Article

Multi-Agent Planning under Uncertainty with Monte Carlo Q-Value Function

Jian Zhang ¹, Yaozong Pan ¹, Ruili Wang ², Yuqiang Fang ^{1,3,*} and Haitao Yang ¹

¹ Space Engineering University, Beijing 101416, China; zjconquer@126.com (J.Z.); zhangjuaner123@163.com (Y.P.); haitaoyang79@126.com (H.Y.)

² Institute of Natural and Mathematical Sciences, Massey University, Albany, Auckland 0632, New Zealand; r.wang@massey.ac.nz

³ State Key Laboratory of Complex Electromagnetic Environment Effects on Electronics and Information System, Luoyang 471003, China

* Correspondence: fangyuqiang@nudt.edu.cn

Received: 12 February 2019; Accepted: 30 March 2019; Published: 4 April 2019

Abstract: Decentralized partially observable Markov decision processes (Dec-POMDPs) are general multi-agent models for planning under uncertainty, but are intractable to solve. Doubly exponential growth of the search space as the horizon increases makes a brute-force search impossible. Heuristic methods can guide the search towards the right direction quickly and have been successful in different domains. In this paper, we propose a new Q-value function representation—Monte Carlo Q-value function Q_{MC} , which is proved to be an upper bound of the optimal Q-value function Q^* . We introduce two Monte Carlo tree search enhancements—heavy playout for a simulation policy and adaptive samples—to speed up computation of Q_{MC} . Then, we present a clustering and expansion with Monte-Carlo algorithm (CEMC)—an offline planning algorithm using Q_{MC} as Q-value function, which is based on the generalized multi-agent A* with incremental clustering and expansion (GMAA*-ICE or ICE). CEMC calculates Q-value functions as required, without computing and storing all Q-value functions. An extended policy pruning strategy is used in CEMC. Finally, we present empirical results demonstrating that CEMC outperforms the best heuristic algorithm with a compact Q-value presentation in term of runtime for the same horizon, and has less memory usage for larger problems.

Keywords: multi-agent; uncertainty; Monte Carlo; Q-value function; Dec-POMDP

1. Introduction

Cooperative decision-making under uncertainty is a common problem in many scenarios such as sensor networks and airport service robots. Consider a scenario where a robot soccer team cooperates to beat the opponent. A total of 11 robot players dash in a simulated field, kicking, passing, and shooting a ball to score a goal. Each of them could only acquire his local observation and every action taken is uncertain, which means passing may be unsuccessful or shooting may be biased. Another example is congestion control, which is an important task in the Internet. The router's buffer accommodates transient packets to guarantee a stable network link. The distributed routers need to come up with a strategy to avoid congestion. These kinds of problems can be modeled as a decentralized partially observable Markov decision process (Dec-POMDP), which provides a general multi-agent model for decision-making under uncertainty. However, Dec-POMDPs are provably intractable (NEXP-Complete, nondeterministic exponential time, it is known that $P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP$) [1] and difficult to solve with the number of joint policies growing explosively [2]. One of the primary reasons is that the search space of Dec-POMDPs grows exponentially with the

number of actions and observations, and doubly exponentially with the horizon of the problem [3]. The optimal policy can be extracted from the optimal Q-value function Q^* for Dec-POMDPs, but solving Q^* is also proved to be NEXP-Complete [4].

In a previous work [5], Monte Carlo tree search (MCTS) [6] is introduced into the MMDP (Multiagent MDP) model, which is a simplified model for Dec-POMDPs, to solve ad hoc agent teams decision-making. MCTS is a sample-based tree search algorithm using Monte Carlo simulations as state evaluations, which has been successful in Computer Go [7]. MCTS will converge to the optimal value if exploration (find new nodes that are not sampled) and exploitation (find promising nodes that are already sampled) are traded off appropriately when given sufficient time and can be stopped at any time to get an approximate result.

To tackle the doubly exponential problem, we propose an offline algorithm called a clustering and expansion with Monte-Carlo algorithm (CEMC) based on incremental clustering and expansion (ICE), a version of generalized multi-agent A* (GMAA*) with incremental clustering and expansion, which is the best heuristic method in this literature. MCTS is integrated into the CEMC framework to fast obtain the approximate Q-value function. MCTS is an anytime algorithm, and its efficiency is up to sample times and the degree of accuracy of the Q-value function evaluation. Thus, we propose an adaptive sample method according to the horizon and the number of joint actions and observations. The best upper bound found in the MCTS algorithm is used to prune the nodes that will not be the optimal ones in any case. The term MCTS in this paper denotes the specific algorithm that uses UCB1 (a version of upper confidence bound) as the tree selection policy proposed by Kocsis and Szepesvári [6,8].

The key contribution is twofold: We present a new Q-value function representation—Monte Carlo Q-value function Q_{MC} and an offline planning algorithm CEMC. Heavy rollout for simulation policy and adaptive sample are introduced to speed up the computation of Q_{MC} . The algorithm CEMC calculates the Q-value functions as required, without computing and storing all Q-value functions and it uses an extended pruning strategy for the policy pool. We present empirical results demonstrating that CEMC outperforms the best heuristic algorithm with a compact Q-value presentation in term of runtime for the same horizon, and has less memory usage for larger problems.

The remainder of this paper is structured as follows. We first introduce some related work in Section 2. Then, we outline some background knowledge on Dec-POMDPs, ICE algorithm, approximate Q-value functions, MCTS, and collaborative Bayesian Games in Section 3. Section 4 gives the Q-value function representation and proves that Q_{MC} is an upper bound of the optimal Q-value function. Then, an adaptive sample and heavy rollout technique are proposed. Section 5 presents the core algorithm CEMC and a pruning strategy. We empirically prove the efficiency and effectiveness of CEMC in Section 6 and discuss the results. We conclude and outline the future research in Section 7.

2. Related Work

The Dec-POMDP is an extension of the POMDP model which has been proved to be doubly exponential [1]. The brute force search becomes intractable when the number of agents or the problem size grows. In recent years, a lot of work has focused on fast computing an optimal policy of larger horizon, such as dynamic programming [9–12], heuristic search [2–4,13–15], and mixed integer linear programming [16].

GMAA* framework [4] is one of the state-of-the-art exact algorithms in the heuristic search literature, which is based on multi-agent A* (MAA*) [2] and uses an approximate heuristic function to guide the search in the whole search space. GMAA*-ICE (or ICE) [3] is a heuristically guided algorithm based on a GMAA* framework [4] and has demonstrated impressive performance over GMAA*. Two successful components used in ICE are lossless clustering [13] and incremental expansion [14]. Recently, a Bayesian learning method [17] and approximate inferences and heuristics [18] are proposed for solving Dec-POMDP.

Silver and Veness [19] proposed a Monte-Carlo algorithm—POMCP—for online planning in large POMDPs. Unlike our approach, POMCP combines a Monte-Carlo update of the agent's belief

state with a Monte-Carlo tree search from the current belief state. Importantly, POMCP is suitable for single-agent partially-observable and unfactored Markov decision process (MDP). Amato and Oliehoek [20] proposed a scalable approach based on sample-based planning called factored-value partially observable Monte Carlo planning (FV-POMCP), to combat the intractability of an increasing number of agents. However, FV-POMCP requires a decomposition of the value function into a set of overlapping factors.

3. Background

Consider a scenario where a number of agents inhabit in a particular environment modeled with a Dec-POMDP, where the perception of the agents is uncertain and they must work together to achieve their goal. At each discrete time step $t = 0, 1, \dots, H - 1$ every agent takes an action and the combination of these actions influences the world, making it transform to the next state. The goal of the agents is to try to come up with a plan that maximizes their expected long-term rewards. The goal or task need to be completed with cooperation, because no single agent is competent enough. Therefore, the plan computed should be decentralized.

In this section, we briefly review some background of the decentralized POMDPs model, ICE, approximate Q-value functions, MCTS, and collaborative Bayesian games. We refer the readers to the previous work [2–4,21,22] about the detailed background.

