Dynamic Partition of Collaborative Multiagent Based on Coordination Trees
Min, F.; Groen, F.C.A.; Hao, L.

Published in:
Advances in intelligent systems and computing

DOI:
10.1007/978-3-642-33932-5_46

Citation for published version (APA):
Dynamic Partition of Collaborative Multiagent based on Coordination Trees

Fang Min¹, Frans C.A. Groen² and Li Hao¹
¹Institute of Computer Science
Xidian University, China
mfang@mail.xidian.edu.cn
²Informatics Institute
University of Amsterdam, The Netherlands
F.C.A.Groen@uva.nl

Abstract. In team Markov games research, it is difficult for an individual agent to calculate the reward of collaborative agents dynamically. We present a coordination tree structure whose nodes are agent subsets or an agent. Two kinds of weights of a tree are defined which describe the cost of an agent collaborating with an agent subset. We can calculate a collaborative agent subset and its minimal cost for collaboration using these coordination trees. Some experiments of a Markov game have been done by using this novel algorithm. The results of the experiments prove that this method outperforms related multi-agent reinforcement-learning methods based on alterable collaborative teams.

Keywords: reinforcement learning, multi-agent, coordination tree, Markov games

1 Introduction

A Markov game, also known as a stochastic game, is generally considered as the combination of a Markov decision process and a matrix game[1]. When multiple agents deal with a Markov game, there are a number of relative independent actions or action series that can be taken[2][3]. For example in a pursuit game, predators need to explore their objects and then to hunt evaders together. In multi-robot team coordination, the robots need to form teams and coordinate other actions dynamically. So, we need an approach to judge the cooperative relationship among agents.

The learning of multiple agents is a complex problem. The collaborative multi-agent Markov decision process[4][5][6] model is often used in research. Hyper-Q learning[7] learns mixed strategies in stochastic games, using observations of other agents’ play. The states in Hyper-Q consist of observations or estimates of opponents’ strategies. By learning a value function of state-action pairs, the Hyper-Q algorithm obtains its mixed strategies. Another solution technique is based on the framework of coordination graphs (CGs)[8]. In a CG each node represents an agent and connected agents indicate a local coordination dependency. Each dependency corresponds to a local payoff function which assigns a value to every possible action combination of the involved agents. The Max-Plus algorithm and Variable
Elimination(VE)[8] based on coordination graphs are used to find the optimal joint action. These algorithms demand that the structure of the CG used is determined beforehand. It is difficult for agents to calculate dynamic neighbors and to select an optimal action. If the neighbors of an agent have a tree structure, the Max-Plus algorithm based on coordination graphs can optimize the global reward. When the neighbors of an agent have a graph structure, Max-Plus algorithm can not cope with the complexity of the underlying graph structure[9].

In summary, an optimal joint action selection method is needed to copy with the complicated relation of agents. Therefore, we present a dynamic distributed selection algorithm of a joint action based on coordination trees. Using this method, the multi-agent reinforcement algorithm of team Markov games could be applied.

2 Coordination trees for describing the collaborative relationship among agents

Suppose n agents are present in two fighting sides. The number of cooperative agents in one side is $N_b$, the number of opposite agents is $N_r$. One of the opposing sides need $k$ agents to collaborate ($k \leq \min(N_b,N_r)$) and take care of an agent of the other side. A key problem is how to obtain the cooperative agents of one agent.

We define a coordination graph which describes a global cooperative relation between agents. In a coordination graph, there are two kinds of nodes. The first one is a node for the agent $A_g$, and the second one is a set node $B_j$ which don’t include the agent $A_g$. The tree structure is shown in fig. 1.

![Coordination Tree](image)

**Fig. 1.** The coordination tree of $A_g$.

**Define:** Suppose that we have a coordination tree $Tr=(V,E)$ with $|V|$ vertices and $|E|$ edges, where $V = \{B \_Set,A_g \_Set\}$, $A_g \_Set = \{A_g,i=1,2,...,N_b\}$, and $Set \_B = \{B_j,j \in [1,\ldots,C^{k-1}_{\max}]\}$.

There are not edges between two agents or two set nodes. An agent node can only link set nodes, and a set node has only sides with agent nodes. The side between node
denotes that the agent \( \text{Ag}_i \) cooperates with the agents in the set \( \text{B}_j \).

All coordination trees of agents are defined based on a coordination graph. For an agent \( \text{Ag}_j \), a tree with root \( \text{Ag}_i \) is defined, which has three layers of nodes and describes the relation of the collaborative agents. The tree structure is shown in fig. 1.

