A distributed self-clustering algorithm for autonomous multi-agent systems

Victor L. Minden, Clifford C. Youn, and Usman A. Khan

***Abstract*— In this paper, we consider clustering in autonomous multi-agent systems where the agents, assumed to belong to either 'blue' or 'red' type, engage in local location-swaps to physically separate the network into two connected clusters. The groupings (red/blue) may reflect hardware differences or some other distinguishing factor that differs across groups. We present a randomized algorithm, Naïve Swap, where the agents randomly swap their locations with an arbitrary neighbor.We then provide two modifications to this naïve strategy, Intelligent Swap and Stable Marriage Swap, that are based on truncated average-consensus and the Gale-Shapley algorithm for solving the stable marriage problem, respectively. These modifications serve as an improvement upon the randomized (naïve) location-swaps by using information on the group association of neighbors to guide swapping decisions. We provide a sketch for the analysis of the proposed schemes via an irreducible Markov chain analogy. We further show the effectiveness of the proposed strategies via simulations.**

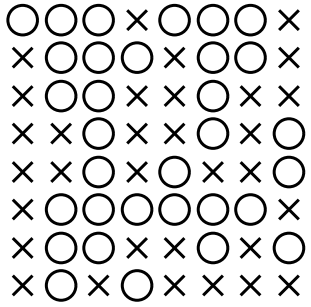

Fig. 1. An initial configuration on a grid with periodic boundary conditions.

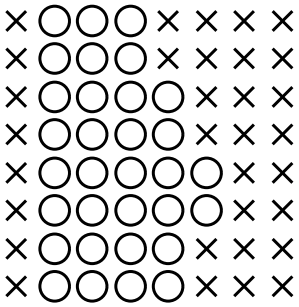

Fig. 2. A self-clustered network. We note that, because of the boundary conditions, the clusters are connected.

I. Introduction

Clustering is an important problem in randomly deployed multi-agent systems where the agents can be categorized into two or more distinct groups. To motivate the problem, we consider the case of a connected network of mobile agents that is in some way heterogeneous. This heterogeneity leads to clustering and could correspond to different types of agents clustered together with distinct physical differences, such as processing/battery power, sensing modalities, or some other, more abstract distinguishing features. The applications of such heterogeneous networks are many: agents with different hardware capabilities, for example, can be used intelligently to lessen overall power consumption [1] or expedite data collection [2].

The problem of interest is what, in this paper, we call *self-clustering*: given a network of autonomous and heterogenous agents, group all agents of the same type together via a distributed algorithm (see Fig. 1 and 2)[1]. This could be applied to situations where we have integrated multiple networks of agents to accomplish a task and wish to retrieve one (sub-)network for redeployment to another task, for example. Simply recalling the agents from their current locations in the network may leave the network disconnected, which makes coordination of the remaining agents impossible. Clustering the agents prior to removal is one method of ensuring that information can continue to flow throughout the graph even after one group of agents is removed.

V. L. Minden, C. C. Youn, and U. A. Khan are with the Department of Electrical and Computer Engineering, Tufts University, 161 College Ave, Medford, MA 02155, USA. victor.minden@tufts.edu, clifford.youn@tufts.edu, khan@ece.tufts.edu

[1]For visualization purposes, the figures in this paper will use $\times$ and $\circ$ for red and blue respectively.

We note that, on the surface, this problem resembles classical graph theory problems such as min-cut or partitioning [3] in that we desire to find groupings of the network. The key distinction is the mobility of the agents – the method of obtaining said groupings is very different. Given the desire for agents to move together, the self-clustering problem also resembles flocking [4], where we consider a network of mobile agents coordinating their positions and velocities to move as a unit, but differs in that we are looking to have agents flock with others that share their type and *not* with the entire network.

The clustering problem can be formulated on a continuous physical space, discretized physical grid (i.e., graphs following Euclidean distance restrictions), or arbitrary graphs. In this paper, we consider the clustering problem on a regular, discrete grid. Generalization to arbitrary dimensions and abstract connectivity, however, is also possible. We asume that, at time k, each agent occupies exactly one node of the graph. At each time step, each agent can swap positions with any one of its neighbors on nodes above/below, to the left/right, or at any of the four diagonals.

