Marissa Graham

CS 670 Final

The 2018 paper "Should we compete or should we cooperate? Applying game theory to task allocation" attempts to frame the problem of distributed task allocation in a drone swarm in terms of the more general behavioral psychology and game theory question of "Should agents in a group compete or cooperate?". Unfortunately, while this is an intriguing idea, the paper fails to yield any meaningful results about the relative effectiveness of cooperation and competition. The methodology it describes is flawed and poorly justified; crucial implementation details are omitted and obvious concerns are not addressed, to the extent that it seems unlikely they were even considered by the authors. The authors present their results baldly, without any insights into why they occur, how the specific implementation limits their scope, or how they might generalize to other situations, and yet make the bold claim that they can assure "Dear robots, you should cooperate[1]!".

We now discuss the specifics of these flaws, and suggest how the approach might have been modified to do justice to the potentially interesting ideas this paper attempts[2] to explore.

Problem definition and success metrics

In the problem of multi-robot task allocation, we must assign N_T tasks to N_R robots in a way which maximizes some utility function, i.e., resources spent, distance covered, time required, quality of results, or some combination of these and other measures. The paper defines two algorithms for solving this problem, one of which it considers competitive and one which it considers cooperative. In both cases, each robot is connected with N_C neighbors, although it is unclear how these neighbors are chosen; one might assume the lack of clarification implies agents simply communicate with their nearest neighbors, but this does not appear to be the case in the graph of robot distributions in Figure 1.

The utility function used is "Social Utility (SU)", which attempts to reward robots for choosing the closest tasks to themselves, while penalizing robots for choosing the same task as other robots. The definition of this utility function, however, is inconsistent. The social utility of a task allocation is stated to be the sum of the utilities for each robot. The utility for a robot r performing a task t is defined to be $U(r,t) = 0$ if two robots perform the same task and $d_{max} - d(r,t) + 1$ otherwise, where d_{max} is a fixed hyperparameter and $d(r,t)$ is the distance between the robot and the task. But then the social utility is defined to be

$$SU = \sum_{i=1}^{N_R} \frac{d_{max} - d(i,t) + 1}{N_t},$$

where N_t is the number of robots performing the task t assigned to robot i. This is not the same as $\sum_{i=1}^{N_R} U(i,t)$, as originally stated; assigning multiple robots to the same tasks merely scales the utility received by each, rather than making it zero. We do not know which of these two definitions is actually used to calculate the SU results reported. Furthermore, this utility is not normalized by the number of robots in the swarm, meaning that its values for different values of N_T are not comparable. The performance of each algorithm with respect to SU is therefore not

[1] Even though they never actually define what constitutes cooperation and competition, or discuss why each of their algorithms should be considered one and not the other.

[2] feebly

a very meaningful metric, especially since the authors do not report variances of the data, only (presumably) means; we cannot be sure whether the better results of the competitive algorithm with respect to SU are due to chance, or whether they represent a true advantage.

Results are also reported with respect to the number of tasks completed. This metric is at least unambiguous, but it suffers from the same lack of reporting of variances. In many cases, the performance differences of the two algorithms are close enough to possibly be explained by variances in the data.

The paper does not explicitly restrict its algorithms to the $N_T = N_R$ case, but the authors do not run any experiments for any $N_T \neq N_R$ case, and do not discuss[3] how their methods would perform if some or all robots must perform multiple tasks. We therefore assume that their algorithms are only intended to apply to the $N_T = N_R$ case.

Algorithms

The authors present two algorithms for approaching the multi-robot task allocation problem:

- **Competitive.** Each robot gets information about its neighbors and its preferred tasks, searches for the best possible allocation of these tasks to itself and its neighbors (presumably using the genetic algorithm described later in the section), and then chooses the task it should perform under this allocation. How each robot's "preferred tasks" are defined is unclear; presumably they are simply the nearest tasks, but we do not know how *many* nearest tasks. These tasks could also be all those within a certain radius.

  This strategy is justified by the assertion that any permutation of tasks assigned is a Nash equilibrium, because no robot has a unilateral incentive to switch tasks after an assignment has been made, as their utility would then be zero. The problem with this justification is that it in no way helps robots choose a task in the first place. The true choice mechanism being used here is the assignment problem; calling it "the best Nash Equilibrium" shows a fundamental lack of understanding of why Nash equilibria are a useful solution concept.

- **Cooperative.** A certain number of robots are chosen to be "leaders", each of which is assigned N_C "citizens" whose "votes" it is responsible for counting. This assignment must partition the "citizens", else a robot might be assigned to perform multiple tasks. Which robots are chosen to be leaders is unclear, as is the method for connecting all robots to the same number of neighbors while still obtaining the necessary partition.

  Each leader determines a preferred allocation of the F_T closest tasks and asks its assigned "citizens" for their preferred allocation of these tasks. These votes are determined using the genetic algorithm described for allocation. It is not clear whether citizens vote to allocate these tasks among their own neighbors or among the leader's neighbors. This poses a problem, which is not addressed by the authors; if each citizen votes to allocate among its leader's neighbors, each vote will essentially be the same, but if each vote is done with respect to a different set of agents, combining votes is less meaningful.

  The social choice mechanism used in this case is combining local solutions of the assignment problem using different voting schemes. This is a less strained connection to game theory than "searching for the best Nash Equilibrium", but still not a natural fit for the problem. Having agents all vote for their preferences is an order of magnitude more expensive than and yet unlikely to be much more effective than an algorithm where each leader assigns

[3] Or seem to have even considered

tasks based on their own local solution to the assignment problem. The centralization is what makes this algorithm effective, not the voting.