3.1. Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) Model

Dec-POMDP generalizes POMDP to a multi-agent version where rewards are common and based on joint actions, but observations are individualistic [23,24]. The detail definition is as follows.

Definition 1. A decentralized partially observable Markov decision process is defined as a 9-tuple $\langle I, S, A, T, \mathcal{O}, O, R, b^0, h \rangle$, where

- $I = \{1, \dots, n\}$ is the finite set of agents;
- $S = \{s_0, \dots, s_{|S|-1}\}$ is the finite set of states, where s_0 is the initial state;
- $A = \times_{i=1}^n A_i$ is the set of joint actions where $\vec{a} = \langle a_1, a_2, \dots, a_n \rangle$, and A_i is the finite set of individual actions of agent i . The component a_i in the joint action $\vec{a}$ belongs to A_i ;
- T is the transition function where $P(s' | s, \vec{a})$ denotes the probability of transiting to the new state s' when taking the joint action $\vec{a}$ in state s ;
- $\mathcal{O} = \times_{i=1}^n \mathcal{O}_i$ is the set of joint observations where $\vec{o} = \langle o_1, \dots, o_n \rangle \in \mathcal{O}$, and $\mathcal{O}_i$ is the finite set of individual observations of agent i . The component o_i in the joint observation $\vec{o}$ belongs to $\mathcal{O}_i$;
- O is the observation function where $P(\vec{o} | \vec{a}, s')$ denotes the probability of observing $\vec{o}$ when the system state transfer to s' taking the joint action $\vec{a}$;
- R is the reward function, where $R(s, \vec{a})$ denotes the immediate reward when taking the joint action $\vec{a}$ in state s ;
- b^0 is the initial state distribution;
- h is the finite horizon.

At every step t , the environment is in a particular state s , which emits a joint observation $\vec{o}^t$ according to the observation model. Each agent obtains its individual observation component o_i^t . Each agent selects its individual action a_i^t , forming a joint action $\vec{a}^t$, which leads to a state s' according to the transition model. Every agent receives an immediate reward r .

The action-observation history θ_i^t for agent i is the sequence of actions taken and observations received, which is denoted as a sequence $(o_i^0, a_i^0, o_i^1, \dots, a_i^{t-1}, o_i^t)$. Thus, the joint action-observation history $\vec{\theta}^t$ is $\langle \theta_1^t, \theta_2^t, \dots, \theta_n^t \rangle$. A policy for agent i is a mapping from history to actions $\pi_i : \theta_i \rightarrow A_i$,

and a joint policy is denoted as $\pi = \langle \pi_1, \pi_2, \dots, \pi_n \rangle$ which specifies a policy for each agent. The goal of solving Dec-POMDPs is to calculate an optimal joint policy to maximize the expected long-term accumulated reward, which is defined as

$$E\left(\sum_{i=1}^{h-1} R(s^i, \vec{a}^i) | \pi, b^0\right) \quad (1)$$

The value of an optimal joint policy π^* can be defined as follows:

$$V^t(\pi^*) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0) Q^*(\vec{\theta}^t, \pi^*(\vec{\theta}^t)) \quad (2)$$

where $\pi^*(\vec{\theta}^t)$ denotes the joint action that π^* specifies for $\vec{\theta}^t$, and

$$Q^*(\vec{\theta}^t, \vec{a}^t) = R(\vec{\theta}^t, \vec{a}^t) + \sum_{\vec{o}^{t+1} \in O} P(\vec{o}^{t+1} | \vec{\theta}^t, \vec{a}^t) Q^*(\vec{\theta}^{t+1}, \pi^*(\vec{\theta}^{t+1})) \quad (3)$$

is the optimal Q-value function.

The policy π_i of agent i specifies actions for all steps of Dec-POMDPs, and π_i can be represented as a sequence of decision rules $\pi_i = (\delta_i^0, \delta_i^1, \dots, \delta_i^{h-1})$. The decision rule δ_i^t for agent i at step t is a mapping from the length- t history to action $\delta_i^t: \theta_i^t \rightarrow A_i$. The partial joint policy φ^t is defined as a sequence of decision rules specified for steps $0, \dots, t-1$, which is denoted as $\varphi^t = (\delta^0, \delta^1, \dots, \delta^{t-1})$.

3.2. Generalized MAA* with Incremental Clustering and Expansion (GMAA*-ICE)

GMAA*-ICE (or ICE) [3] is a heuristically guided algorithm based on GMAA* framework [4], which generalizes the first multi-agent A* search MAA* [2]. It uses $\hat{V}(\varphi^t) = V^{0\dots t-1}(\varphi^t) + H^{t\dots h-1}(\varphi^t)$ to guide A* search over partially specified joint policies to construct a tree from top-down, where $V^{0\dots t-1}(\varphi^t)$ is the actual expected reward achieved over the first t steps and $H^{t\dots h-1}(\varphi^t)$ is a heuristic value for the remaining $h-t$ steps. ICE uses an admissible heuristic to guarantee that it can find an optimal policy.

The key contributions of ICE are lossless clustering and incremental expansion. The authors introduced a probabilistic equivalence criterion that can be used to decide whether two individual action-observation histories, $\theta_{i,a}, \theta_{i,b}$ for agent i , can be clustered. Especially in problems that have many histories satisfying probabilistic equivalence, ICE can significantly speed up planning. Incremental expansion defines a method of incrementally constructing a search tree by expanding the most promising child $\varphi^{t+1} = (\varphi^t, \beta^*)$ where β^* is the optimal solution of collaborative Bayesian games. The author defined a node priority which helps find the tight lower bound early by first expanding deeper nodes [2].

ICE [3] is our baseline algorithm, but our methods are different from it as we integrate Q-value function computing with policy solving, so CEMC calculates the Q-value functions as required, and does not have to store and maintain a large matrix of the Q-value function.

3.3. Approximate Q-Value Functions

As mentioned above, ICE uses $\hat{V}(\varphi^t) = V^{0\dots t-1}(\varphi^t) + H^{t\dots h-1}(\varphi^t)$ to guide the search and $H^{t\dots h-1}(\varphi^t)$ is a heuristically estimated value using approximate Q-value functions. There are three well-known approximate Q-value functions, Q_{MDP} , Q_{POMDP} and Q_{BG} , which are all approximations for the optimal Q-value function Q^* . Although the optimal policy can be extracted from Q^* , it is costly and impractical to compute.

Szer and Charpillet [25] used Q_{MDP} to solve Dec-POMDP that is approximated as an underlying MDP. This underlying MDP is defined by a single agent which takes joint actions but has the same

state set, transition function, and reward function as original Dec-POMDPs. It assumes that the state can be fully observed and the single agent gains more information than Dec-POMDPs do.

Q_{POMDP} [25] is another approximation for Q^* . In this case, Dec-POMDPs are treated as underlying POMDPs defined by a single agent which takes joint actions and receives joint observations but has the same state set, transition function, observation function, and reward function. It assumes that underlying POMDPs are settings of centralized control, which allow for better coordination. This means that the single agent knows the joint action-observation history while the agents in Dec-POMDPs can only gain local information such as individual actions and observations.

Q_{BG} is proposed by Oliehoek et al. [4], which assumes that the agents know the joint action-observation history $\theta^{\rightarrow t-1}$ and the joint action $a^{\rightarrow t-1}$ for the previous time step $t-1$. This assumption defines a Bayesian games (BGs) for each $(\theta^{\rightarrow t-1}, a)$ pair. The actions taken by agents can be solved by constructing BGs.

Q_{MDP} , Q_{POMDP} and Q_{BG} are all proved to be upper bounds of Q^* [4]. These approximate Q-value functions form a hierarchy of upper bounds to Q^* :

$$Q^* < Q_{\text{BG}} < Q_{\text{POMDP}} < Q_{\text{MDP}} \quad (4)$$

This means that all these approximate Q-value functions are admissible heuristic for Dec-POMDPs.

