2013 European Control Conference (ECC)
July 17-19, 2013, Zürich, Switzerland.

A decentralized algorithm for the preferred assignment problem in multi-agent systems

Usman A. Khan[†] and Soummya Kar[‡]
Department of Electrical and Computer Engineering
[†]Tufts University, 161 College Ave., Medford, MA 02144, USA
[‡]Carnegie Mellon University, 5000 Frobes Ave., Pittsburgh, PA 15213, USA
Email: khan@ece.tufts.edu, soummyak@ece.cmu.edu

***Abstract*— In this paper, we consider a task-allocation problem where N agents are to be assigned N distinct tasks without a central coordinator. The agents communicate over a connected network and we assume that each agent has a preference for a distinct unique task, i.e., without loss of generality, agent i prefers task i and is *unhappy* with any other task $(\neq i)$. With the assumption that each agent starts by picking an arbitrary random task, we can divide the *preferred* task-allocation problem into two phases: (i) to reach the set of $N!$ *unique assignments* from the set of N^N total configurations–presented elsewhere; and (ii) to reach the single *preferred assignment* from the set of $N!$ unique configurations.**

In this paper, we consider the phase (ii) of this *preferred assignment* problem, i.e., we assume that the network is already in one of the $N!$ unique configurations. In particular, we assume that each agent has a unique task that may not be preferred and each agent does not know its preference until it sees the task and evaluates its cost. Under mild conditions on the network connectivity, we propose a *swap-stick* algorithm and show that this algorithm reaches the preferred assignment in finite-time (a.s.) using Markov-chain arguments. The analysis in this paper is further extendable to show the convergence of the phase (i) assignment problem.

I. Introduction

In order to motivate the problem, we consider a formation control setup where each agent would like to occupy a preferred (distinct) spot in the formation. Each agent starts by randomly occupying a location where it computes the cost of staying at this location. We assume that (i) the costs are not known a priori; and (ii) for each agent i there is a distinct (optimal) location with minimum associated cost, i.e., no two agents have the same optimal location. Such setup is of interest when the agents perform basic operations at arbitrary locations and over time learn their preferred spot in the formation and, in a sense, is in fact an instance of an adaptive allocation problem. Similarly, in wireless communication, the agents start communicating using arbitrary but distinct channels and learn (over time) their preferred channels by hopping across different channels.

The preferred assignment problem is a simplified formulation of the general assignment problem where N agents are required to pick N distinct tasks such that the overall network cost (some function of the local costs) of the agent-to-task matching is minimized. When the cost function is linear, the assignment problem is termed as the linear assignment problem that can be solved using linear programming and can also be cast as a network flow problem [1]. The well-known Hungarian algorithm [2] and the auction algorithm [3] have been considered to solve the linear assignment problem in polynomial time. However, these solutions are centralized and rely on a central process to handle all of the computation and information in the system. A related algorithm is Fisher-Yates shuffle [4], [5] where the goal is to randomize an array containing N integers, $[1, \ldots, N]$. However, when such a randomization is to be achieved over a sparse graph with no central dispatcher, the existing centralized algorithms are no longer applicable.

With the recent interest in multi-agent coordination and formation, and target/facility/resource allocation problems, it is desirable to have distributed solutions to the assignment problem, see [6], [7], [8] and references therein. Of particular interest are [9] and [10] that describe a distributed auction algorithm for the assignment problems. The cost model used in these references allows for complex agent-to-task costs where an agent may prefer non-distinct and multiple tasks. Roughly speaking, this requires an agreement on the cost structure and preference among the agents, which is carried out via average consensus [11]. As average-consensus is typically asymptotic, finite-time assignment in [9] and [10] that is further conflict-free may not be guaranteed. In other related literature [12], [13], task allocation is considered as a reward maximization (as opposed to cost-minimization) problem. It is not unnatural to assume that such a reward is revealed when an agent encounters a task for the first time. However, the maximum reward that an agent may get is known at each agent a priori. A simple example is a social network where individuals pick an envelope containing a job description, required skill, and expected compensation; the individuals maximize their reward when they perform a task that matches their skill.