1. A root node, \( \text{Ag}_i \), is an agent, which need to select an agent set \( \text{B}_j \) to collaborate with.

2. Each child node of the root is an agent subset \( \text{B}_j \) is an agent subset which may collaborate with \( \text{Ag}_i \). \( \text{CoN}(\text{CoN} \leq \text{Nh}) \) is the number of the agents in the view of the agent \( \text{Ag}_j \). \( \text{CoN} \) is the number of potential subsets collaborating with the agent \( \text{Ag}_i \). For example, there are \( n \) agent sets collaborating with \( \text{B}_j \) in figure 1, in which \( n \) is equal to \( C_{\text{CoN}}^{k-1} \).

\[
\text{B}_j = \{ \text{Ag}_i \mid \text{Ag}_i \in \text{Ag}_\text{Set} \text{ and } \text{Ag}_i \neq \text{Ag}_j, i = 1,2,\ldots,k-1 \}
\]  

Where the set \( \text{Ag}_\text{Set} \) is an agent set. An agent \( \text{Ag}_i \) needs to select \( k-1 \) agents to collaborate with. A tree with a root \( \text{Ag}_i \) is built and has sub nodes \( \text{B}_j \), and \( \text{Ag}_i \neq \text{B}_j \).

3. A team \( \overline{B}_j \) is formed with \( \text{Ag}_i \) and \( \text{B}_j \).

\[
\overline{B}_j = \text{B}_j + \{ \text{Ag}_i \}
\]

(2) A team \( \overline{B}_j \) is formed with \( \text{Ag}_i \) and \( \text{B}_j \).

(3) Each node of tree in the third layer is an agent which may collaborate with agents in \( \text{B}_j \).

An agent \( \text{Ag}_\beta \) might collaborate with subset \( \text{B}_j \), where \( \text{Ag}_\beta \in \text{B}_j \cap \text{Ag}_\beta \neq \text{Ag}_i \).

(4) Two kinds of weights are present.

The weight \( \omega \) on the edge of a tree denotes the minimal collaborative cost of an agent with a subset. The weight \( w \) is the minimal collaborative cost of a subset collaborating with another agent than \( \text{Ag}_i \). For node \( \overline{B}_j \) in the second layer, the weight \( \omega_{\overline{B}_j} \) denotes the collaborative cost of \( \overline{B}_j \) with \( \text{Ag}_i \). The weight \( w_{\beta_j} \) is the collaborative cost of \( \overline{B}_j \) collaborating with an agent \( \text{Ag}_\beta (\text{Ag}_\beta \neq \text{Ag}_i) \).

We need now to select an optimal collaborative subset for \( \text{Ag}_j \) based on coordination trees.

\[\text{Fig. 2a. Communication model of the agent } \text{Ag}_j\]

\[\text{Fig. 2b. Communication model of the agent subset } \text{B}_j\]
(5) Communication modes between an agent and agent subsets.

The communication model of the agent $A_{i}$ is showed in fig.2a, and the communication model of the subset $B_{k}$ is showed in fig.2b. Each agent $A_{i}$ sends collaborative cost or a payoff $\omega_{i,j}$ to the subset $B_{j}$ ($A_{i} \notin B_{j}$) which connect to $A_{i}$ by a edge. The subset $B_{j}$ will return a minimal collaborative cost or a maximal payoff $w_{i,j}$ to $A_{i}$ collaborating with one other agent except the agent $A_{i}$.

3 Calculating a collaboration for an agent

3.1 Dynamic partition of collaborative subsets

We present an algorithm for an agent based on this coordination tree structure. An agent $A_{i}$ can select its collaborating agent subset according to the algorithm CSCT below.

**Algorithm: CSCT- calculating Collaborative Subset based on Coordination Tree**

1. The agent $A_{i}$ sends a cost $\omega_{i,j}$ to each agent in $B_{j}$, which denotes the collaborative cost with $k-1$ agents in subset $B_{j}$.

2. When $B_{j}$ receives a cost $\omega_{i,j}$, it returns a message $w_{i,j}$ to $A_{i}$ which is the cost of $B_{j}$ collaborating with another agent. For example, the subset $B_{j}$ on the second layer in fig.1 selects a potential collaborative agent $A_{j,m}$ instead of $A_{i}$ at the cost:

\[
\omega_{i,j} = \min\{\omega_{m,j}\}
\]

Where $A_{j,m}$ is one child node of $B_{j}$, and the $\omega_{m,j}$ is the weight between the node $B_{j}$ and the node $A_{j,m}$.