Apart from the Q-value functions mentioned above, Oliehoek et al. [18] proposed a factored Q-value function as a heuristic payoff function for factored Dec-POMDP. The precondition of a decomposition is that the Q-value function of every joint policy is factored. Each local component of this kind of Q-value function depends on a subset of state factors and the action-observation histories and actions of a subset of agents.

3.4. Monte Carlo Tree Search (MCTS)

MCTS builds a search tree incrementally and asymmetrically in a given domain by taking a random sample. Each node in the tree denotes a state and links to child nodes which represent actions taken. It works by repeating selection, expansion, simulation, and backpropagation on a loop until some predefined computational budget, usually runtime or memory, is reached [26].

Many extensions have been reviewed in the literature [22]. MCTS is used to find the optimal feature subset through the feature space in [27]. In this paper, we use plain UCT (upper confidence bounds for trees) [6] as the tree policy for computing the Q-value function. Plain UCT is MCTS with UCB1 (a version of upper confidence bound), the tree policy of which is:

$$\bar{x}_i + 2c\sqrt{\frac{2\ln n}{n_i}} \quad (5)$$

where $\bar{x}_i$ is the mean value; n is the number of times that the parent node has been visited; n_i is the number of times child i has been visited, and $c > 0$ is a constant.

The role of a tree policy is to select and create a node, which is carried out in the selection and expansion phase. A default policy is in the simulation phase. Its function is to play out the domain to estimate a state value. In a previous paper [28], four criteria for selecting actions are listed: (i) max child, (ii) robust child, (iii) max-robust child, and (iv) secure child. If the search terminates, one can choose one of the four selection mechanisms above for selecting an action to take. Our tree policy chooses the highest value node in the selection phase and only one node is expanded in the expansion phase. In the default policy (simulation phase), we replaced a random sample with simulation enhancement best-first policy. The advantage of randomly selecting actions is that it is simple, requires no domain knowledge, and can quickly simulate many times. However, it is not likely to be realistic compared

to rational players [22]. The enhancement we proposed can use domain knowledge to simulate realistically. The details will be introduced in Section 3.3.

3.5. Collaborative Bayesian Games

A Bayesian game (BG) [29] is an augmented normal form game in which a number of agents interact in a one-shot manner, and each of them holds some private information. This private information defines the type of the agent. A collaborative Bayesian game (CBG) is a BG in which agents receive identical payoffs. A CBG is defined as follows:

Definition 2. A collaborative Bayesian game (CBG) [3,4] $B(b^0, \varphi^t) = \langle I, A, \Theta, \Pr(\cdot), u \rangle$ modeling stage t of a Dec-POMDP, given initial state distribution b^0 and past joint policy φ^t , consists of:

- $I = \{1, 2, \dots, n\}$ is the finite set of agents;
- A is the set of joint actions;
- Θ is the set of joint types, each of which specifies a type for each agent $\theta = \langle \theta_1, \theta_2, \dots, \theta_n \rangle$;
- $\Pr(\cdot)$ is a probability distribution over joint types;
- u is a heuristic payoff function mapping joint types and joint actions to a real number: $u(\theta, a)$.

That is, given b^0 , for each φ^t , it is possible to construct a CBG $B(b^0, \varphi^t)$. Therefore, $B(b^0, \varphi^t)$ represents a decision-making problem for stage t , if φ^t was followed for the first t stages starting from b^0 . We simply write $B(\varphi^t)$ below.

In a CBG, each agent uses a Bayesian game policy β_i that maps individual types to actions $\beta_i(\theta_i) = a_i$. If β denotes the joint policy for the CBG, it corresponds to a joint decision rule: $\beta \leftrightarrow \delta^t$. Oliehoek et al. [4] model Dec-POMDP with series of CBGs, or in other words, a stage of a Dec-POMDP can be modeled with a CBG. Therefore, we can solve a Dec-POMDP problem stage by stage, which facilitates A*-like search algorithms.

4. Monte Carlo Q-Value Function

In this section, we propose a new Q-value function representation—Monte Carlo Q-value function Q_{MC} —and give proof of an upper bound of the optimal Q-value function Q^* . Q_{MC} combines the precision of tree search and the anytime character of sample-based Monte Carlo, and has more flexibility than other Q-value functions do. Then, we present two MCTS enhancements—heavy payout for simulation policy and adaptive sample—to speed up the computation of Q_{MC} .

4.1. Q_{MC}

In reinforcement learning [30], the Q-value function $Q(s, a)$ is defined as the expected return starting from state s , taking action a , and then following policy π :

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma Q^\pi(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma Q(s', a')]. \end{aligned} \quad (6)$$

Different from the former, the state in Q_{MC} is the joint history $\vec{\theta}_t$ at time step t . We define Q_{MC} as:

$$Q_{MC}(\vec{\theta}^t, \vec{a}) = R(\vec{\theta}^t, \vec{a}) + \sum_{\vec{o}^{\rightarrow t+1} \in O} P(\vec{o}^{\rightarrow t+1} | \vec{\theta}^t, \vec{a}) \max_{MC(\vec{o}^{\rightarrow t+1})} Q_{MC}(\vec{\theta}^{\rightarrow t+1}, MC(\vec{o}^{\rightarrow t+1})) \quad (7)$$

where MC is the Monte Carlo method, which returns the optimal action $MC(\vec{o}^{\rightarrow t+1})$ when observing $\vec{o}^{\rightarrow t+1}$. Because of the partially observability, the probability multiplied by Q_{MC} is a conditional probability of all observations at $\vec{\theta}_t$ and $\vec{a}$.

The solution of Q_{MC} can be illustrated in Figure 1. The black filled nodes are joint histories, and the solid links and dotted links are respectively joint actions taken and joint observations received by agents. Each joint history node contains two statistical values— $N(\vec{\theta})$, visiting count of node $\vec{\theta}$ and $N(\vec{\theta}, \vec{a})$, number of taking $\vec{a}$ at $\vec{\theta}$.

Figure 1. Tree of Monte Carlo Q-value function.

At the selection phase of Q_{MC} , UCB1 is used to choose the best action a^*

$$a^* = \underset{\vec{a} \in A}{\operatorname{argmax}} \left\{ R + \hat{R} + c \sqrt{\frac{2 \ln N(\vec{\theta})}{N(\vec{\theta}, \vec{a})}} \right\} \quad (8)$$

where R and $\hat{R}$ are respectively the immediate and future reward at .., and the term added to it is called a UCT bonus, which is used to balance exploitation and exploration. $R + \hat{R}$ is the mean value of reward. The c parameter is a constant ($c > 0$), which is used to tune how much influence the bonus has on the action selection. It is set bigger for problems with more branches and smaller for deeper ones. We did not consider adaptive setting of parameter c in this paper.

At the expansion phase of Q_{MC} , it expands only one node. At the simulation phase, Q_{MC} replaces the default random policy with heavy playout—an enhancement of Monte Carlo, which is presented in Section 3.3. At the backpropagation phase, Q_{MC} updates the values or statistics of nodes that are visited in this iteration with the result of the finished play.

Lemma 1. Q_{MC} is an upper bound of the optimal Q-value function Q^* .

Proof See appendix. $\square$

From the representation of Q_{MC} we can see that it treats Dec-POMDPs as models in which a single agent takes joint actions and receives joint observations, but has the same state set S , transition function T , observation function O and reward function R . The assumption makes more information available than the underlying Dec-POMDPs. Q_{MC} is an overestimation of the optimal Q-value function and makes A*-like search find the optimal policy. The pseudocode of computing Q_{MC} is shown in Algorithms 1 and 2.

