Learning to Coordinate with Coordination Graphs in Repeated Single-Stage Multi-Agent Decision Problems

ICML 2018 - Vrije Universiteit Brussel and DeepMind

Shengzhe Xu, Machine Learning Group, MSRA

Background

Coordinate between multiple agents is an important problem.

Key: Exploiting loose coupling.

Loose coupling is an approach to interconnecting the components in a system or network so that those components, also called elements, depend on each other to the least extent practicable.

Background

Fully cooperative games, multi-agent multi-armed bandits (MAMABs)

Target is to prove a regret bound that is logarithmic in the number of arm pulls and only linear in the number of agents.

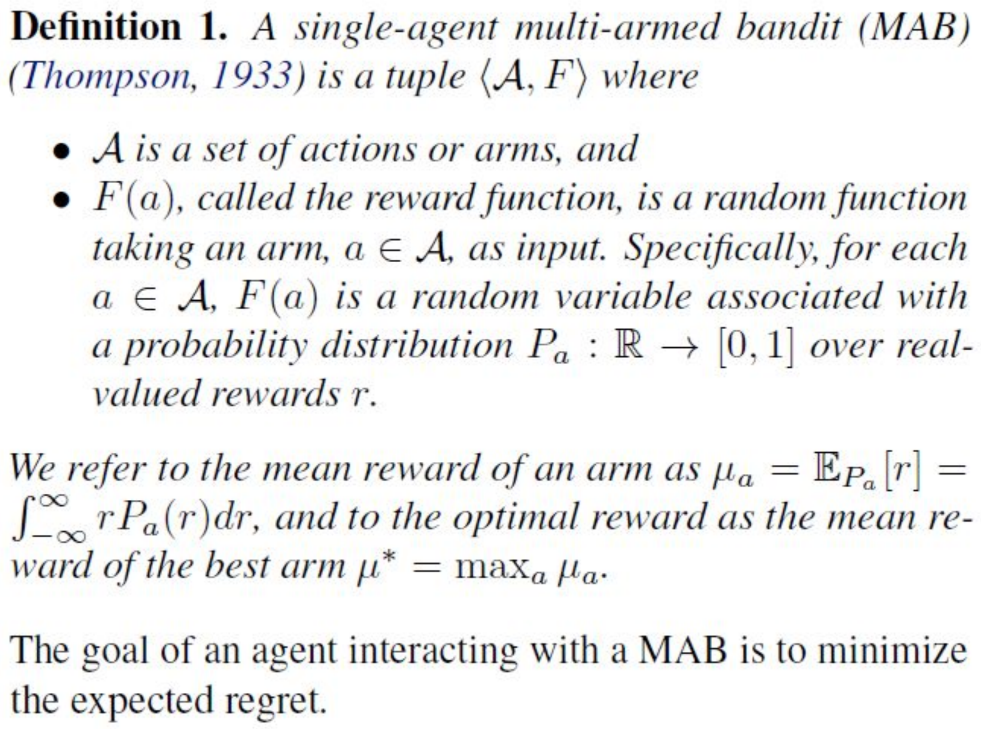
Research Problem

What: To prove a regret bound that is logarithmic in the number of arm pulls and only linear in the number of agents. Also the regret bound depends on the harmonic mean of the local upper confidence bounds, rather than their sum.

How: They propose multi-agent upper confidence exploration (MAUCE), a new algorithm for MAMABs that exploits loose couplings.

In the experiments section, it performs better than sparse cooperative Q-learning and a state-of-art combinatorial bandit approach.

