \sum_{i=1}^j d_i$, $\epsilon_j = j \cdot \lfloor \frac{\epsilon}{k} \cdot D \rfloor$, and $N = D + \epsilon_k$. Interval I_j has length of $(d_j + \lfloor \frac{\epsilon}{k} \cdot D \rfloor)/N$, which is slightly larger than d_j/N , the fraction of work needed. The additional length is to accommodate the approximation error of the (normalized) ranks of the agents. Each agent a will use the estimated normalized rank X_a to determine which task to work on. In particular, 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 6.** *Suppose that $X_a \in [\text{nank}(a) - \epsilon/(6k), \text{nank}(a) + \epsilon/(6k)]$ 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

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

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 $n \cdot (\frac{d_j}{N} + \frac{1}{N}) - 1 \geq d_j + 1 - 1 \geq d_j$, since $n \geq N$. Moreover, if $\text{nank}(a) \in I'_j$, then its estimate X_a must be in I_j , since $X_a \in [\text{nank}(a) - \epsilon/(6k), \text{nank}(a) + \epsilon/(6k)]$. 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 workers 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 $0, \lfloor \frac{\epsilon n}{6k} \rfloor, 2 \cdot \lfloor \frac{\epsilon n}{6k} \rfloor, \dots, \lceil \frac{6k}{\epsilon} \rceil \cdot \lfloor \frac{\epsilon n}{6k} \rfloor$. This can be done in $O((\log^2 n)/\epsilon)$ rounds by running $O(k)$ selection algorithms of [13] 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 7.** *After running the algorithm described above, we have $X_a \in [\text{nank}(a) - \epsilon/(6k), \text{nank}(a)]$.*

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

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 $O(\epsilon n) \cdot \text{OPT}$. 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 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  $X_t \leftarrow \begin{cases} 1, & \text{if } \text{ID}_{a'} < \text{ID}_a. \\ 0, & \text{otherwise.} \end{cases}$ 
    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 8.** *Suppose that agent a finishes its $T = \Theta((\frac{k}{\epsilon})^2 \log n)$ rounds after its last transient fault, w.h.p. $X_a \in [\text{nank}(a) - \frac{\epsilon}{6k}, \text{nank}(a) + \frac{\epsilon}{6k}]$.*

Proof. Notice that $\Pr(X_t = 1) = \text{rank}(a)/n = \text{nank}(a)$. Therefore, $\mathbb{E}[X_t] = \text{nank}(a)$. By Hoeffding's inequality, we have

$$\begin{aligned}
 \Pr\left(|X_{t-T+1} + X_{t-T+2} + \dots + X_t - T \cdot \mathbb{E}[X_t]| > T \cdot \frac{\epsilon}{6k}\right) &\leq \exp\left(-\frac{(\frac{\epsilon T}{6k})^2}{T}\right) \\
 \Pr\left(|X_a - \text{nank}(a)| > \frac{1}{6k}\right) &\leq \exp\left(-\frac{\epsilon^2 \cdot T}{36k^2}\right) \\
 &\leq \exp(-\Omega(\log n)) \\
 &\leq 1/\text{poly}(n)
 \end{aligned}$$

◀

Therefore, by Lemma 8 and Lemma 6, 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 $O(k \log n) \cdot \text{OPT}$. By Lemma 2, it suffices to show $\gamma(\mathcal{X}) = O(\log n)$.

► **Lemma 9.** *After $\Omega((\frac{k}{\epsilon})^2 \log n)$ rounds, w.h.p. $\gamma(\mathcal{X}) = O(\log n)$ (Recall that $\gamma(\mathcal{X}) = \sup_{0 \leq x \leq n-1} \sum_{X_a \in \mathcal{X}} |X_a \cap [\frac{x}{N}, \frac{x+1}{N})|$).*

Proof. For any $0 \leq x \leq N-1$, consider the interval $[\frac{x}{N}, \frac{x+1}{N}) \subseteq [0, 1)$. For $-D \leq i \leq D$, 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,

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

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

4.2.1 Fault Tolerance for Crash Faults

In this section, 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.