In this paper, we consider a simplified cost model where the cost, c_{ij}, of agent i being assigned task j is $c_{ii} = 0$ and $c_{ij} > 0$ when $i \neq j$. Clearly, the preferred assignment is matching agent i to task i resulting in zero network cost–any other assignment will result in a (strictly) positive cost. However, we assume that the labeling in the task-space is unknown a priori and, the label of a task (or its cost) is revealed to an agent only when the agent encounters it. The *preferred assignment problem* is to pick the *least-cost*

configuration out of all possible $N!$ unique configurations[1]. In this context, we provide a swap-stick algorithm that starts by each agent picking an arbitrary random task and consists of two steps: (i) *Swap step*–Two agents swap their tasks if one of them does not have its preferred task; and (ii) *Stick step*–Two agents stick to their tasks if they have their preferred tasks. Using the arguments from Markov chains, we show that this algorithm reaches the preferred assignment in finite-time.

We now describe the rest of the paper. Section II describes the problem formulation whereas the swap-collide is presented in Section III. We present the swap-stick algorithm along with its convergence properties in Section IV. We then consider generalization of the proposed agent-to-task cost structure in Section V. Finally, we present simulations in Section VI and Section VII concludes the paper.

II. Problem formulation

Consider a network of N agents in the set $\mathcal{V} = \{1, \ldots, N\}$ to be assigned N tasks in the set $\mathcal{T} = \{1, \ldots, N\}$, uniquely, such that the following cost function, $c = \sum_{i=1}^{N} c_{ij}$, is minimized, where c_{ij} is the local cost of performing task j at agent i. We assume the local cost function to be:

$$c_{ij} = \begin{cases} 0, & i = j, \\ > 0, & i \neq j. \end{cases} \tag{1}$$

Clearly, the minimum over all possible (i, j) matchings with $i \in \mathcal{V}$ and $j \in \mathcal{T}$ is achieved when agent i is matched with the ith task, i.e., with this matching $c = 0$. We term this as the *preferred assignment problem* and seek a decentralized solution where the local cost, c_{ij}, is only revealed when agent i encounters the jth task.

Let $x_i(t)$ be the state of the agent i at time t. When each agent randomly picks a task, i.e., $x_i(0)$ for each i is a random integer in the set $\mathcal{T} = \{1, \ldots, N\}$, it can be easily verified that the network state, $\mathbf{x}(0) \triangleq [x_1(0), \ldots, x_N(0)]^T$, lies in one of the possible N^N states. We further assume that the network is already in one of the possible $N!$ unique configuration, i.e.,

$$x_i(0) \neq x_j(0), \forall i, j, \qquad \cup_i x_i(0) = \mathcal{T}.$$

Then the preferred assignment problem is to find a protocol such that $\mathbf{x}(t)$ reaches (and stays in) the *preferred* state

$$\mathbf{x}_P \triangleq [1, \ldots, N]^T \tag{2}$$

for some finite t from any of the possible $N!$ initial conditions, $\mathbf{x}_0$.

A. Communication model

We assume a generalized gossip-based communication policy [15], in which at each iteration the symmetric communication graph consists of pairwise disjoint links[2], chosen (randomly) from the set of allowable links. Consider the N agents to communicate over an undirected graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is the edge set containing pairs of connected vertices. Furthermore, let $\mathcal{A}$ denote the adjacency matrix of the graph. We assume the following generic communication model, which subsumes the widely used gossiping protocol for real time embedded architectures [16], and the graph matching based communication protocols for internet architectures [17]. Define the set $\mathcal{M}$ of symmetric 0-1 $N \times N$ matrices:

$$\mathcal{M} = \left\{ A \;\middle|\; \mathbf{1}^T A = \mathbf{1}^T, \quad A\mathbf{1} = \mathbf{1}, \quad A \leq \mathcal{A} \right\}. \tag{3}$$

In other words[3], $\mathcal{M}$ is the set of adjacency matrices, such that, every node is incident to exactly one edge (including self edges) and allowable edges are only those included in $\mathcal{E}$. Let $\mathcal{D}$ be a probability distribution on the space $\mathcal{M}$. We make the following assumption of connectivity on the average:

Assumption (C.1): The sequence of time-varying adjacency matrices, $\{A(t)\}_{t \geq 0}$, governing the inter-agent communication, is an i.i.d. sequence in $\mathcal{M}$ with distribution $\mathcal{D}$. Also the mean adjacency matrix, $\overline{A}$, given by

$$\overline{A} = \mathbb{E}\left[A(t)\right] = \int_{\mathcal{M}} A \; \mathrm{d}\mathcal{D}(A), \tag{4}$$

is assumed to be irreducible and aperiodic.