The rest of the paper is organized as follows. Section II develops the problem of self-clustering in a mathematical context. Section III presents a simple clustering algorithm, Naïve Swap, and discuss its convergence properties. Section IV employs truncated average-consensus to introduce an improvement on the Naïve Swap, that we term as 'Intelligent Swap', and further includes a discussion on the convergence. Section V outlines Stable Marriage Swap, a proposed strategy to improve upon the Intelligent Swap. Finally, Section VI provides numerical simulation results, and Section VII concludes the paper.

II. Problem Statement

Consider a network of N agents in the set $\mathcal{A} = \{1, ..., N\}$ and a connected, undirected graph, $G = (\mathcal{V}, \mathcal{E})$, with N nodes. Here, $\mathcal{V} = \{1, ..., N\}$ is the vertex set, an $\mathcal{E}$ is the edge set giving the pairs of connected vertices of G. We assume each agent to occupy exactly one node of the graph at any given time, with the assignment of agents to nodes at discrete time $k \in \mathbb{N}$ denoted by M_k, where M_k is an $N \times N$ permutation matrix that takes agents to node locations.

The graph, G, has a static, $N \times N$ symmetric adjacency matrix A, where

$$A_{nl} = \begin{cases} 1, & \text{if } (n,l) \in \mathcal{E}, \\ 0, & \text{if } (n,l) \notin \mathcal{E}. \end{cases}$$

We assume A to have unity diagonal.

Consider the case where the agent set, $\mathcal{A}$, has two distinct subsets, each of size $N/2$[2], i.e.,

$$\mathcal{A} \equiv \mathcal{R} \cup \mathcal{B}, \qquad |\mathcal{R}| = |\mathcal{B}| = \frac{N}{2}.$$

We will refer to agents in $\mathcal{R}$ as "red", and those in $\mathcal{B}$ as "blue", though these labels are arbitrary. We assume an ordering on the agents such that

$$\mathcal{A} = \{r_1, ..., r_{N/2}, b_1, ..., b_{N/2}\},$$

where $r_i \in \mathcal{R}$ and $b_i \in \mathcal{B}$. With this, we write the agent vector as $\mathbf{a} = [r_1, ..., r_{N/2}, b_1, ..., b_{N/2}]^T$ and the state vector as $\mathbf{x}_k = M_k \mathbf{a}$.

While the neighborhood of any given node is static, agents can move throughout the graph as time progresses, changing which node they occupy. Thus, the neighborhood of any given agent is not static. The communication matrix between agents at time k, C_k, is simply

$$C_k \equiv M_k^{-1} A M_k = M_k^T A M_k,$$

where here we have used the fact that permutation matrices are orthogonal. This communication matrix maps agents to graph nodes, finds adjacencies, and then maps nodes back to agents via a similarity transformation. As such, we will refer to the agents with which agent i can communicate at time k as the neighbor set of i,

$$\mathcal{N}_{k,i} = \{j \mid \text{The } (i,j) \text{ entry of } C_k \neq 0\}.$$

At each time k, we assume that the subsequent node assignment matrix M_{k+1} is formed by swaps between adjacent agents according to C_k, where a swap of agents i and j at time k means

$$\begin{aligned} \mathbf{x}_i(k+1) &= \mathbf{x}_j(k), \\ \mathbf{x}_j(k+1) &= \mathbf{x}_i(k), \end{aligned}$$

and M_{k+1} reflects these modifications. We restrict each agent to participate in at most one swap per time step. Our goal is to find a swap decision algorithm which uses local information at the agent level to cluster the network in finite-time, where we will say loosely that the network is clustered when the red agents are assigned to a well-connected subgraph of G, and so are the blue agents.

Formally, we will measure this indirectly by means of a cost function. We define an agent to be "happy" at time k if it shares a color with at least half of its neighbors at that time. Our cost function, then, is simply a binary measure,

$$J(M_k) = \begin{cases} 0, & \text{if all agents are happy,} \\ 1, & \text{otherwise.} \end{cases}$$

We aim to minimize $J(M_k)$, that is to say, achieve universal happiness.