Algorithm 1: $Q_{MC}(\vec{\theta}^t, \vec{a})$

Input: joint action-observation history $\vec{\theta}^t$ at t , joint action $\vec{a}$
Output: Monte-Carlo Q-value function $Q_{MC}(\vec{\theta}^t, \vec{a})$

```

1:  $K \leftarrow \text{adaptSet}(\text{num}A, t)$  //adaptive setting of sample
2:  $\text{imm}R \leftarrow \text{ComputeImm}R(\vec{a})$ 
3:  $t \leftarrow \vec{\theta}^t.\text{timeStep}$ 
4: for  $k < K$  do
5:    $\text{fut}R \leftarrow 0$ 
6:   for all  $\vec{o} \in O$  do
7:      $\vec{\theta}^{t+1} \leftarrow (\vec{\theta}^t, \vec{a}, \vec{o})$ 
8:      $\text{fut}R += \gamma \cdot P(\vec{\theta}^{t+1}) \cdot Q^*(\vec{\theta}^{t+1}, t+1)$  //compute future reward
9: return  $\text{imm}R + \text{fut}R$ 

```

Algorithm 2: $Q^*(\vec{\theta}, t)$

Input: joint action-observation history $\vec{\theta}$, time step t
Output: the optimal reward at $\vec{\theta}$

```

1: if  $\vec{\theta}$  has unexpanded actions then
2:   return  $\text{Rollout}(\vec{\theta}, t)$  //calling Rollout function
3: else
4:    $\text{best}A \leftarrow \arg \max_{\vec{a} \in A} (\text{imm}R + \text{fut}R + c \sqrt{\frac{2 \ln N(\vec{\theta})}{N(\vec{\theta}, \vec{a})}})$ 
5:    $\text{imm}R \leftarrow \text{ComputeImm}R(\text{best}A)$ 
6:   if  $(\neg \text{laststep})$  then
7:     for all  $\vec{o} \in O$  do
8:        $\vec{\theta}' \leftarrow (\vec{\theta}, \text{best}A, \vec{o})$ 
9:        $\text{fut}R += \gamma \cdot P(\vec{\theta}') \cdot Q^*(\vec{\theta}', t+1)$ 
10:   $N(\vec{\theta}) \leftarrow N(\vec{\theta}) + 1, N(\vec{\theta}, \text{best}A) \leftarrow N(\vec{\theta}, \text{best}A) + 1$  //backpropagation
11:   $Q(\vec{\theta}, \text{best}A) += \frac{1}{N(\vec{\theta}, \text{best}A)} (\text{imm}R + \text{fut}R - Q(\vec{\theta}, \text{best}A))$ 
12: return  $Q(\vec{\theta}, \text{best}A)$ 

```

4.2. Adaptive Sample

Due to the anytime character of MCTS, sufficient sample times (K) can make Q_{MC} converge to Q^* , but it is costly to calculate a converged Q_{MC} . However, a small K leads to a loose Q-value function or even a wrong policy. Thus, in this article, we present a method for an adaptive sample, which is a function of joint actions and time steps. That is:

$$K = m \frac{|A|}{(t+1)^2} \quad (9)$$

where m is a constant, determined by the specific problem, $|A|$ is the number of joint actions and t is a time step.

K is in proportion to joint actions and inverse proportion to the square of time steps. This means that problems with more joint actions need larger K to make Q_{MC} converge. With the tree of Q_{MC} constructed from top-down, K shrinks gradually. We use this technique to balance the accuracy of convergence and computing time.

4.3. Best-First Simulation Policy

Simulation policy uses a default policy to play a game quickly. After that, MCTS propagates back the simulation results. The default policy is typically domain dependent. As there is no value like $Q(\vec{\theta}^t, \vec{a})$ for guiding the action selection, uniformly choosing available actions is a common scheme.

However, the heuristic knowledge of the problem can be involved into the default policy to strengthen simulation. Silver [31] discusses some factors that influence simulation performance. In this paper, we extend the rollout phase by incorporating domain knowledge into playouts, which are called heavy playouts [32]. If it is not the last time step, the algorithm recursively calls for the Rollout function. Then it returns the best immediate reward ($immR$) and backpropagates the value obtained and updates $N(\vec{\theta})$ and $N(\vec{\theta}, \vec{a})$. This Best-First Simulation Policy (BFSP) returns the promising node first to ensure that the heuristic value is admissible or a guaranteed overestimation. The detail is shown in Algorithm 3.

Especially we would like to note that the $selectBestA()$ function selects the best action combining an idea of reinforcement learning and the history heuristic. Reinforcement learning (RL) reinforces the actions that make success. The history heuristic assumes that an action good in one position may be good in another. Joint actions, which acquire the highest simulating value, will be given more chances in the later simulation. Concretely, we keep an average accumulated value $v(\vec{a})$ of each joint action in a playout. Apart from $v(\vec{a})$, a separate visit of each action n_i and total visits of all actions, n in the simulation phase is also stored. An action is selected by a softmax function with inputs $z = v(\vec{a}) \cdot \frac{n_i}{n}$. This implies that the larger z will correspond to a larger probability. If it is at an initial state or the chance of nodes are equal, a random legal action will be selected. For example, if a problem has nine joint actions, in the start of a simulation phase, an action is selected through this softmax function. After the simulation ends, its accumulated value to the terminal is recorded and the visit number increases by one. In another simulation phase, the statistics are reused, which we call the history heuristic.

Algorithm 3: Rollout($\vec{\theta}, t$)

Input: joint action-observation history $\vec{\theta}$, time step t

Output: the optimal reward of rollout

```

1: if (isLastTimeStep) then
2:    $immRbest \leftarrow -\infty$ 
3:   for all  $\vec{a} \in A$  do
4:      $immR \leftarrow \text{ComputeImmR}(\vec{a})$ 
5:      $N(\vec{\theta}, \vec{a}) \leftarrow N(\vec{\theta}, \vec{a}) + 1$ 
6:      $immRbest \leftarrow \max(immR, immRbest)$ 
7:    $N(\vec{\theta}) \leftarrow N(\vec{\theta}) + 1$ 
8:   return  $immRbest$ 
9: else
10:  for all  $\vec{a} \in A$  do
11:     $bestA \leftarrow selectBestA(\vec{a})$ 
12:   $immR \leftarrow \text{ComputeImmR}(bestA)$ 
13:  for all  $\vec{o} \in O$  do
14:     $\vec{\theta}' \leftarrow (\vec{\theta}, bestA, \vec{o})$ 
15:     $futR += \gamma \cdot P(\vec{\theta}') \cdot \text{Rollout}(\vec{\theta}', t + 1)$ 
16:   $N(\vec{\theta}) \leftarrow N(\vec{\theta}) + 1, N(\vec{\theta}, bestA) \leftarrow N(\vec{\theta}, bestA) + 1$ 
17:  return  $immR + futR$ 

```

5. Multi Agent Planning Using Q_{MC}

In this section, we present our offline planning algorithm—CEMC—which is based on an ICE framework. Then, we introduce a new policy pruning strategy to speed up planning. CEMC integrates Q-value function computing and plan solving, and calculates the Q-value functions as required, which avoids the need to compute and store all Q-value functions. Thus, CEMC does not calculate and maintain a large matrix for Q-value, which achieves improvement in runtime and memory.

5.1. Offline Planning Algorithm—CEMC

Offline algorithms generate a complete policy before execution begins. The policy specifies what actions to take in any possible runtime situation. Before presenting CEMC, we first introduce two successful techniques used in ICE, which is also adopted by CEMC.

The first one is lossless clustering. Two individual action-observation histories $\theta_{i,a}, \theta_{i,b}$ for agent i can be losslessly clustered when they satisfy probabilistic equivalence criterion:

$$\forall \vec{\theta}_{\neq i}, \forall s, P(s, \vec{\theta}_{\neq i} | \theta_{i,a}) = P(s, \vec{\theta}_{\neq i} | \theta_{i,b}) \quad (10)$$

where $\vec{\theta}_{\neq i}$ is the joint action-observation histories of all agents except agent i . This means that the probability distribution over the other agent's joint action-observation histories are identical for both $\theta_{i,a}, \theta_{i,b}$.

The second one is incremental expansion which defines a method of incrementally constructing a search tree by expanding the most promising child $\varphi^{t+1} = (\varphi^t, \beta^*)$ where β^* is the optimal solution of CBGs. A child is selected by node priority. Node q in the search tree is a 3-tuple $\langle \varphi^t, \hat{v}, h \rangle$. Node priority $q < q'$ if:

$$\begin{cases} \hat{v} < \hat{v}' & , \quad \text{if } \hat{v} \neq \hat{v}' \\ h < h' & , \quad \text{otherwise if } h \neq h' \end{cases} \quad (11)$$

That is, if the heuristic values are equal, the greater depth will be chosen first. This helps find the tight lower bound early by first expanding deeper nodes and converge quickly.