The stochasticity of $\overline{A}$ is inherited from the elements of $\mathcal{M}$. We are not concerned with the properties of the distribution $\mathcal{D}$ as long as the *weak* connectivity assumption is satisfied. The matrix $\overline{A}$, being irreducible, depends on the allowable edges $\mathcal{E}$ and the distribution $\mathcal{D}$. In order to show the applicability of assumption **(C.1)** and justify the notion of weak connectivity, we note that such a distribution $\mathcal{D}$ always exists if the graph $(\mathcal{V}, \mathcal{E})$ is connected, for example, the $\mathcal{D}$ generated by the simple pairwise gossip protocol in [16].

III. Swap-collide algorithm

In [14], we provided a decentralized task-allocation algorithm that reaches one of $N!$ unique configurations (any permutation of $\mathcal{T}$) from any of the possible N^N initial conditions in finite-time (a.s.). The algorithm consists of a swap-step and a collide-step given as follows and can be operated in a memoryless or a memory-based scenario.

A. Memoryless

At each time $t + 1, t \geq 0$, if agent i is connected with agent j $(i \neq j)$, then agents i and j perform one of the following operations: (i) Swap: If $x_i(t) \neq x_j(t)$, the agents swap their tasks, i.e.,

$$x_i(t+1) = x_j(t), \tag{5}$$
$$x_j(t+1) = x_i(t). \tag{6}$$

(ii) Collide: If $x_i(t) = x_j(t)$, then one of the agents, say i, randomly picks a new task, i.e.,

$$x_i(t+1) = g(\mathcal{T} \setminus \{x_i(t)\}), \tag{7}$$
$$x_j(k+1) = x_j(t), \tag{8}$$

[1]We assume that the network is already in one of the $N!$ unique configurations or may use the swap-collide algorithm from [14] to reach this state.

[2]A graph is said to consist of pairwise disjoint links if at most one link is incident to each vertex.

[3]The inequality $A \leq \mathcal{E}$ is to be interpreted component-wise in (3).

where $g(\cdot)$ is a task generator function that randomly (with uniform distribution) chooses a task from the tasks in its arguments. Since the goal in swap-collide is to converge to any unique assignment, we denote by $\mathcal{C}$ the set of all possible $(N!)$ unique assignments. As a performance metric of our distributed assignment algorithms we define the (random) hitting time of the set $\mathcal{C}$ as follows:

$$T_{\mathcal{C}} = \inf\{k \geq 0 \mid \mathbf{x}(k) \in \mathcal{C}\}, \tag{9}$$

where, by convention, the infimum of an empty set is taken to be ∞. In other words, $T_{\mathcal{C}}$ denotes the earliest time at which the network reaches or hits the acceptable configuration set $\mathcal{C}$ (provided an acceptable configuration is ever reached). Convergence of the assignment algorithm or achievement of an acceptable task configuration corresponds to whether $T_{\mathcal{C}}$ is finite or not. The following result characterizes the reachability properties of the memoryless swap-and-collide algorithm.

Theorem 1: Let assumption **(C.1)** on the communication graph hold. Then,

(i) The process $\{\mathbf{x}(k)\}$ is a Markov chain on $\mathbb{Z}_N^N$. Moreover, the random time $T_{\mathcal{C}}$ is a stopping time with respect to (w.r.t.) the filtration generated by the Markov chain.

(ii) The Markov chain $\{\mathbf{x}(k)\}$ is transient with $\mathcal{C}$ as the set of absorbing states. Hence,

$$\mathbb{P}(T_{\mathcal{C}} < \infty) = 1. \tag{10}$$

The proof of the above theorem is not included in [14] but an avid reader may readily see that the analysis of swap-stick in Section IV is directly applicable to swap-collide. A characteristic of swap-collide is that the agents have no mechanism to realize that a network-wide unique assignment is reached and thus indefinitely operate in swap mode. In order to avoid this problem, one can extend the swap-collide to the case when each agent possesses memory as follows.