A. Assumptions

In this paper, we assume the graph G to be a regular lattice graph where each node is connected to the surrounding 8 neighbors. To simplify the boundary conditions, the top nodes are connected to the bottom and the left nodes are connected to the right[3]. Additionally, we assume that the agents have some synchronization scheme. This is important as our later algorithms consist of separate phases, between which agents must transition as a whole.

III. Naïve Swap

We first consider a naïve approach in which each agent does not use any information from its neighbors in determining which neighbor with which to swap location. Specifically, we suppose that the agents exchange positions according to an asynchronous gossip-type protocol [6], where at each time step, k, a random unhappy node swaps with one of its neighbors of the other color. This "Naïve Swap" algorithm can be seen in Algorithm 1.

Algorithm 1 Naïve Swap

$k \leftarrow 0$
while $\exists$ unhappy agent i **do**
  Swap i with a random neighbor of the opposite color, with equal probability over all such neighbors
  $k = k + 1$
end while

We note that, as stated, the algorithm has a clear terminating condition – when all agents are happy, the loop exits. It is not immediately clear, however, that this condition is ever satisfied for arbitrary initial agent-node pairing M_0. We now show that the Naïve Swap algorithm, in fact, terminates in finite-time, with all agents happy. To do this, we require a few lemmas:

Lemma 3.1: The Naïve Swap is a Markov chain.

Proof: We give an informal sketch. Consider a single agent, i, at time k. If i is happy, which is a function of its current neighbor set, $N_{k,i}$, then it will not initiate a swap. If i is unhappy, it will swap with one of its opposite-color neighbors (chosen over all such neighbors, with equal probability). Thus, the location of agent i at time $k+1$ is

[2]We assume for convenience that N is even.

[3]It can be shown that this connectivity leads to an agent deployment on the surface of a torus, see [5] for details.

a random process which depends only on the location of all agents at time k and the lemma follows. ■

By assumption, there is an assignment of agents to nodes which makes all agents happy, and thus there exists at least one fixed point of the iteration. We note that there are no fixed points that correspond to any agent being unhappy, by definition of the algorithm. Defining the set of all fixed points of the iteration to be $\mathcal{F}$, we provide the following lemma.

Lemma 3.2: The Markov chain of the Naïve Swap is irreducible with $\mathcal{F}$ being the set of absorbing states.

Proof: A formal proof is outside the scope of this paper but we provide the following sketch. We need to show that the probability of reaching any fixed-point in $\mathcal{F}$ is non-zero. Because all transitions between an unhappy agent and an adjacent agent of the opposite color are equally probable, we just need to show that there is not a state of the network which causes "looping", where, with probability 1, unhappy agents are confined to some subgraph with no existing happy configuration. We argue that this is prevented by the graph topology, as having such a subgraph would require the rest of the network to also have unhappy nodes, and thus the configuration is not sustainable. ■

IV. Intelligent Swap

We now consider a modification to the Naïve Swap algorithm that uses communication between adjacent agents to make more intelligent swap decisions than the previous strictly random decisions. In particular, we will employ the average-consensus algorithm (see, for example, [7]).

In the classical average-consensus iteration, each node is assumed to have some initial value, $\mathbf{Y}_i(0)$, and we aim to calculate the average across all i in a distributed fashion. The Intelligent Swap algorithm uses a *truncated* average-consensus, i.e., the average-consensus is terminated in finite-time.

Algorithm 2 shows the pseudo-code for the Intelligent Swap. The matrix $\mathbf{Y}_k$ is an $N \times 2$ matrix where the $(i,1)$ entry, $\mathbf{Y}_{k,i,1}(t)$ is the number of red neighbors of agent i at time k at average-consensus iteration t, and the $(i,2)$ entry is defined similarly for blue neighbors. We note that the algorithm alternates between two distinct stages, the average-consensus stage and the swapping stage. The average-consensus stage at each outer iteration first propagates information about the distribution of red and blue agents on the graph to all agents. The swapping stage acts as an effective gossip-type protocol as before, where now the agents have some preference list they attempt to follow.

Heuristically, we claim that truncating the iteration in the average-consensus stage gives each agent a value that *approximates* the mean, but is biased higher or lower depending on the number of neighbors of each color that are nearby. Thus, these values give some extra information when performing a swap, and give the potential to make more intelligent decisions.

