Active Learning to Solve Combinatorial Optimization Problems in Adversarial Environments

1 Task Assignment Problem

We consider the task assignment problem with N agents in the set $\mathcal{N} = \{1, \cdots, N\}$. There are K tasks in the set $\mathcal{K} = \{1, \cdots, K\}$. Each task can be assigned to multiple agents. In order for each task k to be completed, a total of E_k units of energy is required (energy could stand for time, money or any other resource that has to be utilized for the task completion.). If task k is allocated to agent i, she can exert $A_{i,k}$ units of energy towards completion of that task. $A_{i,k}$ accounts for the different levels of expertise of agents for different tasks. For the assignment of each task k to agents, we split the E_k units of energy that it requires into E_k individual parts and each agent i that has been assigned to this task completes $A_{i,k}$ units. Each agent i has a total of B_i units of energy (her total budget). We define the binary assignment vector $\boldsymbol{x} = (x_{i,k})_{i\in\mathcal{N},k\in\mathcal{K}} \in \mathcal{X} = \{0,1\}^{N\times K}$, where $x_{i,k} = 1$ if task k is assigned to agent i and 0 otherwise. We also denote the variables corresponding to the tasks assigned to agent i by $\boldsymbol{x}_i = (x_{i,k})_{k\in\mathcal{K}}$. For the assignment vector $\boldsymbol{x}$, agent i pays the cost $f(\boldsymbol{x}_i; \theta_i)$, where we have considered a parametric cost function with parameter θ_i. The goal is to choose the assignment $\boldsymbol{x}$ to minimize the social cost, i.e., $\sum_{i\in\mathcal{N}} f(\boldsymbol{x}_i; \theta_i)$. The optimization problem is formalized as follows.

$$\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{i\in\mathcal{N}} f(\boldsymbol{x}_i; \theta_i) \tag{1a}$$

$$s.t. \quad \sum_{i\in\mathcal{N}} x_{i,k} A_{i,k} \geq E_k, \quad \forall k \in \mathcal{K} \tag{1b}$$

$$\sum_{k\in\mathcal{K}} x_{i,k} A_{i,k} \leq B_i, \quad \forall i \in \mathcal{N} \tag{1c}$$

As a special case, note that if the cost functions $f(\boldsymbol{x}_i; \theta_i)$ are linear, then we have $f(\boldsymbol{x}_i\theta_i) = \sum_{k\in\mathcal{K}} x_{i,k}\theta_{i,k}$, where $\theta_{i,k}$ is the cost that agent i incurs if she is assigned to task k.

An example of this task assignment problem can be seen in distributed computation where some computational tasks need to be assigned to a number of processors. Each processor can have a different speed and capacity for conducting each task. The goal could be to do the computations in the minimum possible time[1]. Another example is a network of drones or robots, where the tasks include monitoring different sections of the target area, and possibly delivering some items to a destination. Depending on the position and capacities of each drone, they can have different levels of contribution in completing the tasks.

2 Human Bot Teaming

The agents in this problem could either be human agents or bots. A human agent naturally pays higher costs for completing a task but can contribute to some tasks better than a bot. On the other hand, the cost function of a human agent is not known and needs to be learned. More specifically, the cost function parameters, θ_i, of human agents are not known and they have to be learned by

[1]Notice that in our formulation the costs of agents are added to each other, while if we consider a parallel computation setting and the cost stands for the time it takes to complete the task, the costs of agents might not be added to each other.

observing the value of these cost functions at some candidate solutions. That is, if an assignment has been done to the agents, each agent will receive a feedback about her cost after completing the tasks and through these feedbacks, the cost functions are learned. This is similar to the centralized (single agent) optimization of an unknown cost function that is studied in [1].

3 Distributed Reinforcement Learning Approach

We can leverage the distributed nature of the social cost function ($\sum_{i\in\mathcal{N}} f(\boldsymbol{x}_i;\theta_i)$) and propose distributed algorithms to approximately find the solution of the optimization problem. The social cost function allows for parallel computation of cost functions. For example, each agent can compute the gradient of her own cost function and their summation will form the gradient of the social cost function.

There is also a privacy concern about the cost functions of human agents. These agents are not willing to share their learned cost function with others and they prefer to do some computations locally and share the global learned model as opposed to their private cost function. A distributed reinforcement learning (or federated reinforcement learning) algorithm can preserve the privacy constraints of agent.

On the other hand, in many applications such as drone or robot networks there is no centralized controller and this is another reason that we are interested in distributed learning algorithms. Furthermore, by proposing a distributed algorithm, we will improve the fault tolerance and robustness of the learning algorithm to agent failures and adversarial attacks. We will further discuss about this in the next section.

We assume the agents are distributed over a graph and the process of learning the solution will be a distributed deep reinforcement learning type algorithm where agents choose their assignments based on their local information and the learned model so far. Because of the feasibility constraints, this setting can be considered a constrained distributed deep reinforcement learning problem.