B. Memory-based

In addition to $x_i(k)$, let node i maintain a flag,

$$f_i(k) \in \{0, 1\},$$

initialized to 0. The modified algorithm leads to the following update of the node states and flags: At time $k+1$, if $(i, j) \in G$ and $i \neq j$, the nodes i and j perform one of the following steps depending on the status of their flags:

Case I: If $f_i(k) = f_j(k) = 0$, the nodes perform the basic swap-and-collide and update their flags as follows. Flag Update: node i sets its flag to 1 if all the N objects *pass* through it, i.e.,

$$f_i(k+1) = \begin{cases} 1, & \text{if } \cup_{l=0}^{k+1}\{x_i(l)\} = \mathcal{T}, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The flag update at node j is done similarly by looking at its past and current allocations.

Case II: If at least one of $f_i(k)$ and $f_j(k)$ is 1 (say $f_i(k) = 1$ in the following for definiteness), the nodes perform the following: (i) No State Update: The nodes do not swap their state, i.e.,

$$x_i(k+1) = x_i(k), \tag{12}$$
$$x_j(k+1) = x_j(k). \tag{13}$$

(ii) Flag Update: Both nodes set their flags to 1, i.e.,

$$f_i(k+1) = f_j(k+1) = 1.$$

We note that in the above update rule, the node memory is only used in the flag update steps. Also, it is readily seen that as soon as an node gets to set its flag to 1, it essentially halts the state update process. In turn, the algorithm terminates in finite time (a.s.) *iff* every network node gets to set its flag to 1 in finite time (a.s.)

In addition to the hitting time, $T_{\mathcal{C}}$, of the set $\mathcal{C}$, we define the following random times (for each network agent i):

$$T_{\mathcal{C}}^i = \inf\{k \geq 0 \mid f_i(k) = 1\}, \tag{14}$$
$$\overline{T}_{\mathcal{C}} = \max_{1 \leq i \leq N} T_{\mathcal{C}}^i, \tag{15}$$

where, by convention, the infimum of an empty set is ∞. As before, $T_{\mathcal{C}}$ denotes the earliest time when the network reaches the acceptable configuration set $\mathcal{C}$, of which the agents are generally unaware. From the above discussion, it follows that agent i stops its state update process at the random time $T_{\mathcal{C}}^i$ (provided it is finite), whereas, the entire network halts the update process at the random time $\overline{T}_{\mathcal{C}}$ (provided it is finite).

The following result summarizes the convergence of the modified swap-and-collide algorithm introduced above and determines a distributed algorithm termination criterion.

Theorem 2: Let assumption **(C.1)** on the communication graph hold. Then,

(i) The process $\{\mathbf{x}(k)\}$ is a Markov chain on $\mathbb{Z}_N^N$. Moreover, the random times $T_{\mathcal{C}}, T_{\mathcal{C}}^i$ and $\overline{T}_{\mathcal{C}}$ are stopping times w.r.t. the filtration generated by the Markov chain.

(ii) The Markov chain $\{\mathbf{x}(k)\}$ is transient with $\mathcal{C}$ as the set of absorbing states. Hence,

$$\mathbb{P}(T_{\mathcal{C}} < \infty) = 1. \tag{16}$$

(iii) The algorithm terminates network-wide in finite time a.s., i.e.,

$$\mathbb{P}(T_{\mathcal{C}} \leq T_{\mathcal{C}}^i \leq \overline{T}_{\mathcal{C}} < \infty) = 1. \tag{17}$$

The proof of Theorem 2 is more involved and is beyond the scope here. We note that the first and second assertions are similar to those of Theorem 1 and show that the network reaches $\mathcal{C}$ in a.s. finite time. In general, the network agents may not be aware of this hitting time $T_{\mathcal{C}}$. However, unlike the memoryless case, eventually the agents realize that the network has reached $\mathcal{C}$ and terminate the state update process. This follows from the third assertion of Theorem 2 which states that in a.s. finite time (larger than the actual hitting time $T_{\mathcal{C}}$), all the network agents set their flags to one thus terminating the state update process on a unique acceptable configuration. Finally, in passing, we note that the availability of agent memory has implications beyond the design of

distributed stopping criteria; it can be shown by pathwise comparison arguments that by replacing the task generation in (7) by

$$x_i(k+1) = g(\mathcal{T} \setminus \cup_{l=0}^{k}\{x_i(l)\}),$$

the convergence time (time to reach $\mathcal{C}$) and the halting time(s) can be approached earlier. However, in that case, the process $\{\mathbf{x}(k)\}$ no longer stays Markov.