Research Problem



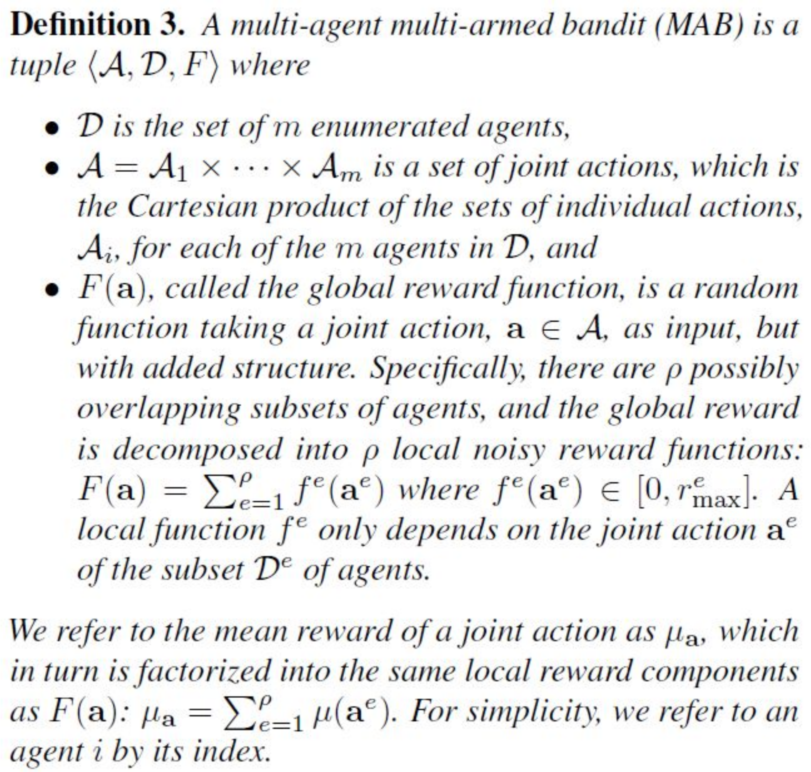
Definition 1. *A single-agent multi-armed bandit (MAB) (Thompson, 1933) is a tuple $\langle \mathcal{A}, F \rangle$ where*

- *$\mathcal{A}$ is a set of actions or arms, and*
- *$F(a)$, called the reward function, is a random function taking an arm, $a \in \mathcal{A}$, as input. Specifically, for each $a \in \mathcal{A}$, $F(a)$ is a random variable associated with a probability distribution $P_a : \mathbb{R} \to [0, 1]$ over real-valued rewards r.*

We refer to the mean reward of an arm as $\mu_a = \mathbb{E}_{P_a}[r] = \int_{-\infty}^{\infty} r P_a(r) dr$, and to the optimal reward as the mean reward of the best arm $\mu^* = \max_a \mu_a$.

The goal of an agent interacting with a MAB is to minimize the expected regret.

Research Problem



Definition 3. *A multi-agent multi-armed bandit (MAB) is a tuple $\langle \mathcal{A}, \mathcal{D}, F \rangle$ where*

- *$\mathcal{D}$ is the set of m enumerated agents,*
- *$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_m$ is a set of joint actions, which is the Cartesian product of the sets of individual actions, $\mathcal{A}_i$, for each of the m agents in $\mathcal{D}$, and*
- *$F(\mathbf{a})$, called the global reward function, is a random function taking a joint action, $\mathbf{a} \in \mathcal{A}$, as input, but with added structure. Specifically, there are ρ possibly overlapping subsets of agents, and the global reward is decomposed into ρ local noisy reward functions: $F(\mathbf{a}) = \sum_{e=1}^{\rho} f^e(\mathbf{a}^e)$ where $f^e(\mathbf{a}^e) \in [0, r_{\max}^e]$. A local function f^e only depends on the joint action $\mathbf{a}^e$ of the subset $\mathcal{D}^e$ of agents.*

We refer to the mean reward of a joint action as $\mu_{\mathbf{a}}$, which in turn is factorized into the same local reward components as $F(\mathbf{a})$: $\mu_{\mathbf{a}} = \sum_{e=1}^{\rho} \mu(\mathbf{a}^e)$. For simplicity, we refer to an agent i by its index.

Challenge

Naive approach of considering the regret of full joint action is exponential in the number of agents.

Naive approach considering a super agent whose action space could be prohibitively large.

How to balance exploration and exploitation in the joint action taken by the agents, such that the loss due to taking suboptimal joint actions during learning is bounded.

Key Concept

While learning MAUCE leverages the graphical properties of the MAMAB. It treating both exploration and exploitation as separate objectives.

MAUCE selects the action that best balances exploration and exploitation according to the joint overall mean reward plus (upper confidence) exploration bound.