► **Lemma 10.** *Let $T = \Theta((\frac{k}{\epsilon})^2 \log n)$. Let f denote the number of agents who crashed during rounds $t-T+1, t-T+2, \dots, t$. Suppose that $f = O(\epsilon n/k)$, then w.h.p. $X_a \in [\text{nank}'(a) - \frac{\epsilon}{12k}, \text{nank}'(a) + \frac{\epsilon}{12k}]$, where $\text{nank}'(a)$ denote the normalized rank of a at round t .*

Proof. Recall that $X_a = \sum_{i=t-T+1}^t X_i/T$. First we will show that $\mathbb{E}[X_a] \in [\text{nank}'(a) - \frac{\epsilon}{12k}, \text{nank}'(a) + \frac{\epsilon}{12k}]$.

Let $\text{rank}(a), \text{nank}(a), n$ denote the rank, normalized rank, and the number of agents at round $t-T+1$. Let $\text{rank}(a)', \text{nank}(a)', n'$ denote those quantities at round t . First notice that the normalized rank of a during rounds $t-T+1, t-T+2, \dots, t$ is at least $\text{rank}'(a)/n$. Also note that $n' = n - f$. Therefore, for any $i \in [t-T+1, t]$,

$$\begin{aligned} \text{nank}'(a) - \mathbb{E}[X_i] &\leq \frac{\text{rank}'(a)}{n'} - \frac{\text{rank}'(a)}{n} \\ &= \text{rank}'(a) \cdot \left(\frac{1}{n-f} - \frac{1}{n} \right) \\ &= \text{rank}'(a) \cdot \left(\frac{f}{(n-f) \cdot n} \right) \\ &= \frac{f}{n} \\ &\leq \frac{\epsilon}{12k} \end{aligned} \quad \begin{aligned} \text{rank}'(a) &\leq n' = n - f \\ f &= O\left(\frac{\epsilon n}{k}\right) \end{aligned} \quad (1)$$

On the other hand, the normalized rank of a during rounds $t - T + 1, t - T + 2, \dots, t$ is at most $\max_{0 \leq j \leq f} \frac{\text{rank}'(a) + j}{n' + j} \leq \frac{\text{rank}'(a) + f}{n}$. Therefore, for any $i \in [t - T + 1, t]$,

$$\begin{aligned} \mathbb{E}[X_i] - \text{nank}'(a) &\leq \frac{\text{rank}'(a) + f}{n} - \frac{\text{rank}'(a)}{n'} \\ &\leq \frac{f \cdot (n - f - \text{rank}'(a))}{(n - f) \cdot n} \\ &\leq \frac{f}{n} \leq \frac{\epsilon}{12k} \quad f = O(\epsilon n/k) \end{aligned} \quad (2)$$

Therefore,

$$\mathbb{E}[X_a] \in \left[\text{nank}'(a) - \frac{\epsilon}{12k}, \text{nank}'(a) + \frac{\epsilon}{12k} \right]. \quad (3)$$

Since X_a is a sum of (not necessarily identical) independent random variables, we can drive the same bound obtained by Lemma 8 by using a Hoeffding's inequality:

$$\Pr \left(X_a \in \left[\mathbb{E}[X_a] - \frac{\epsilon}{12k}, \mathbb{E}[X_a] + \frac{\epsilon}{12k} \right] \right) \geq 1 - 1/\text{poly}(n). \quad (4)$$

Combining (3) and (4), we conclude that w.h.p. $X_a \in [\text{nank}'(a) - \frac{\epsilon}{6k}, \text{nank}'(a) + \frac{\epsilon}{6k}]$. ◀

► **Lemma 11.** *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.*

Proof. Consider the round t after the $(1 - \frac{\epsilon n}{12fk}) \cdot T$ additional rounds. It suffices to show that the expected X_a -value of a lies in $[\text{nank}(a) - \frac{\epsilon}{12k}, \text{nank}(a) + \frac{\epsilon}{12k}]$.

The current estimate of X_a is an average over the rounds $t - T + 1, t - T + 2, \dots, t$. We can divide the rounds into two parts, where the second part consists of the “additional” $(1 - \frac{\epsilon n}{12fk}) \cdot T$ rounds and the first part is the $\frac{\epsilon n}{12fk} \cdot T$ rounds before the additional rounds. If round i belongs to the second part, then we have $\mathbb{E}[X_i] = \text{rank}(a)/n$. Otherwise, from (1) and (2) in the proof of Lemma 10, we have:

$$\text{nank}(a) - \frac{f}{n} \leq \mathbb{E}[X_i] \leq \text{nank}(a) + \frac{f}{n}.$$