IV. Swap-stick algorithm

We now describe the basic swap-stick algorithm. Let $\{x_i(t)\}$ denote the sequence of intermediate allocations (also referred to as states) at agent i as the iterative algorithm progresses. We assume that $\mathbf{x}(0) \in \mathcal{C}$ where $\mathcal{C}$ is the set of all $(N!)$ permutations of $\mathcal{T}$ and $\mathbf{x}$ is a vector of agent states. At each time $t+1, t \geq 0$, if $(i,j) \in A(t)$ and $i \neq j$, the agents i and j perform one of the following operations: (i) Swap: If $x_i(t) \neq i$ or $x_j(t) \neq j$, the agents swap their tasks, i.e.,

$$x_i(t+1) = x_j(t), \quad (18)$$
$$x_j(t+1) = x_i(t). \quad (19)$$

(ii) Stick: If $x_i(t) = i$ and $x_j(t) = j$, then the agents keep their tasks, i.e.,

$$x_i(t+1) = x_i(t), \quad (20)$$
$$x_j(t+1) = x_j(t). \quad (21)$$

Remarks:

(a) *Autonomous*: The above algorithm does not assume that the agents know the costs of any task a priori. Each agent evaluates the cost of the particular task it encounters. Without loss of generality, we have assumed that this setup results into agent i having 0 cost at task i. The algorithm can be modified such that each agent first computes the cost of its current task and then enters into either the swap or stick mode; motivating an autonomous operation that does not require any pre-processing.

(b) *Cooperative*: The swap-step is crucial for the information propagation within the network. It may seem counter-intuitive that an agent loses its preferred task when its neighbor does not also have its preferred task at the same time. However, the effectiveness of the swap-step can be shown by the following example. Consider Fig. 1 where the agent index is provided within the circle and its current task is listed at the top. Clearly, if agent 5 were not to swap its preferred task, the left and right components of the network will never encounter their preferred tasks. With the swap between agent 5 and any one of its neighbors, the tasks begin to flow in the network. In this sense, the swap-stick protocol is cooperative, where an agent temporarily goes into a non-optimal state for the benefit of the entire network.

(c) *Decentralized*: It is readily seen that the swap-stick algorithm does not require any central coordination. In addition, none of the agents are required to know the graph topology or the preferred tasks. The operation at each agent is decentralized. We have the following results.

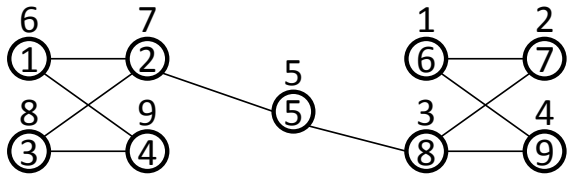

Fig. 1. An example to show the effectiveness of the swap-step.

Lemma 1 (Task preservation): If the network state starts in a unique configuration, i.e., $\mathbf{x}(0) \in \mathcal{C}$, then it remains in $\mathcal{C}$, i.e., $\mathbf{x}(t) \in \mathcal{C}$, for all $t > 0$.

Proof: At any $t \geq 0$, an active link, say $(i,j) \in A(t)$, *either* swaps $x_i(t-1)$ and $x_j(t-1)$ *or* keeps the tasks at $t-1$. Hence, no task is dropped from the network. Since the number of agents and the number of tasks are the same, we have $\mathbf{x}(t) \in \mathcal{C}$ for all $t \geq 0$. ■

Lemma 2 (Fixed-point): The preferred state, $\mathbf{x}_P$, is the fixed-point of the algorithm, i.e., once the algorithm reaches the preferred state, it stays in the preferred state.

Proof: Define t_P as the time when the state becomes

$$\mathbf{x}(t_P) = \mathbf{x}_P, \quad (22)$$

i.e., the ith agent has its preferred task i. Then the swap-stick always operates in the stick-step and the lemma follows. ■

A. Markovianity

The following lemmas establish the Markovianity of the process $\{\mathbf{x}(t)\}$.