A. Convergence of Intelligent Swap

We note that the Intelligent Swap algorithm as it stands falls victim to looping, as deterministic structures can be developed which cause the early-truncation heuristic to give poor information. However, we introduce a small modification to the algorithm that we claim reduces the proof of convergence to the proof of convergence of the Naïve Swap algorithm.

Algorithm 2 Intelligent Swap

$k \leftarrow 0$
while $\exists$ unhappy agent **do**
  Form $\mathbf{Y}_k(0)$ by counting neighbors, i.e.,
  $\mathbf{Y}_{k,i,1}(0) \leftarrow |\{j \mid j \in \mathcal{N}_{k,i} \cap \mathcal{R}\}|$
  $\mathbf{Y}_{k,i,2}(0) \leftarrow |\{j \mid j \in \mathcal{N}_{k,i} \cap \mathcal{B}\}|$
  Run $O(\text{diam}(G))$ iterations of average-consensus on $\mathbf{Y}_k(0)$ with communication matrix C_k.
  $\mathcal{T} \leftarrow \{i \mid i \text{ is unhappy}\}$
  for each i in $\mathcal{T}$ that has not already moved **do**
    if $i \in \mathcal{R}$ **then**
      Rank each neighbor $j \in \mathcal{B}$ by $\mathbf{Y}_{k,j,1}(n)$
    else
      Rank each neighbor $j \in \mathcal{R}$ by $\mathbf{Y}_{k,j,2}(n)$
    end if
    Swap with highest-ranked neighbor of the other color who has not already swapped
  end for
  $k = k + 1$
end while

In particular, we introduce a randomized component to the algorithm by choosing some probability parameter $p \in (0,1]$. In Algorithm 2, consider the swap phase that subsequently follows each average-consensus iteration. We replace the deterministic "swap with highest-ranked available neighbor" with a randomized step: each unhappy agent swaps with their highest-ranked neighbor of the other color with probability p, else they swap randomly.

This modification causes all probabilistic arguments applied to Naïve Swap thus far to apply to the Intelligent Swap – we have effectively hybridized the algorithm.

V. Alternative Strategy – Stable Marriage Swap

We consider one final alternative strategy that goes beyond (truncated) average-consensus. This is motivated by the observation that many unhappy agents that are near each other at some time step k may have similar preference lists, and, thus, there will be competition between them to swap with the same agents.

We are interested in coordinating the swaps such that each unhappy agent chooses a neighbor with which to swap in a way that leads to the overall swap, maximizing the utility of the agents as a whole. This is related to the stable marriage problem, an assignment problem that seeks to find stable matchings in a bipartite graph [8]. In [9], Floréen et al. give results on a distributed version of the well-known Gale-Shapley algorithm for this problem, which we explore in our simulations. In particular, rather than having agents swap asynchronously after the average-consensus step in Algorithm 2, we run a fixed number of iterations of

distributed Gale-Shapley on the unhappy agents and their neighbors, using the average-consensus results, again, as our preference list.

VI. Simulation Results

To compare the performance of the algorithms discussed in this paper, we run MATLAB simulations on a trial lattice graph of size $N = 64$ (i.e., the topology of Fig. 1). We perform 1000 Monte Carlo trials of each algorithm to observe the convergence behavior, with each initial assignment of agents to nodes being randomly generated. For Intelligent Swap and Stable Marriage Swap, the "intelligent" swap was chosen with probability $p = 0.95$.

Fig. 3 shows the results of the Naïve Swap algorithm. We restrict the total number of swaps in each run to 3000, and thus the last histogram bucket represents those runs that took 3000 or more iterations to achieve convergence. We see that the distribution is spread over a large range – the algorithm is not very predictable. We note that given the random initialization, this distribution could be more indicative of the initial configuration than of the algorithm.

Fig. 4 shows the results for Intelligent Swap. We see the distribution has a clearer peak and requires, on average, fewer swaps than the Naïve Swap case. The number of average-consensus iterations required averaged about 208.

Fig. 5 shows the results for Stable Marriage Swap. The swap distribution seems to improve upon Intelligent Swap, and required fewer average-consensus iterations on average (about 145). We note, however, that we also require a number of iterations of the Gale-Shapley algorithm at each average-consensus iteration, increasing communication costs.