Therefore,

$$\begin{aligned} \mathbb{E}[X_a] &\leq \frac{\epsilon n}{12fk} \cdot \left(\text{nank}(a) + \frac{f}{n} \right) + \left(1 - \frac{\epsilon n}{12fk} \right) \cdot \text{nank}(a) \\ &\leq \text{nank}(a) + \frac{\epsilon}{12k}. \end{aligned}$$

Similarly, $\mathbb{E}[X_a] \geq \text{nank}(a) - \frac{\epsilon}{12k}$. ◀

5 Open Problems Motivated by This Work

In the following, we propose some open problems motivated from 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 D^2 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 ranking problem in the gossip model.** In our ranking algorithm, the preprocessing phase relies on the estimation of ranks for every agent. We have two algorithms for estimating the normalized rank up to $\pm\epsilon$ in $O(\frac{1}{\epsilon^2} \cdot \log n)$ and $O(\frac{1}{\epsilon} \cdot \log^2 n)$ rounds w.h.p. in the uniform gossip model. The question is whether one can do better, say in $O(\frac{1}{\epsilon} \cdot \log n)$ rounds using $O(\log n)$ bits of messages.
- **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 $2 \cdot \text{OPT}$ is achievable, while our lower bound shows that this is the best possible. In fact, we have partial results showing that for $D \leq 7$ (see Appendix A), we can achieve a switching cost of $2 \cdot \text{OPT}$. For $D > 7$, we could not generalize the pattern and therefore it is yet to be investigated.

References

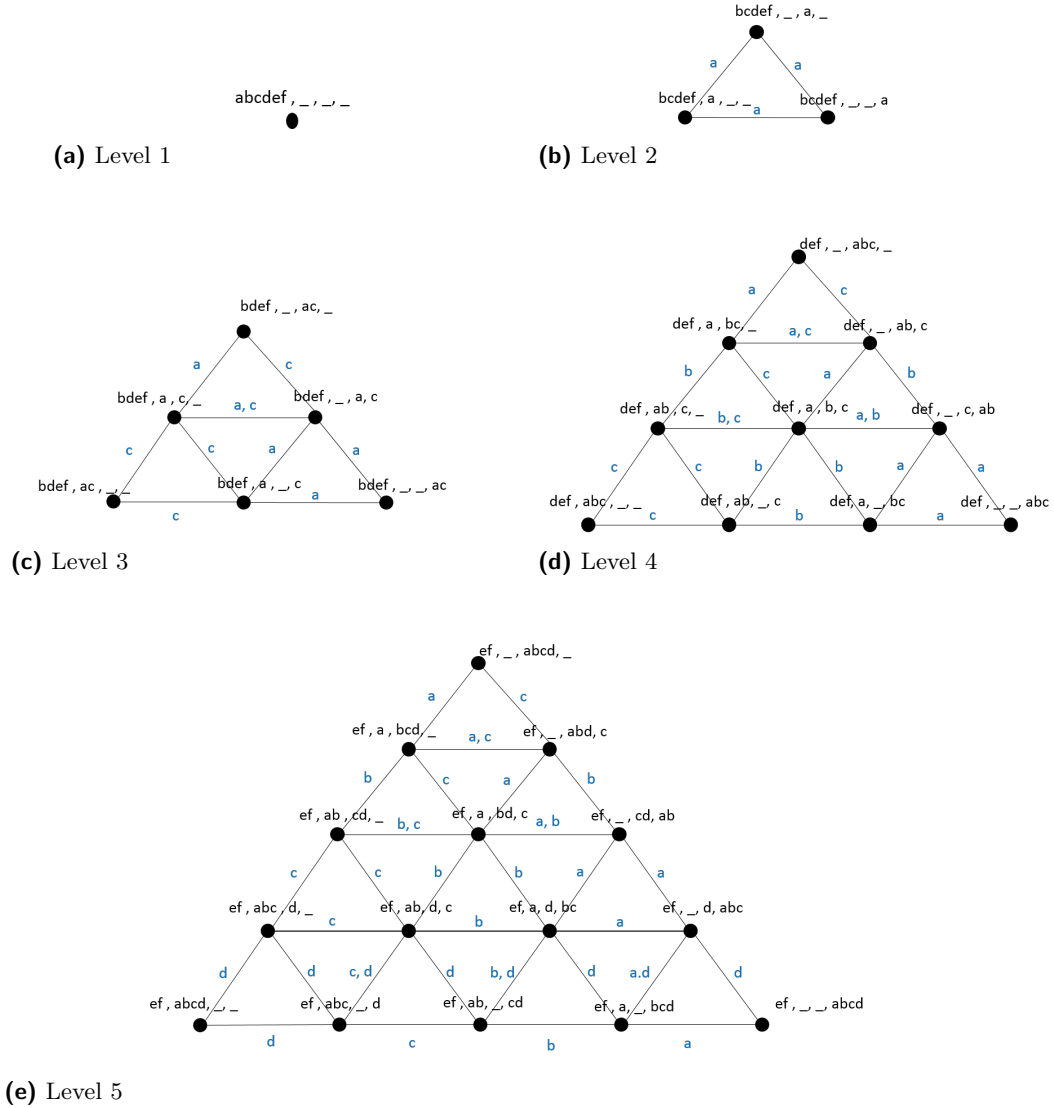
- 1 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
- 2 Samuel N. Beshers and Jennifer H. Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46(1):413–440, 2001.
- 3 E. Bonabeau, G. Theraulaz, and J-L. Deneubourg. Fixed response thresholds and the regulation of division of labor in insect societies. *Bulletins of Mathematical Biology*, 60:753–807, 1998.
- 4 Eric Bonabeau, Guy Theraulaz, and Jean-Louis Deneubourg. Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies. *Proceedings of the Royal Society of London B: Biological Sciences*, 263(1376):1565–1569, 1996.
- 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 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. The communication complexity of distributed task allocation. In *Proc. 31st ACM Symposium on Principles of Distributed Computing (PODC)*, pages 67–76, 2012.
- 9 Ofer Feinerman and Amos 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.
- 10 Alan M. Frieze and Geoffrey R. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.
- 11 Brian P. Gerkey and Maja 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.