The CEMC algorithm (Algorithm 4) starts planning with an unspecified partial joint policy φ^0 in an initialized policy pool. It greedily selects the best partial joint policy in the pool according to the node priority(Select(L)). Then, it proceeds by constructing a CBG using lossless clustering(ClusterBG()). After that, the CBG is solved, which returns a 2-tuple $\langle \beta^t, \hat{V}(\beta^t) \rangle$ where β^t is a Bayesian game policy, and $\hat{V}(\beta^t)$ is given by the function Solver():

$$\hat{V}(\beta^t) = \sum_{\vec{\theta}^t} P(\vec{\theta}^t | b^0, \varphi^t) Q_{MC}(\vec{\theta}^t, \beta(\vec{\theta}^t)) \quad (12)$$

where $P(\vec{\theta}^t | b^0, \varphi^t)$ is the probability distribution over joint histories and $Q_{MC}(\vec{\theta}^t, \beta(\vec{\theta}^t))$ is the Q-value function computed by Algorithm 1.

Algorithm 4: CEMC

Input: a Dec-POMDP, an empty policy pool L

Output: an optimal joint policy π^*

1: Initialize()

2: **while** $\neg \text{empty}(L)$ **do**

3: $q \leftarrow \text{Select}(L)$

4: $B(\varphi^t) \leftarrow \text{ClusterBG}()$ //clustering

5: $\langle \beta^t, \hat{V}(\beta^t) \rangle \leftarrow B(\varphi^t).\text{Solver}()$ //computing Q_{MC} as required

6: $N_{IE} \leftarrow \text{IncExp}()$ //incrementally expand nodes

7: $v^* = \text{BestValue}(Q_{MC})$ //return the best Q_{MC} at step t

8: **if** $\text{depth}(q) = h - 1$ **then**

9: $\langle \pi, v \rangle \leftarrow \text{ComputePolicyAndValue}(N_{IE})$

10: **if** $v > \underline{v}$ **then**

11: $\pi^* \leftarrow \pi, \underline{v} \leftarrow v$

12: $L.\text{Prune}(\underline{v})$ //pruning the policy pool

13: **else**

14: $L.\text{Insert}(N_{IE})$

15: $L.\text{Prune}(v^*)$ //pruning the policy pool

16: **return** π^*

The set of child nodes N_{IE} is expanded using the incremental expansion technique and only the promising child node $\beta^{t,*}$ will be put into $N_{IE}(\text{IncExp}())$. The heuristic value $\hat{v}$ of the node is given by the function `ComputePolicyAndValue()`:

$$\hat{V}(\varphi^{t+1}) = V^{0\dots t-1}(\varphi^t) + \hat{V}(\beta^t) \quad (13)$$

where $V^{0\dots t-1}(\varphi^t)$ is the actual expected reward of φ^t and $\hat{V}(\beta^t)$ is given by Equation (12).

We use two pruning strategies during planning. If time step $t < h - 1$, nodes with values less than the best Q-value function Q_{MC}^* will be pruned in advance. If $t = h - 1$, CEMC computes fully expanded policies. Nodes with values less than the best lower bound $\underline{v}$ can be pruned. CEMC ends when the policy pool becomes empty or the highest value in the policy pool is less than that of the best full policy.

The main idea of CEMC is that it integrates the Q-value function computing with plan solving, which benefits from the anytime character of Q_{MC} . The original ICE computes and stores the Q-value functions in a $N_{jaoh} \times N_{ja}$ matrix, where N_{jaoh} is the number of joint action-observation histories and N_{ja} is the number of joint actions. It is costly to calculate complete Q-value functions in advance. For example, for the benchmark problem, Mars Rovers (Appendix Table A1), there are about 2.82×10^{14} joint action-observation histories and 36 joint actions. It needs to compute about 1.02×10^{16} $(\vec{\theta}, \vec{a})$ value pairs. The algorithm CEMC calculates the Q-value functions only when required, without computing and storing all Q-value functions.

5.2. Policy Pruning Strategy

Pruning is split into hard pruning, which removes branches of the tree permanently, and soft pruning, which temporarily removes branches, but then adds them back into the search at some later point [33]. In this paper, we use hard pruning as our pruning strategy.

The ICE algorithm maintains a policy pool. A node in the pool is denoted as a tuple $\langle \varphi, \hat{v}, h \rangle$, where φ is a partial policy, $\hat{v}$ is its value, and h is its depth. We know the value of a node is an actual value plus an estimated value: $\varphi = V^{0\dots t-1}(\varphi^t) + \hat{V}(\beta^t)$. The estimated value $\hat{V}(\beta^t)$ is an overestimation of the actual value from this node to a terminal. Actually, the value of a node φ is the value of a path or a policy across this node. Therefore, with the search descending deep down to the terminal or the finite horizon, the value of a node is also descending (note that the actual value becomes larger and larger, and the estimated value becomes close to the actual value). When the search reaches the terminal, the value of the terminal node can be seen as the best lower bound ever found. If the value of a node in the policy pool is smaller than the best lower bound, it would never be the optimal. Therefore, it uses the best lower bound $\underline{v}$ to prune the policy pool. However, for some problems in the literature, the policy pool shrinks slowly, which wastes a lot of time in computing useless nodes.

We use an example to illustrate it. Consider, for example, Figure 2 is a search tree. The policy pool is a priority list, which has several nodes, such as $[\langle \varphi^2, 10.125, 2 \rangle, \langle \varphi^1, 10.125, 1 \rangle, \langle \varphi^2, 9.721, 2 \rangle, \langle \varphi^{2''}, 6.182, 2 \rangle, \dots]$. The superscript of φ is its time step or its depth. At each loop the ICE algorithm chooses the first node in the pool and expands it. If the algorithm reaches the last step, it returns the fully expanded node $(\langle \varphi^3, 5.363, 3 \rangle)$ as the best lower bound to prune the policy pool. However, node $\langle \varphi^{2''}, 6.182, 2 \rangle > \langle \varphi^3, 5.363, 3 \rangle$ according to the node priority, and it will not be pruned. Assuming that the child of the node $\varphi^{2''}$ is $\langle \varphi^{3'}, 4.182, 3 \rangle$, this fully expanded policy is obviously not the optimal one and should have been pruned in advance, but node $\langle \varphi^{2''}, 6.182, 2 \rangle$ will be still expanded in these iterations. Not pruning in advance makes the search algorithm go the wrong direction and this is to be solved in our policy pruning strategy.

Thus, these useless nodes like $\langle \varphi^{2''}, 6.182, 2 \rangle$ should be removed as early as possible. We improve the pruning function using the max Q-value function at time step t $Q_{MC}^{t,*}$, combined with the best lower bound $\underline{v}$, that is:

$$\begin{cases} \{q \in L | \hat{v} > v^*\} & , \quad \text{if } t < h - 1 \\ \{q \in L | \hat{v} > \underline{v}\} & , \quad \text{if } t = h - 1 \end{cases} \quad (14)$$

where t is the deepest level in the policy pool, v^* is computed through $Q_{MC}^{t,*}$:

$$v^* = V^{0 \dots t-1}(\varphi^t) + \max_a Q_{MC}(\vec{\theta}^t, a) \quad (15)$$

The Equation (14) illustrates that if $t < h - 1$, all nodes with values less than v^* can be pruned and if CEMC reaches the last step, any node with values less than $\underline{v}$ can be pruned. Note that pruning using v^* is done during the planning, it does not have to wait for full expansion. This technique can reduce the number of nodes in the policy pool and make the algorithm converge quickly, which is tested experimentally in Section 5.