4 Adversarial Attacks and Agent Failures

In this section, we describe the different agent and communication failures that we consider in our setting. The agent failure can happen due to adversarial attacks to the bots and as a result, the compromised bots can not either complete their assigned tasks, or might try to poison the learning algorithm by sharing malicious updates.

We also consider human agents being compromised due to natural human errors in completing the task. The difference between these human agent errors and malicious inputs from compromised bots is that human errors are unintentional and they are not consistent through time, while malicious inputs from compromised bots are consistent.

Communication failures throughout the learning process are also taken into account. The communication failures are translated in some links in the graph being dropped. This can happen due to adversarial attacks, environment change or mobility of agents.

5 Active Learning of Compromised bots

In order to mitigate the threats from compromised bots to the learning algorithm and task completion process, we need to identify these bots and remove them from both the learning process and

task assignment. The network actively learns about the compromised bots by having the human agents evaluating their neighboring bots and identifying (labeling) the compromised ones. This process can be thought of as a new set of tasks that can be assigned to the human agents. The human agents have different levels of expertise in identifying malicious bots and therefore, will incur different costs by doing this labeling. The labels will pass along the network together with the other model updates.

We can incorporate the active learning of the bot labels into the optimization problem (1) as follows. We assume each bot i is compromised with probability p_i. In this formulation, we assume the compromised bots will not complete any task assigned to them. We denote the set of human agents by $\mathcal{N}_h$. Each human agent i can label her neighboring bots and we define $G_{i,j} = 1$ to indicate the ability of agent i to label bot j (whether agent i is connected to bot j) and we set $G_{i,j} = 0$ if agent i is not connected to bot j. If agent i is assigned to label bot j, we set $l_{i,j} = 1$. We denote $\boldsymbol{l} \in \mathcal{L} = 2^{N_h \times (N - N_h)}$ to be the vector of labeling assignments where $N_h = |\mathcal{N}_h|$. We define $y_i = 1$ if bot i is labeled and 0 otherwise. Therefore, we have $y_j = 1$ is $\sum_{i \in \mathcal{N}_h} l_{i,j} G_{i,j} \geq 1$. Each agent i incurs a cost $C_{i,j}$ for labeling bot j. There is a cost F_k for each unit of task k not getting completely done because of some unexposed compromised bot that is assigned to that task. We have the following reformulation of the optimization problem.

$$\min_{\boldsymbol{x} \in \mathcal{X}, \boldsymbol{l} \in \mathcal{L}} \sum_{i \in \mathcal{N}} f(\boldsymbol{x}_i; \theta_i) + \sum_{i \in \mathcal{N}_h} \sum_{j \in \mathcal{N} \setminus \mathcal{N}_h} C_{i,j} l_{i,j} G_{i,j} + \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}} p_i F_k x_{i,k} A_{i,k} (1 - y_i) \tag{2a}$$

$$s.t. \quad \sum_{i \in \mathcal{N}} ((1 - p_i) y_i + (1 - y_i)) x_{i,k} A_{i,k} \geq E_k, \quad \forall k \in \mathcal{K} \tag{2b}$$

$$\sum_{k \in \mathcal{K}} x_{i,k} A_{i,k} \leq B_i, \quad \forall i \in \mathcal{N} \tag{2c}$$

$$y_j = \delta(\sum_{i \in \mathcal{N}_h} l_{i,j} G_{i,j} \geq 1) \tag{2d}$$

where we define $\delta(a) = 1$ is a is true and $\delta(a) = 0$ otherwise. Notice that since the optimization is done before knowing any of the labels of the agents, using the prior probability of each node being compromised, the average of the costs and energies spent towards completion of tasks are considered in the optimization problem.

6 Task Assignment from Team Science Point of View

The task assignment problem to a number of agents can be studied from the team science point of view. Depending on the composition of the team, i.e., which agents are assigned to a given task, their performance can be different. The team functionality is affected by how many agents are in the team, how the team is constructed, e.g., what the proportion of bots versus humans is, how they are communicating with each other, e.g., what their position over a graph is (what their social network looks like) and some other team features. The different performance of each team can be quantified by how the energies of the teammates are aggregated towards completion of the task in equation (1b). In equation (1b) in optimization problem (1), the energies of agents assigned to each task are added to each other. Depending on the functionality of the team, this aggregation function might not be linear and it could some more complex nonlinear function. If

the agents do not interact well with each other, their energies will be wasted towards some team malfunctioning. Thus, the aggregation function will be below the linear function. On the other hand, if the teammates interact well, they might level up each other and therefore, the aggregation function could be above the linear function. We can learn this aggregation function from a team science point of view by experimenting on the team compositions and its effects on the overall team performance.

On the other hand, if we can identify the sources of the different team functionalities, we can incorporate that knowledge in the task assignment problem. That is, if we know which team composition will result in the most effective aggregation function, then we can add some constraints to the optimization problem to ensure those requirements are met.

References

[1] P. Campigotto, A. Passerini, and R. Battiti, "Active learning of combinatorial features for interactive optimization," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 336–350.