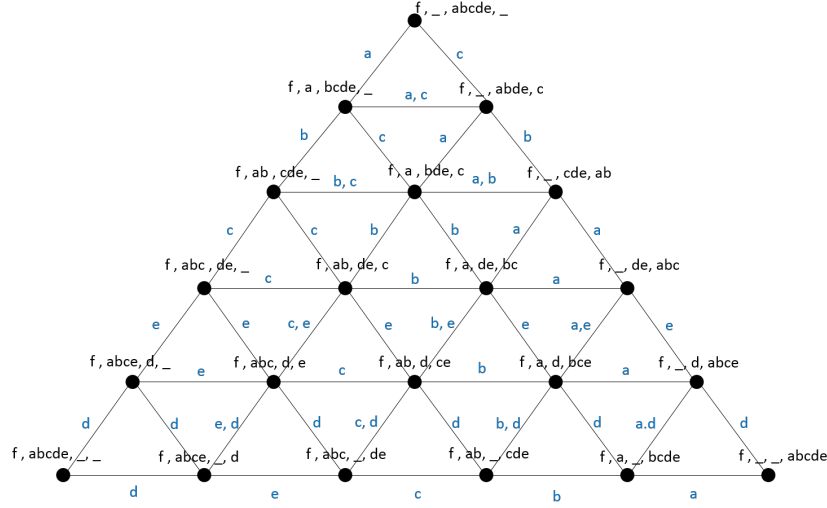
- 12 Mohsen Ghaffari, Cameron Musco, Tsvetomira Radeva, and Nancy A. Lynch. Distributed house-hunting in ant colonies. In *Proc. 34th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 57–66, 2015.
- 13 George Giakkoupis, Anne-Marie Kermarrec, and Philipp Woelfel. Gossip protocols for renaming and sorting. In *Proc. 27th Symposium on Distributed Computing (DISC)*, pages 194–208, 2013.
- 14 Deborah M. Gordon. The organization of work in social insect colonies. *Complexity*, 8(1):43–46, 2002.
- 15 Deborah M. Gordon, Brian C. Goodwin, and L.E.H. Trainor. A parallel distributed model of the behaviour of ant colonies. *Journal of Theoretical Biology*, 156(3):293–307, 1992.
- 16 Michael J. Greene and Deborah M. Gordon. Interaction rate informs harvester ant task decisions. *Behavioral Ecology*, 18(2):451–455, 2007.
- 17 William O.H. Hughes, Seirian Sumner, Steven Van Borm, and Jacobus 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.
- 18 David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *IEEE 44th Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2003.
- 19 Stephan Kozub. Amazon’s new drone delivery plan includes package parachutes. *The Verge*, 2017. URL: <https://www.theverge.com/2017/2/14/14611242/amazon-drone-package-delivery-parachute-patent-prime-air>.
- 20 Tobias Langner, David Stolz, Jara Uitto, and Roger Wattenhofer. Fault-Tolerant ANTS. In *28th International Symposium on Distributed Computing (DISC)*, pages 31–45, 2014.
- 21 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.
- 22 Lantao Liu and Dylan A Shell. Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33(3):291–307, 2012.
- 23 Cameron Musco, Hsin-Hao Su, and Nancy 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.
- 24 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.
- 25 Henrique M. Pereira and Deborah M. Gordon. A trade-off in task allocation between sensitivity to the environment and response time. *Journal of Theoretical Biology*, 208(2):165–184, 2001.
- 26 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.
- 27 Tsvetomira Radeva. *A Symbiotic Perspective on Distributed Algorithms and Social Insects*. Dissertation, Massachusetts Institute of Technology, 2017.
- 28 Gene E. Robinson. Regulation of division of labor in insect societies. *Annual Review of Entomology*, 37(1):637–665, 1992.
- 29 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.
- 30 Frederic Tripet and Peter 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.
- 31 Edward O. Wilson. Caste and division of labor in leaf-cutter ants (hymenoptera: Formicidae: Atta). *Behavioral Ecology and Sociobiology*, 7(2):157–165, 1980.