Lemma 3: [Markov chain] Let assumption (**C.1**) on the communication graph hold and let $\mathbf{x}(0) \in \mathcal{C}$. Then the process $\{\mathbf{x}(t)\}$, generated by the swap-stick algorithm, is a Markov chain on $\mathcal{C}$. Moreover, the random hitting time t_P is a stopping time with respect to the filtration generated by the Markov chain.

Proof: By Lemma 1 we note that once initialized in the set of the acceptable configurations, the process $\{\mathbf{x}(t)\}$ never leaves the set $\mathcal{C}$. The Markovianity of $\{\mathbf{x}(t)\}$ then readily follows from the fact that the random adjacency matrices $A(t)$ are temporally i.i.d. and the memorylessness of the swap-stick algorithm, i.e., the local assignment dynamics at a given instant t is only a function of the current network configuration. That t_P is a stopping time, follows readily from standard arguments [18] and the lemma follows. ■

Lemma 4: [Absorbing state] The preferred state, $\mathbf{x}_P$, is the unique absorbing state of the Markov chain $\{\mathbf{x}(t)\}$.

The proof of Lemma 4 is omitted due to space limitations. The main idea behind the proof is that for each configuration $\mathbf{x} \in \mathcal{C}$, such that, $\mathbf{x} \neq \mathbf{x}_P$, there exists another configuration $\mathbf{y} \in \mathcal{C}$ with $\mathbf{y} \neq \mathbf{x}$, such that,

$$p_{\mathbf{x},\mathbf{y}} > 0. \quad (23)$$

The above combined with Lemma 2 leads to the fact that any configuration, $\mathbf{x} \neq \mathbf{x}_P$, is not absorbing and hence, $\mathbf{x}_P$ is the unique absorbing state.

Lemma 5: [Transient states] Each state, $\mathbf{x} \neq \mathbf{x}_P$, is a transient state of the Markov chain, $\{\mathbf{x}(t)\}$, with unique absorbing state $\mathbf{x}_P$.

The proof of Lemma 5 is omitted due to space limitations. The main idea behind the proof is that given the Markov chain is in some configuration $\mathbf{x} \neq \mathbf{x}_P$ at an instant t, there exists a time $t < t' < \infty$, when the chain reaches the unique absorbing state, $\mathbf{x}_P$, i.e.,

$$\mathbb{P}\left(\mathbf{x}(t') = \mathbf{x}_P \mid \mathbf{x}(t) = \mathbf{x}\right) > 0. \tag{24}$$

The above readily establishes the fact that any non-absorbing state ($\neq \mathbf{x}_P$) reaches the absorbing state in finite-time and is thus, a transient state of the Markov chain.

B. Finite-time (a.s.) convergence

We now characterize the reachability properties of the swap-stick algorithm. As a performance metric, define the (random) hitting time of the preferred state, $\mathbf{x}_P$, as

$$t_P = \inf\{t \geq 0 | \mathbf{x}(t) = \mathbf{x}_P\}, \tag{25}$$

where, by convention, the infimum of an empty set is taken to be infinity. In other words, t_P denotes the earliest time at which the network reaches hits the preferred state, $\mathbf{x}_P$ (provided a preferred state is ever reached). Convergence of the preferred assignment problem or hitting the preferred task corresponds to whether t_P is finite or not.

Theorem 3: Let assumption (**C.1**) on the communication graph hold. Then

$$\mathbb{P}(t_P < \infty) = 1.$$

Proof: The proof follows from Lemmas 3, 4, and 5. ■

We note that Theorem 3 does not only establish the convergence of the proposed approach, i.e., starting from an arbitrary initial configuration in $\mathcal{C}$, the network reaches the preferred configuration in finite time a.s., it naturally provides a distributed termination criterion. Indeed, as soon as the network reaches $\mathbf{x}_P$, the stick operation prevents further assignment swapping among the agents.

V. Arbitrary cost assignment

The preferred assignment problem we described in this paper can be motivated to have the following assumption on the agent-to-task cost structure. The local cost minimizers (at each gent) are *unique* and *distinct*. We now briefly study the swap-stick algorithm when this assumption does not hold.