Figure 2. Incremental clustering and expansion (ICE) search tree. Due to inappropriate policy pruning, some nodes like $\varphi^{2''}$ will be left in the policy pool and they may be still expanded in later iterations. The dashed circle is an unvisited node.

6. Experiments

This section experimentally demonstrates and validates the effectiveness of all techniques proposed: the new Q-value function representation Q_{MC} , the adaptive sample, best-first simulation policy (BFSP), policy pruning strategy, and the core algorithm CEMC. We give several benchmark problems in the literature, followed by experimental data. Finally, we analyze and discuss the results.

6.1. Experiments Setup

We give some benchmark problems, including Dec-Tiger, Grid-Small, Recycling Robots, Box Pushing, Mars Rovers, Broadcast Channel. All these problems have two agents, with different numbers of states, actions and observations, as shown in Table A1 (see Appendix A), which lists the number of joint policies for different horizons. These domains can be downloaded at MASPlan (<http://masplan.org/>).

The problem, Broadcast Channel, has two agents, each of them must choose whether or not to send a message in a Broadcast Channel. If both agents send, a collision occurs. Therefore, they must cooperate to send messages. The problem, Recycling Robots, also has two agents in the environment, in which both of them move around using their motors, pick up cans, and place them in a specified place using their mechanical arms. They must complete the task cooperatively. The benchmark Mars Rovers involve two rovers performing certain scientific experiments by choosing to check, drill, or sample. They can move in four directions. Some locations need only one agent and some locations require their cooperation.

Experiments were run on Ubuntu 16.04 with Intel Core i5 CPU and 6 GB memory. CEMC was implemented in C++ based on the MADP Toolbox [34]. Every average result was run over 100 times. We compared CEMC with the state-of-the-art exact heuristic algorithm ICE with a compact Q-value function implementation Q_{BG} . Five statistics are taken into consideration: time of computing Q-value function t_Q , time of calculating the optimal joint policy t_p , overall time t_o , value of optimal policy V^* , and value of random policy V_r . We would like to note that ICE may not get the same performance contrasted with the original paper due to the computing resources, but all results including ICE and CEMC were tested in the same environment, so the time and scalability results are directly comparable.

6.2. Experiment Results

6.2.1. Comparing CEMC and ICE

Table 1 shows performance results for ICE and CEMC. “-” indicates unknown values, caused by getting the wrong results or a time limit violation (less than 3600 s is valid). “*” indicates out of memory. Table 1 clearly shows that CEMC outperforms the state-of-the-art heuristic algorithm with Q_{BG} . Especially in solving scalable problems such as Grid-Small, Box Pushing, and Mars Rovers, CEMC shows significant improvement without computing and storing all Q-values. CEMC allows for getting the needed Q-value to guide A* search without pre-computation of all Q-value function.

Table 1. Comparable results of some benchmarks.

(a)						
h	Algo	$t_Q(s)$	$t_p(s)$	$t_o(s)$	V^*	V_r
Dec-Tiger						
2	ICE	<0.01	<0.01	0.27	−4.00	−93.05
	CEMC	0	<0.01	0.27	−4.00	−93.05
3	ICE	0.06	0.02	0.43	5.19	−140.81
	CEMC	0	0.06	0.28	5.19	−140.81
4	ICE	-	-	-	-	-
	CEMC	0	65.08	65.36	4.80	−186.67

“-” indicates unknown values, caused by getting the wrong results or a time limit violation (less than 3600 s is valid).

(b)						
Recycling Robots						
2	ICE	<0.01	<0.01	0.27	6.8	2.6
	CEMC	0	<0.01	0.27	6.8	2.6
3	ICE	0.05	0.02	0.40	9.76	3.17
	CEMC	0	0.06	0.33	9.76	3.17
4	ICE	1.55	0.01	1.85	11.73	3.24
	CEMC	0	0.49	0.80	11.73	3.24
5	ICE	-	-	-	-	-
	CEMC	0	0.83	1.21	13.76	3.78
10	ICE	*	*	*	*	*
	CEMC	0	1.07	1.51	21.2	-
20	ICE	*	*	*	*	*
	CEMC	0	2.53	2.92	28.19	-

“*” indicates out of memory.

(c)						
GridSmall						
2	ICE	0.01	0.01	0.95	0.86	0.20
	CEMC	0	0.04	0.95	0.86	0.20
3	ICE	3.88	0.03	4.84	1.37	0.38
	CEMC	0	0.91	1.81	1.37	0.38
4	ICE	158.44	0.03	159.23	1.88	0.56
	CEMC	0	11.76	12.66	1.88	0.56
5	ICE	*	*	*	*	*
	CEMC	0	50.23	51.33	2.53	-

Table 1. Cont.

(d)						
Box Pushing						
2	ICE	30.73	0.01	32.39	17.60	−0.01
	CEMC	0	0.54	1.81	17.60	−0.01
3	ICE	492.58	21.76	540.00	66.08	−0.61
	CEMC	0	0.82	2.36	66.08	−0.61
4	ICE	2974.12	371.83	3253.22	98.59	−
	CEMC	0	3.24	5.98	98.59	−
5	ICE	*	*	*	*	*
	CEMC	0	21.49	30.79	107.72	−
(e)						
Mars Rovers						
2	ICE	1171.76	0.17	1179.05	5.80	−3.17
	CEMC	0	3.81	10.93	5.80	−3.17
3	ICE	−	−	−	−	−
	CEMC	0	7.50	14.74	9.38	−
4	ICE	−	−	−	−	−
	CEMC	0	102.74	122.97	10.18	−
(f)						
Broadcast Channel						
2	ICE	<0.01	<0.01	0.28	2	0.85
	CEMC	0	<0.01	0.27	2	0.85
3	ICE	<0.01	0.01	0.28	2.99	1.13
	CEMC	0	0.03	0.29	2.99	1.13
4	ICE	0.13	<0.01	0.42	3.89	1.47
	CEMC	0	0.21	0.30	3.89	1.47
5	ICE	2.77	0.01	3.09	4.79	1.76
	CEMC	0	0.23	0.32	4.79	1.76
10	ICE	*	*	*	*	*
	CEMC	0	0.38	0.46	9.29	−
20	ICE	*	*	*	*	*
	CEMC	0	2.06	2.63	18.31	−
30	ICE	*	*	*	*	*
	CEMC	0	219.22	220	27.42	−

From Table 1 we can see that in some problems like Grid-Small, Box Pushing, and Mars Rovers, CEMC is significantly better than ICE, while in some problems like Dec-Tiger, Recycling Robots, and Broadcast Channel the difference between both may not be as obvious as the three previous ones. Actually, the number of joint decision rules for time step t in the search tree is

$$O(|A|^n(|O|^t)) \quad (16)$$

which means the complexity is exponential with the number of actions and observations, and doubly exponential with the time step. The search tree contains the action-observation histories, the actions taken, and the observations received. Therefore, the number of branches is related to the action and observation space. Although the domain Mars Rovers has a larger state space, it has little influence on the scale of the search tree. What it influences is the efficiency of state transition and the computing time of the probability like $P(o|a, s')$.

6.2.2. Analysis of Policy Pruning Strategy

To illustrate the effectiveness of the extended policy pruning strategy, we list the max number of policies in the policy pool for different horizons. Figure 3 shows a comparison of the original ICE and CEMC. Note that the original ICE maintains a larger policy pool than CEMC. For problems with approximatively $\hat{V}(\varphi^t)$, ICE expands nodes back and forth to construct a tree carefully. This will

significantly increase the search time. We use Equation (14) to prune the policy pool, and the promising nodes are left.

Figure 3. Max number of policies in the policy pool for different horizons. This indicator illustrates that ICE maintains a larger policy pool than CEMC, which may make the former expand useless nodes.