A 2-Optimal Switching Cost for Small Demands with 4 Tasks

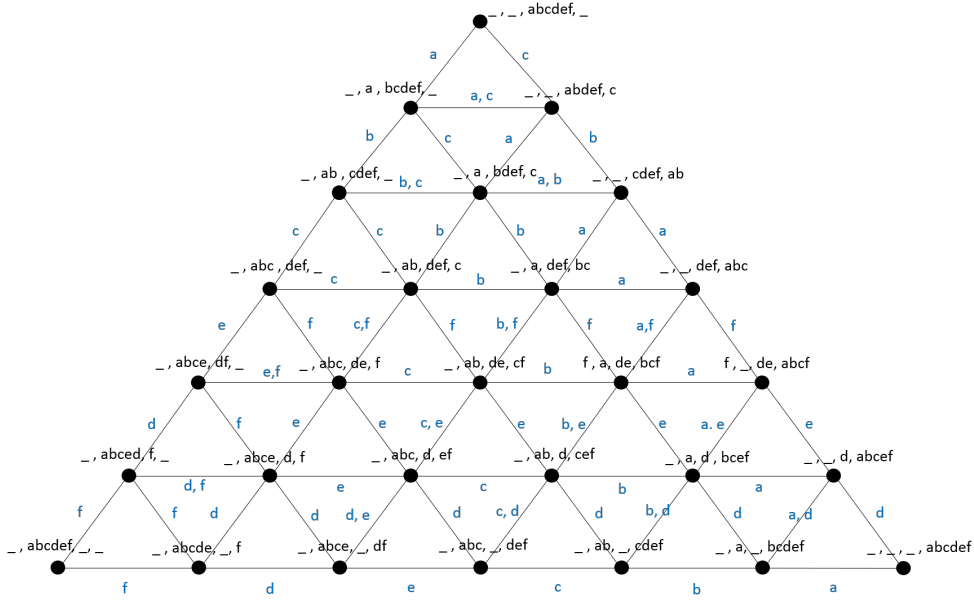
We give a solution for achieving a 2-optimal switching cost for $n = D = 7$ and $k = 4$ by figures. For $n = D < 7$, the solutions can be easily derived from that.

Figure 1 A solution for $k = 4$ and $D = 7$. Each node represents a demand vector. For example, the node in the first level represents $(7, 0, 0, 0)$. Therefore, the set of all demands (d_1, d_2, d_3, d_4) such that $d_1 + d_2 + d_3 + d_4 = 7$ form a 3D simplex. Two nodes are adjacent if the L_1 difference of the demands vectors they represents equal to 2. The labels on the edges denote the ants that switch tasks when the demands change from one endpoint to another (the edges across levels are omitted due to the difficulty of drawing). Since each adjacent move involves at most 2 agents, the solution is 2-optimal.