As the first case, we consider when the local minimizer is unique at each agent but not distinct. Without loss of generality, consider agent 9 in Fig. 2 to have cost, $c_9 = 0$, for task 3 where all the other costs, $c_i, i = 1, \ldots, 8$ follow the description in (1) and suppose that the network state is according to Fig. 2. Since agent 9 does not have its preferred task, whenever a link is active with agent 9 being one of the nodes, agent 9 will always force a swap. This swap will result in another agent to have an non-preferred task and the network (which can be easily seen to be in the optimal configuration according to this cost) will move to a non-optimal state. Although this setup violates our assumption as the local minimizers are not distinct, the following modification may be helpful.

If two agents have non-preferred tasks they swap. If one of the agent has its preferred task, it engages in the *swap* operation with a probability α^t and does not engage with a probability $1 - \alpha^t$ for some $\alpha \in (0, 1]$. As $t \to \infty, \alpha^t \to 0$ and in our case agent 9 is unable to swap after a while as its neighbors will not engage. Loosely speaking, as the network keeps returning to the state in Fig. 2, it tends to stick in this state as t goes large. The precise analysis of this modification is beyond the scope of this paper.

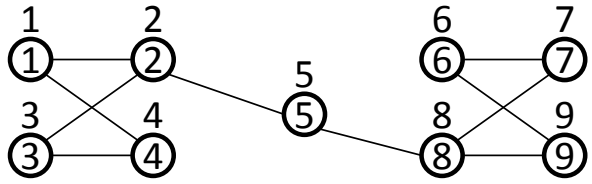

Fig. 2. Non-distinct task preference.

The second case is when the local minimizer is not unique but the union of the local minimizers result in $\mathcal{T}$. In other words, each task is preferred at least at one agent. Consider both agents 1 and 9 (in Fig. 2) to have the minimum cost 0 for both tasks 1 and 9. Since the algorithm starts in a unique configuration, it can be shown that the fixed-point will be either $[1, \ldots, 9]^T$ or $[9, 2, \ldots, 8, 1]^T$. It is straightforward to show that once the network state reached any one of these points, it will stay there because of the stick-step. The key assumptions here are that each task is preferred at least at one agent and the network starts in a unique configuration. For the sake of simplicity, we assume that local cost minimizers (at each gent) are both *unique* and *distinct*, but the uniqueness can be easily removed and Theorem 3 can be adjusted accordingly.

VI. Simulations

We consider $N = 9$ nodes connected randomly according to: (i) a circulant graph; and (ii) a nearest neighbor rule. We assume that the ith agent prefers the ith task. The graph

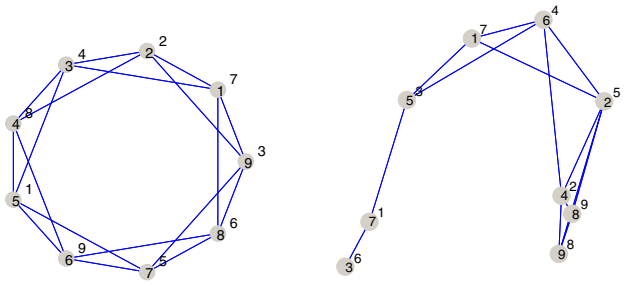

Fig. 3. (Left) Circulant graph with a (unique) random task assignment. (Right) Nearest-neighbor graph with a (unique) random task assignment.

topologies are shown in Fig. 3 (Left) and Fig. 3 (Right),

where the initial (random but in $\mathcal{C}$) task allocation is also marked at the top of the agents. This initial assignment is a result of the swap-collide algorithm (see [14]) that results in a unique agent-to-task pairing. Clearly, this task assignment is arbitrary, random, and may not be preferred at all agents. The swap-stick algorithm (over a meaningful set of iterations) for each of these configurations is show in Fig. 4 (circulant graph) and Fig. 5 (nearest-neighbor graph). Each curve in, for example Fig. 5, represents the state of an agent and shows the trajectory of this state as the function of the algorithm iterations.