3. The agent $A_{i}$ selects the optimal collaborative subset $B_{j}$.

Because each agent learns joint actions rationally, it makes decision for a maximal reward of all agents. Each agent can estimate the possible actions of other agents based on a coordination tree. The rate $\omega_{i,j} / w_{i,j}$ is a probability measure that denotes $B_{j}$ collaborating with $A_{i}$. The $B_{j}$ with minimal $\omega_{i,j} / w_{i,j}$ value is the appropriate subset to collaborate with $A_{i}$.

\[
B_{j} = \arg\min_{B_{j}} (\omega_{i,j} / w_{i,j})
\]
3.2 Policy of joint actions

After calculating the collaborative subset based on the coordination tree, the joint action in a team $\mathcal{B}_j$ should be decided. In multi-agent's learning procedure, the system's state switching is determined by the actions of the learning agent and other agents together. In most cases, the agent’s action in a certain state is a random behavior which obeys some probability distribution\[10\][11][13]. Thus, the learning agent can model the beliefs of other relevant agents by observing their behavior histories. Other agents' strategies obey certain probability distributions, which can be partially determined by prior knowledge and observations, thus their strategies can be estimated. Therefore, through observation and statistical analysis in the learning process, one agent can learn the strategies of other agents and understand their influence on the environment[12][14]. Through making use of the estimated probabilities of other agents’ actions to ensure the selection of optimal joint actions, members in teams take the best response actions according to other agent’s actions in the same environment[15]. Every cooperative team learns independently, multiple cooperative teams can prevent the problem of dimensionality to some extent. Single agent is able to learn in a multi-agent framework. The joint action in a team $\mathcal{B}_j$ is determined by using the behavior probability estimation and joint action statistic.

4 Experiment setup and simulation

4.1 Environment and agent setting

There are $n$ agents for the red side and the blue side fighting each other. $Nb$ is the number of agents in the blue side, and $Nr$ is the number of agents in red side. Only surrounded by $k$ opposite agents simultaneously an agent can be destroyed. We use a two-dimensional plane as the experimental environment. Each agent has a view field $\text{vision}_v$. Blue side agents perform according to a reinforcement learning method, the agents of red side act according to specific policy and they know nothing about the action policy of each other. For a blue side agent, the threat distance of a red side agent is $d$, where $d \leq \text{vision}_v$, which means that the distance between two opposite agents is larger than the threat distance $d$ in the view, there is no threat each other.

4.2 Parameter setting

We design two experiments and analyze the performance of the above methods. The first experiment has a $15 \times 15$ grid, with four red side agents versus four blue ones. The second experiment has a $20 \times 20$ grid, with four red side agents versus five blue ones. When a red side agent was destroyed by the blue side team, a reward $r$ is given. When a blue side agent is destroyed by the red side team, a reward -$r$ is given as punishment. The exploration rate is denoted as $\varepsilon$. In order to prove the suitability of the
algorithm presented in this paper, we compare our method to a nearest neighbor approach. In that approach teams of collaborative agents are formed with the closest other agent.

In our experiments, the parameters reward, the discount $\gamma$, the exploring rate $\varepsilon$ are set to 1000, 0.9, 0.3 respectively, and the number of collaborative agent is all set to 2. The attenuation factor is set to 0.6. The action set is \{stop, up, down, left, right\}. The current state $s$ is the agent’s location. At the beginning of each simulation, agents start from different positions, and the game ends when all opposite agents are caught successfully. The h-step back method is used to redistribute the reward. That means the reward $\beta^hr$ of the final state is regarded as the reward of the h-step back, where $\beta$ is an attenuation factor. The number of back states should less than the size of view, therefore, $vision_v \geq h$.

![Graph](image)

**Fig. 3.** Comparison of 4 versus 4 agents in a 15×15 grid

In fig.3, the horizontal axis shows the number of learning iterations, the vertical axis shows the number of steps needed at each capture. For showing and analyzing the experimental result conveniently, we draw the average value per 4 time steps in the learning periods. Each experiment is repeated 20 times. From the first experimental result in fig.3, the Q-learning of team Markov games based on coordination trees could converge to the optimal value finally. It is observed that, the closer $d$ and $h$ are to $vision_v$, the better results we get. When $d$ is close to $vision_v$, blue side agents can easily sense the threat from the red side agent in a larger view. The bigger $h$ is, the more previous states the rewards feed back through the Q-table, thus accelerating the convergence.
From the second experiment results in fig. 4, we find that the policy based on coordination trees is better than the method of selecting the nearest neighbors. When using the coordination tree method, $A_{ij}$ calculates the probability of collaborating with other agents based on communication, the agent subset with a maximum probability was selected as its cooperation object. So each agent can select its cooperation object freely according to observations on the environment and the behavior probability estimation and the nearest agents of $A_{ij}$ don’t necessarily select $A_{ij}$ as its cooperation object.

### 5 Conclusion

In the study of Markov game, it is difficult to form the team of cooperating agents dynamically and to decide on the joint action of agents. On the basis of multi-agent reinforcement learning with agent teams in Markov games, we present a cooperation tree-structure by using the subset of cooperation agents as the nodes of a tree. Two kind of weights are defined which describe the cost of an agent collaborating with or without an agent subset respectively. Each agent calculates its collaborative agent subset with a minimal cost based on coordination trees. From the results of our experiments in a simulation environment, the collaborative team calculating and joint action learning of multi-agent based belief model can all improve the algorithm performance.

### Acknowledgements.

Many thanks are given to Prof. Shimon Whiteson in Amsterdam University who gives some helpful suggestions for this paper.
References