VII. Conclusion

We observed that the Intelligent Swap and the Stable Marriage Swap algorithms require, on our graph topology, substantially fewer swaps than the Naïve Swap algorithm. Furthermore, they perform much more predictably with a smaller spread than the purely random case.

We note that the utility of the algorithms explored in this paper depend on the relative cost of agent communication as compared to agent movement – if communication is faster and cheaper than movement, as seems likely for most physical applications, reducing the number of swaps at the cost of adding average-consensus iterations is logical.

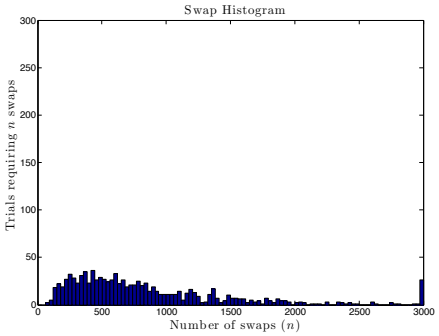

Fig. 3. Histogram of total number of swaps required for convergence of Naïve Swap. The last bin represents those trials that did not converge in 3000 iterations.

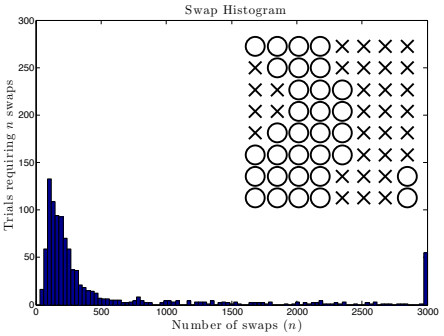

Fig. 4. Histogram of total number of swaps required for convergence of Intelligent Swap. Inset: an example ending configuration.

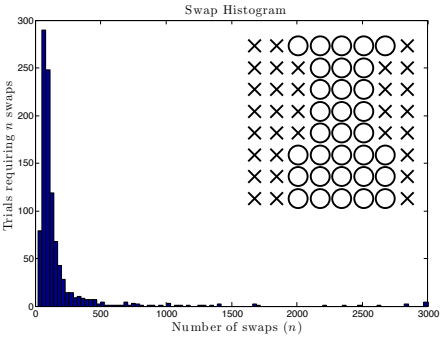

Fig. 5. Histogram of total number of swaps required for convergence of Stable Marriage Swap. Inset: an example ending configuration.

References

[1] V. Mhatre and C. Rosenberg, "Homogeneous vs heterogeneous clustered sensor networks: a comparative study," in *Communications, 2004 IEEE International Conference on*, June 2004, vol. 6, pp. 3646 – 3651.

[2] A.T. Erman, L.V. Hoesel, P. Havinga, and J. Wu, "Enabling mobility in heterogeneous wireless sensor networks cooperating with UAVs for mission-critical management," *Wireless Communications, IEEE*, vol. 15, no. 6, pp. 38 –46, December 2008.

[3] P. Fjallstrom, "Algorithms for graph partitioning: A survey," *Linkoping Electronic Articles in Computer and Information Science*, vol. 3, 1998.

[4] 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.

[5] J.L. Gross and T.W. Tucker, *Topological Graph Theory*, Dover Books on Mathematics. Dover Publications, 2001.

[6] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, pp. 2508 – 2530, June 2006.

[7] L. Xiao, S. Boyd, and S. Lall, "A scheme for robust distributed sensor fusion based on average consensus," in *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, Los Angeles, CA, April 2005, pp. 63–70.

[8] D. Gale and M. Sotomayor, "Some remarks on the stable matching problem," *Discrete Applied Mathematics*, vol. 11, no. 3, pp. 223 – 232, 1985.

[9] P. Floréen, P. Kaski, V. Polishchuk, and J. Suomela, "Almost stable matchings by truncating the Gale-Shapley algorithm," *Algorithmica*, vol. 58, pp. 102–118, 2010.

[10] S. Meyn and R. L. Tweedie, *Markov Chains and Stochastic Stability*, Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[11] U. A. Khan and S. Kar, "A decentralized algorithm for the preferred assignment problem in multi-agent systems," *in review*, 2012.