(f) Level 6



(g) Level 7

B Proof of Lemma 5

► **Lemma 5.** *If $\text{TK}_a \neq \text{TK}_{a'}$ for any $a \neq a'$, then $\text{TK}'_a \neq \text{TK}'_{a'}$ for any $a \neq a'$. Moreover, $\text{TK}'_a \in \{0, 1, \dots, \lceil (1 + \epsilon)D \rceil - 1\}$.*

Proof. First, we try to bound of the range of TK'_a . Note that originally we have $\text{TK}_a \in [\sum_{i=1}^{j(a)-1} d_i + \epsilon_{j(a)-1}, \sum_{i=1}^{j(a)} d_i + \epsilon_{j(a)})$. Consider the case where a did not switch tasks first. If $d'_{j(a)} < d_{j(a)}$, then it must be the case that $\text{TK}_a < (\sum_{i=1}^{j(a)-1} d_i) + d'_{j(a)} + \epsilon_{j(a)}$. Otherwise,

we must have $\text{TK}_a < (\sum_{i=1}^{j(a)-1} d_i) + d_{j(a)} + \epsilon_{j(a)}$. In any case,

$$\left(\sum_{i=1}^{j(a)-1} d_i \right) + \epsilon_{j(a)-1} \leq \text{TK}_a < \left(\sum_{i=1}^{j(a)-1} d_i \right) + \min(d'_{j(a)}, d_{j(a)}) + \epsilon_{j(a)} \quad (5)$$

Therefore, we can derive the following:

$$\begin{aligned} \text{TK}_a + \Delta_{j(a)} &\leq \text{TK}'_a < \text{TK}_a + \Delta_{j(a)} \\ \left(\sum_{i=1}^{j(a)-1} d_i \right) + \epsilon_{j(a)-1} + \sum_{i=1}^{j(a)-1} (d'_i - d_i) &\leq \text{TK}'_a < \left(\sum_{i=1}^{j(a)-1} d_i \right) + \\ &\quad \min(d'_{j(a)}, d_{j(a)}) + \epsilon_{j(a)} + \sum_{i=1}^{j(a)-1} (d'_i - d_i) \\ \left(\sum_{i=1}^{j(a)-1} d'_i \right) + \epsilon_{j(a)-1} &\leq \text{TK}'_a < \left(\sum_{i=1}^{j(a)-1} d'_i \right) + \min(d'_{j(a)}, d_{j(a)}) + \epsilon_{j(a)} \end{aligned} \quad (6)$$

On the other hand, consider the case where a switched task, then it must switch to a task $j'(a)$ such that $d'_{j'(a)} > d_{j'(a)}$. Therefore,

$$\begin{aligned} \text{TK}'_a &= \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d_{j'(a)} + \epsilon_{j'(a)} + \text{rank}_{A'}(a) - \left(\sum_{i=1}^{j'(a)-1} \phi_i \right) \\ &< \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d_{j'(a)} + \epsilon_{j'(a)} + \left(\sum_{i=1}^{j'(a)-1} \phi_i \right) + d'_{j'(a)} - d_{j'(a)} - \left(\sum_{i=1}^{j'(a)-1} \phi_i \right) \\ &= \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d'_{j'(a)} + \epsilon_{j'(a)} \end{aligned} \quad (7)$$

Also,

$$\text{TK}'_a = \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d_{j'(a)} + \epsilon_{j'(a)} + \text{rank}_{A'}(a) - \left(\sum_{i=1}^{j'(a)-1} \phi_i \right) \geq \left(\sum_{i=1}^{j'(a)-1} d'_i \right) + d_{j'(a)} + \epsilon_{j'(a)} \quad (8)$$

In both cases (whether a switched task or not), if a ends up working on task $j'(a)$, we have

$$\text{TK}'_a \in \left[\left(\sum_{i=1}^{j'(a)-1} d'_i \right) + \epsilon_{j'(a)-1}, \left(\sum_{i=1}^{j'(a)} d'_i \right) + \epsilon_{j'(a)} \right)$$

Therefore, for any two agents a and a' such that $j'(a) \neq j'(a')$, it is impossible that $\text{TK}'_a = \text{TK}'_{a'}$. Thus, it suffices to consider the case where $j'(a) = j'(a')$. In this case, if both a and a' switched tasks, then it is clearly that $\text{TK}'_a \neq \text{TK}'_{a'}$ by definition. Similar, if both a and a' did not move, then it is also true that $\text{TK}'_a \neq \text{TK}'_{a'}$. Finally, if a switched task but a' did not, then by (6), (7), and (8), the possible ranges of TK'_a and $\text{TK}'_{a'}$ are disjoint and thus they cannot be the same. \blacktriangleleft