The genetic algorithm described by the authors to allocate preferences seems reasonable enough in its own right, but it is almost certainly a poor choice. It has not been thoroughly tested–the authors only run experiments for 10 robots and 10 tasks, and again do not report variances–but the bigger problem is that the authors do not seem to know that what their algorithm is attempting to solve is the assignment problem.

They claim that the allocation of 10 tasks to 10 robots generates a space of 3,522,880 permutations[4] of 10 numbers. It is true that a brute force algorithm for the assignment problem would have an $O(n!)$ time complexity, but the Hungarian algorithm is well-known, has existed since 1955, and solves the problem in maximum $O(n^3)$ time[5] For swarms of at most 400 robots, this is more than suitable. If a work which involves solving the assignment problem neither uses the Hungarian algorithm (or a variant) nor justifies why their method is cheaper, it is unlikely to be worth taking seriously.

The Hungarian algorithm does require a square matrix (i.e., the number of tasks must be the same as the number of assignments), which may not be the case for these allocations. Each robot has N_C neighbors, but F_T can vary and seems to always be higher[6] than N_C. However, this is not an excuse for using an original algorithm without justifying its use; an extension of the Hungarian algorithm to rectangular matrices was published[7] by Bourgeois and Lassale in 1971.

Results and discussion

The authors test their two algorithms for varying numbers of F_T, various voting schemes for the cooperative algorithm, varying numbers of agents, and varying number of neighbors. In each case, they do not specify the values for the parameters which are not being varied, and they report results in opaque tables that really should have been plots.

The author's explanation for how they chose F_T is problematic; they simply state that the citizens each leader coordinates know $2N_C + 1 = 21$ different robots, which is both unjustified and implausible. They note that the more tasks assigned to each leader, the fewer tasks which are completed. This is unsurprising, since assigning more tasks to each leader means that each task is being considered by more leaders, which results in more robots being assigned to the highest-utility (nearest) tasks. This also explains the "curious result" (which the authors do not attempt to explain) that in the "realistic application test", the cooperative robots were grouped along the route of the aircraft. The authors suggest throwing more drones, dispersing the drones more widely, or running the algorithm multiple times to solve this problem, but a better approach would be to just use a different algorithm.

There really is very little to recommend either of these algorithms. They are expensive, their performance is mediocre, they only apply to a very specific case, and they attempt to apply game theory concepts to a different domain without truly understanding them or why they are effective in their original domain. Both methods require agents to know how far away they are from each task, as well as how far away their neighbors are from those same tasks, and yet struggle to complete 70% of tasks even with one robot for every task that must be completed. The higher-performing of

[4]This is incorrect; it should be 10! =3,628,800.

[5]And a single line of Python code, by calling `scipy.optimize.linear_sum_assignment`.

[6]Note that the authors only test their algorithm in the square case and do not discuss how it behaves differently in the non-square case.

[7]Which is incorporated into `scipy.optimize.linear_sum_assignment`.

the two is drastically more expensive and not fully decentralized. The conclusion that cooperation is an effective strategy is still most likely true, but this paper only supports it weakly if at all.

To more effectively investigate this hypothesis, we need to determine what constitutes "competition" and "cooperation" beyond a strained analogy about robots "competing" for tasks, or "cooperating" by delegating their decision-making ability to a designated leader–hierarchy is not cooperation. The more relevant distinction, in my opinion, is whether agents make decisions unilaterally based on their own interests, or whether they are willing to sacrifice some of their own utility in order to achieve an overall better societal results. In other words, competition and cooperation are simply defection and cooperation in a giant prisoner's dilemma.

Individual drones in a swarm do not truly have their own preferences, or care about maximizing their own utility. Incentivizing drones to maximize their own utility is simply a mechanism for decentralizing the decision making process of a swarm and minimizing the resources it uses to complete its tasks. With this in mind, it is *absurdly* obvious that drones should cooperate; the relevant question (which this paper completely ignores) is how they should do so.

The main difficulty with task assignment in a decentralized swarm (and therefore the goal of cooperation) lies in ensuring that all tasks are completed with minimal duplicate effort. This paper makes the implicit assumption that the correct way to ensure this is to have all agents do their best to find the "best assignment" (i.e., the most efficient assignment in terms of resources, while penalizing duplicate effort); presumably then everyone will agree on it.

However, you'd likely be far better at getting all the tasks completed in less time with far less communication if you simply had agents make their decisions asynchronously (but in quick succession), with agents who choose later updating their preferences accordingly based on what's left. In real groups who are splitting up tasks, assignments aren't all made simultaneously; people with stronger preferences tend to choose according to their preferences first, and then everybody else sort of just picks something.

You could mimic this by setting an increasing threshold where agents within a certain distance of a task simply take that task, at which point their neighbors remove that task from consideration, until all agents have chosen a task. In this scenario, agents would only choose the same task if they were both within a certain nearness threshold of it at the same timestep. Alternatively, if we prefer the "leader" model, we could have the leaders make their decisions first (greedily). Other agents would not make a decision until at least one of their neighbors has made a decision, at which point they choose a (still unassigned) task greedily. If each agent only has a few neighbors, this should help significantly with avoiding duplicate assignments, while avoiding the need to implement a nearness threshold.

Both of these strategies are inspired by the very common "seed-and-extend" strategy for global protein-protein interaction (PPI) network alignment. All algorithms for global alignment of PPI networks use an assignment problem approach, but few use the Hungarian method to create a mapping; greedy methods are even cheaper and often perform even better than the more principled method, and it is likely that this would also be the case here.