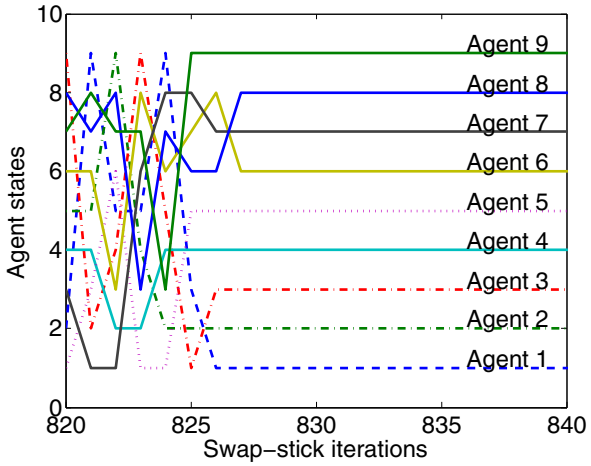

Fig. 4. The swap-stick algorithm for the agent/tasks in Fig. 3 (Left).

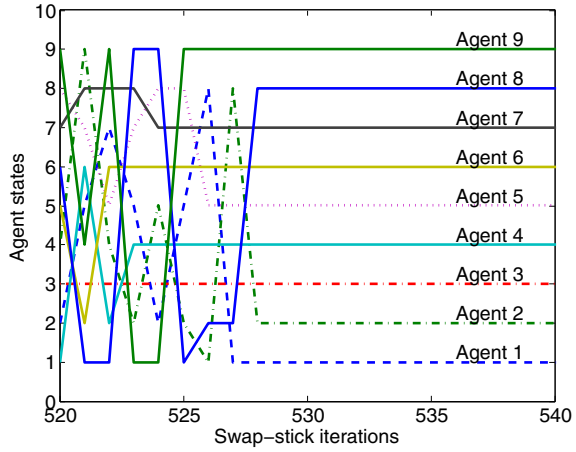

Fig. 5. The swap-stick algorithm for the agent/tasks in Fig. 3 (Right).

VII. Conclusions

In this paper, we present the *preferred assignment problem* where N agents have to be assigned N tasks such that each agent has a unique and distinct task preference. We assumed that the agents already have a unique assignment (or the network can be configured in this unique assignment with the swap-collide algorithm we presented elsewhere.) Under some mild assumptions on network connectivity, we provide a swap-stick algorithm and show that it converges to the preferred assignment among the agents using Markov chain arguments. We note that the agent-to-task cost structure of this paper is not the most general; nevertheless, the main goal in this paper is to motivate decentralized algorithms with finite-time performance guarantees (as opposed to consensus based asymptotic strategies.) Using the swap-stick (and swap-collide) intuition one may be able to generalize the cost structure to complex scenarios where, in addition, the number of agents and tasks may also be different.

Acknowledgments

We would like to thank the reviewers for their insightful review and thoughtful comments towards improving this paper.

References

[1] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[2] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955.

[3] D. P. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, vol. 21, pp. 152–171, 1981.

[4] R. A. Fisher and F. Yates, *Statistical Tables for Biological, Agricultural and Medical Research*, Oliver and Boyd, 1963.

[5] R. Durstenfeld, "Algorithm 235: Random permutation," *Communications of the ACM*, vol. 7, no. 7, Jul. 1964.

[6] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, Dec. 1998.

[7] S. Kloder and S. Hutchinson, "Path planning for permutation-invariant multirobot formations," *IEEE Transactions on Robotics*, vol. 22, no. 4, pp. 650–665, Aug. 2006.

[8] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 863–868, May 2007.

[9] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas, "A distributed auction algorithm for the assignment problem," in *47th IEEE Conference on Decision and Control*, Cancun, Mexico, Dec. 2008, pp. 1212 –1217.

[10] H. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[11] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems and Controls Letters*, vol. 53, no. 1, pp. 65–78, Apr. 2004.

[12] T. Balch, "Reward and diversity in multirobot foraging," in *In IJCAI Workshop on Agents Learning About, From and With other Agents*, Jul. 1999.

[13] B. P. Gerkey and M. J. Mataric, "Sold!: auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, Oct. 2002.

[14] U. A. Khan and S. Kar, "A coordination-free distributed algorithm for simple assignment problems using randomized actions," in *45th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2011, pp. 58–61.

[15] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.

[16] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, pp. 2508–530, Jun. 2006.

[17] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, Aug. 1999.

[18] J. Jacod and A. N. Shiryaev, *Limit theorems for stochastic processes*, Springer, New York, NY, 2002.