6.2.3. Convergence of Q_{MC}

More sample times can make the approximate Q-value function Q_{MC} converge and be close to Q^* , but it needs more runtime. To illustrate the convergence of Q_{MC} , we gathered the statistics of the max Q_{MC} of Dec-Tiger with the sample time K increasing, as shown in Figure 4. The max values of Q_{MDP} , Q_{POMDP} and Q_{BG} for Dec-Tiger with horizon $h = 3$ and $h = 4$ are also listed. We can see from the figure that the max Q_{MC} declines and converges as K increases. Note that Q_{MC} is as compact as Q_{POMDP} with horizon $h = 3$. In horizon $h = 4$, Q_{MC} is more compact than Q_{POMDP} but looser than Q_{BG} . The goal of Figure 4 is to show that Q_{MC} is an upper bound of Q^* and we can use a less compact Q-value function to reduce time of computing, but it still can guide A* to search the optimal policy.

Figure 4. Convergence of Q_{MC} as the sample time K increases. Q_{MC} converges to Q_{POMDP} , which means Q_{MC} is an upper bound for Q^* , but is less compact than Q_{BG} .

6.3. Analysis and Discussion

CEMC outperforms the state-of-the-art exact heuristic algorithm ICE with compact Q-value function Q_{BG} in terms of runtime and memory. There are three reasons for CEMC's performance.

Firstly, we propose a new Q-value function representation Q_{MC} , which is an upper bound for the optimal Q-value function Q^* . Q_{MC} is an anytime Q-value function, which makes it possible to compute as required. Adaptive sample and heavy rollout speed up the computation of Q_{MC} .

Secondly, CEMC calculates the Q-value function as required and does not maintain a large matrix to store all $Q(\vec{\theta}^t, \vec{a})$. For problems with a large joint action-observation space and horizon, maintaining the matrix will run out of memory. In addition, it can be seen from problems, Box Pushing and Mars Rovers, in Table 1 that it is costly to calculate the Q-value function (time of computing Q-value for ICE is more than 1100 s). If not every $(\vec{\theta}^t, \vec{a})$ pair is computed, it will save much time. A tighter upper bound can speed up planning, as it allows the algorithm to prune the policy pool more [4], but a tighter value function increases time of computing Q-value function, which accordingly increases the overall time (total time for computing Q-value and planning).

Thirdly, an extended policy pruning strategy is used in CEMC. The pruning strategy uses the best lower bound ever found $\underline{v}$ and the best accumulated Q-value function at time step t $Q_{MC}^{t,*}$ to prune the policy pool. This technique avoids expanding those nodes that will not be the optimal ones in any case.

7. Conclusions

In this paper, we combine the advantages of the heuristic algorithm and MCTS, and propose a new Q-value function representation and a novel offline algorithm CEMC, generalizing the prior method. In particular, we demonstrate two important results: (i) Computation of the Q-value function as required is an important reason for the CEMC's performance. This permits us no longer to maintain a large matrix to store all Q-value function pairs, which saves much memory; (ii) There is no need to compute Q-value functions as compact as possible. Experimental results demonstrate that it is costly to compute compact Q-value functions. Other techniques such as an adaptive sample, heavy rollout, and policy pruning strategy are introduced to speed up the algorithm. In the future work, we plan to analyze the relationship between the loose extent and sample times in order to find the smallest K to optimally solving Dec-POMDPs.

Author Contributions: J.Z. proposed the method and wrote the main part of this paper; Y.P. designed the experiment of this paper; R.W. and Y.F. checked the grammar; H.Y. supported this work financially and checked the paper.

Funding: This research was funded by the research foundation of Space Engineering University, grant number: zx10356.

Acknowledgments: The authors acknowledge the financial support received from the research foundation of Space Engineering University, grant number: zx10356, for their support and encouragement in carrying out this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Table A1. Domain parameters and number of π for different h .

Problem	Domain Parameters				num. of π for h		
	n	S	A	O	$h = 2$	$h = 5$	$h = 10$
Dec-Tiger	2	2	3	2	6561	3.43×10^{30}	1.39×10^{977}
Recycling Robots	2	4	3	2	6561	3.43×10^{30}	1.39×10^{977}
Grid-Small	2	16	5	2	390625	5.42×10^{44}	3.09×10^{1431}
Box Pushing	2	100	4	5	3.34×10^7	5.23×10^{940}	$1.25 \times 10^{2,939,746}$
Mars Rovers	2	256	6	8	1.69×10^{14}	1.88×10^{7285}	$2.57 \times 10^{238,723,869}$
Broadcast Channel	2	4	2	2	256	1.84×10^{19}	3.23×10^{616}

Lemma 1. Proofs

We use a Dec-POMDP with k steps delayed communication model to prove Lemma 1. This model can be denoted as an augmented MDP, which is a reformulation of the prior work [35,36] and extended to the Dec-POMDP setting [4].

The augmented MDP is defined as $\hat{M} = \langle \hat{S}, \hat{A}, \hat{T}, \hat{R} \rangle$, where an augmented state $\hat{s} \in \hat{S}$ is composed of a joint action-observation history $\vec{\theta}^t$ and a joint policy tree q^t :

$$\hat{s} = \begin{cases} \langle \vec{\theta}^t, q^t \rangle & , \quad 0 \leq t \leq h - k - 1 \\ \langle \vec{\theta}^t, q^{\tau=h-t} \rangle & , \quad h - k \leq t \leq h - 1 \end{cases} \quad (A1)$$

The contained q^t is a joint depth- k policy tree $q^t = \langle q_1^t, \dots, q_n^t \rangle$ starting from stage t and $q^{\tau=h-t}$ is a τ -steps-to-go sub-tree policy. For example, if $k = 2$, q^t represents a joint depth-2 policy tree for $0 \leq t \leq h - 3$. For $h - 2 \leq t \leq h - 1$, state $\hat{s}$ contains $q^{\tau=2}$ and $q^{\tau=1}$.

An augmented action $\hat{a} \in \hat{A}$ is a joint policy $\hat{a} = \langle \delta_1^{t+k}, \dots, \delta_n^{t+k} \rangle$ mapping length- k observation histories to joint actions at stage $t + k$. The transition model $\hat{T}$ is a probability $P(\hat{s}^{t+1} | \hat{s}^t, \hat{a}^t) = P(\langle \vec{\theta}^{\tau=t+1}, q^{\tau=t+1} \rangle | \langle \vec{\theta}^t, q^t \rangle, \delta^{t+k}) = P(o^{t+1} | \vec{\theta}^t, q^t(\vec{o}_{\emptyset}))$ with an empty observation $\vec{o}_{\emptyset}$ at stage t , where $\vec{\theta}^{\tau=t+1} = (\vec{\theta}^t, a^t, o^{t+1})$, $q^t(\vec{o}_{\emptyset})$ is an initial joint action at stage t and $q^{t+1} = \langle q^t \circ \delta^{t+k} \rangle$ meaning appending a policy δ^{t+k} to q^t . The reward model $\hat{R}$ is defined as $\hat{R}(\hat{s}) = \hat{R}(\langle \vec{\theta}^t, q^t \rangle) = R(\vec{\theta}^t, q^t(\vec{o}_{\emptyset}))$.

Therefore, the optimal Q-value function Q_k for a k -steps delayed communication model for $0 \leq t \leq h - k - 1$ is as follows.

$$Q_k(\hat{s}, \delta^{t+k}) = R(\vec{\theta}^t, q^t(\vec{o}_{\emptyset})) + \sum_{o^{t+1} \in O} P(o^{t+1} | \vec{\theta}^t, q^t(\vec{o}_{\emptyset})) \max_{\delta^{t+k}} Q_k(\hat{s}^{t+1}, \delta^{t+k}) \quad (A2)$$

For $h - k \leq t \leq h - 1$, there are $\tau = h - t$ stages to go and the Q-value function is the same as the original one.

If $k = 0$, q^t is a depth-0 policy and $\hat{s}$ reduces to $\langle \vec{\theta}^t, \emptyset \rangle$ or $\vec{\theta}^t, \delta^{t+k}$ reduces to a^t . Equation (18) reduces to $Q_0(\vec{\theta}^t, a^t) = R(\vec{\theta}^t, a^t) + \sum_{o^{t+1} \in O} P(o^{t+1} | \vec{\theta}^t, a^t) \max_{a^{t+1}} Q_0(\vec{\theta}^{\tau=t+1}, a^{t+1})$. Contrasting with Equation (7), we can get that Q_{MC} corresponds to the Q-value function Q_0 of a 0-delayed communication system. According to the theory of [4], a Dec-POMDP is identical to an h -steps delayed communication system. The optimal Q-value function Q_k of a Dec-POMDP with k -steps delayed communication is an upper bound to Q_{k+1} , that of a $k + 1$ -steps delayed communication system. This means that Q_{MC} , the Q-value function of a 0-delayed communication system, is an upper

bound of the optimal Q-value function Q^* of a Dec-POMDP (h -steps delayed communication system), that is:

$$Q^* < Q_{MC}$$

References

- Bernstein, D.S.; Givan, R.; Immerman, N.; Zilberstein, S. The complexity of decentralized control of Markov decision processes. *Math. Oper. Res.* **2002**, *27*, 819–840. [\[CrossRef\]](#)
- Szer, D.; Charpillet, F.; Zilberstein, S. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh, UK, 4 June 2005; pp. 576–583.
- Oliehoek, F.A.; Spaan, M.T.; Amato, C.; Whiteson, S. Incremental clustering and expansion for faster optimal planning in Dec-POMDPs. *J. Artif. Intell. Res.* **2013**, *46*, 449–509. [\[CrossRef\]](#)
- Oliehoek, F.A.; Spaan, M.T.; Vlassis, N. Optimal and approximate Q-value functions for decentralized POMDPs. *J. Artif. Intell. Res.* **2008**, *32*, 289–353. [\[CrossRef\]](#)
- Wu, F.; Zilberstein, S.; Chen, X. Online planning for ad hoc autonomous agent teams. In Proceedings of the International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, 28 June 2011; pp. 439–445.
- Kocsis, L.; Szepesvári, C. Bandit based Monte-Carlo planning. In Proceedings of the 17th European Conference on Machine Learning, Berlin, Germany, 18 September 2006; pp. 282–293.
- Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of Go without human knowledge. *Nature* **2017**, *550*, 354–359. [\[CrossRef\]](#) [\[PubMed\]](#)
- Kocsis, L.; Szepesvári, C.; Willemson, J. *Improved Monte-Carlo Search*; University of Tartu: Tartu, Estonia, 2006.
- Amato, C.; Dibangoye, J.S.; Zilberstein, S. Incremental policy generation for finite-horizon DEC-POMDPs. In Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling, Thessaloniki, Greece, 16 October 2009; pp. 569–576.
- Lam, R.; Willcox, K.; Wolpert, D.H. Bayesian Optimization with a Finite Budget: An Approximate Dynamic Programming Approach. In *Advances in Neural Information Processing Systems*; Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Barcelona, Spain, 2016; pp. 883–891.
- Rahmanian, H.; Warmuth, M.K. Online Dynamic Programming. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Long Beach/Los Angeles, CA, USA, 2017; pp. 2827–2837.
- Boularias, A.; Chaib-Draa, B. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, Sydney, Australia, 14 September 2008; pp. 20–27.
- Oliehoek, F.A.; Whiteson, S.; Spaan, M.T.J. Lossless clustering of histories in decentralized POMDPs. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 10 May 2009; pp. 577–584.
- Spaan, M.T.J.; Oliehoek, F.A.; Amato, C. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In Proceedings of the International Joint Conference on Artificial Intelligence, Barcelona, Spain, 28 June 2011; pp. 2027–2032.
- Lanctot, M.; Winands, M.H.M.; Pepels, T.; Sturtevant, N.R. Monte Carlo Tree Search with heuristic evaluations using implicit minimax backups. In Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, Dortmund, Germany, 26 August 2014; pp. 1–8.
- Aras, R.; Dutech, A. An investigation into mathematical programming for finite horizon decentralized POMDPs. *J. Artif. Intell. Res.* **2010**, *37*, 329–396. [\[CrossRef\]](#)
- Liu, M.; Amato, C.; Liao, X.; Carin, L.; How, J.P. Stick-Breaking Policy Learning in Dec-POMDPs. In Proceedings of the 24th International Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015; pp. 2011–2017.
- Oliehoek, F.; Whiteson, S.; Spaan, M.T.J. Approximate Solutions for Factored Dec-POMDPs with Many Agents. In Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), St. Paul, MN, USA, 6–10 May 2013; Volume 1, pp. 563–570.

19. Silver, D.; Veness, J. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23*; Lafferty, J.D., Williams, C.K.I., Shawe-Taylor, J., Zemel, R.S., Culotta, A., Eds.; Curran Associates, Inc.: Barcelona, Spain, 2010; pp. 2164–2172.
20. Amato, C.; Oliehoek, F.A. Scalable Planning and Learning for Multiagent POMDPs. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.
21. Oliehoek, F.A.; Amato, C. *A Concise Introduction to Decentralized POMDPs*; Springer: Berlin, Germany, 2016.
22. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. Ai Games* **2012**, *4*, 1–43. [\[CrossRef\]](#)
23. Amato, C.; Chowdhary, G.; Geramifard, A.; Üre, N.K. Decentralized control of partially observable Markov decision processes. In Proceedings of the IEEE Conference on Decision and Control, Florence, Italy, 10–13 December 2013; pp. 2398–2405.
24. Omidshafiei, S.; Aghamohammadi, A.; Amato, C.; How, J.P. Decentralized Control of Partially Observable Markov Decision Processes using Belief Space Macro-actions. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 5962–5969.
25. Szer, D.; Charpillet, F. An Optimal Best-First Search Algorithm for Solving Infinite Horizon Dec-POMDPs. In *Machine Learning: ECML 2005*; Springer: Berlin/Heidelberg, Germany; Porto, Portugal, 2005; Volume 3720, pp. 389–399.
26. Chaslot, G.; Bakkes, S.; Szita, I.; Spronck, P. Monte-Carlo Tree Search: A New Framework for Game AI. In Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, CA, USA, 22 October 2008; pp. 216–217.
27. Chaudhry, M.U.; Lee, J.-H. MOTiFS: Monte Carlo Tree Search Based Feature Selection. *Entropy* **2018**, *20*, 385. [\[CrossRef\]](#)
28. Schadd, F. *Monte-Carlo Search Techniques in the Modern Board Game Thurn and Taxis*; Maastricht University: Maastricht, The Netherlands, 2009.
29. Osborne, M.J.; Rubinstein, A. *A Course in Game Theory*; MIT Press: Cambridge, MA, USA, 1994.
30. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; The MIT Press: Cambridge, MA, USA, 2018.
31. Silver, D.; Tesauro, G. Monte-Carlo simulation balancing. In Proceedings of the International Conference on Machine Learning, Montreal, QC, Canada, 14 June 2009; pp. 945–952.
32. Drake, P.; Uurtamo, S. Move Ordering vs Heavy Playouts: Where Should Heuristics Be Applied in Monte Carlo Go. In Proceedings of the 3rd North American Game-On Conference, Shiga, Japan, 15 September 2007; pp. 171–175.
33. Sephton, N.; Cowling, P.I.; Powley, E.; Slaven, N.H. Heuristic move pruning in Monte Carlo Tree Search for the strategic card game Lords of War. In Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, Dortmund, Germany, 26–29 August 2014; pp. 1–7.
34. Oliehoek, F.A.; Spaan, M.T.J.; Terwijn, B.; Robbel, P.; Messias, J.V. The MADP toolbox: An open source library for planning and learning in (multi-)agent systems. *J. Mach. Learn. Res.* **2017**, *18*, 1–5.
35. Aicardi, M.; Davoli, F.; Minciardi, R. Decentralized optimal control of Markov chains with a common past information set. *IEEE Trans. Autom. Control* **1987**, *32*, 1028–1031. [\[CrossRef\]](#)
36. Ooi, J.M.; Wornell, G.W. Decentralized control of a multiple access broadcast channel: Performance bounds. In Proceedings of the 35th IEEE Conference on Decision and Control, Kobe, Japan, 11 December 1996; Volume 1, pp. 293–298.