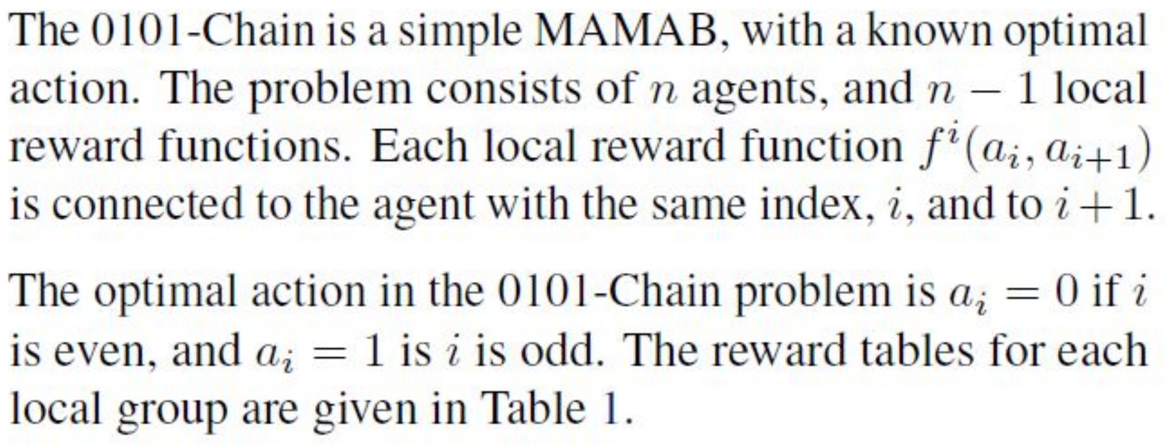
They proven a regret bound for MAUCE that is only linear in the number of agents rather than exponential.

Experiments

Baselines: uniformly random action selector; sparse cooperative Q learning; learning with linear rewards.

Games: 0101-Chain; Gem Mining; Wind Farm

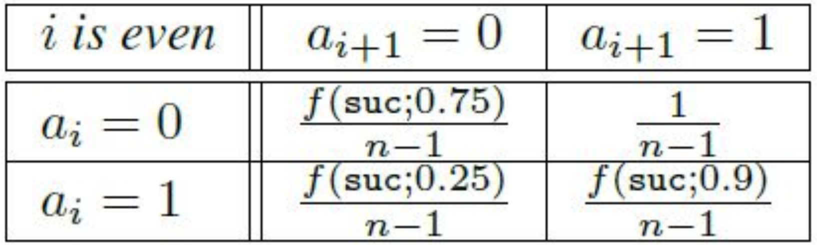
Experiments - 0101-Chain



The 0101-Chain is a simple MAMAB, with a known optimal action. The problem consists of n agents, and $n-1$ local reward functions. Each local reward function $f^i(a_i, a_{i+1})$ is connected to the agent with the same index, i, and to $i+1$.

The optimal action in the 0101-Chain problem is $a_i = 0$ if i is even, and $a_i = 1$ is i is odd. The reward tables for each local group are given in Table 1.

Experiments - 0101-Chain



i *is even*	$a_{i+1} = 0$	$a_{i+1} = 1$
$a_i = 0$	$\frac{f(\text{suc};0.75)}{n-1}$	$\frac{1}{n-1}$
$a_i = 1$	$\frac{f(\text{suc};0.25)}{n-1}$	$\frac{f(\text{suc};0.9)}{n-1}$

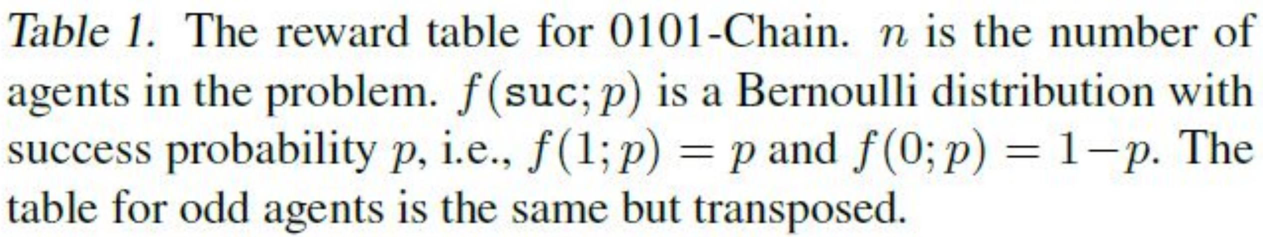
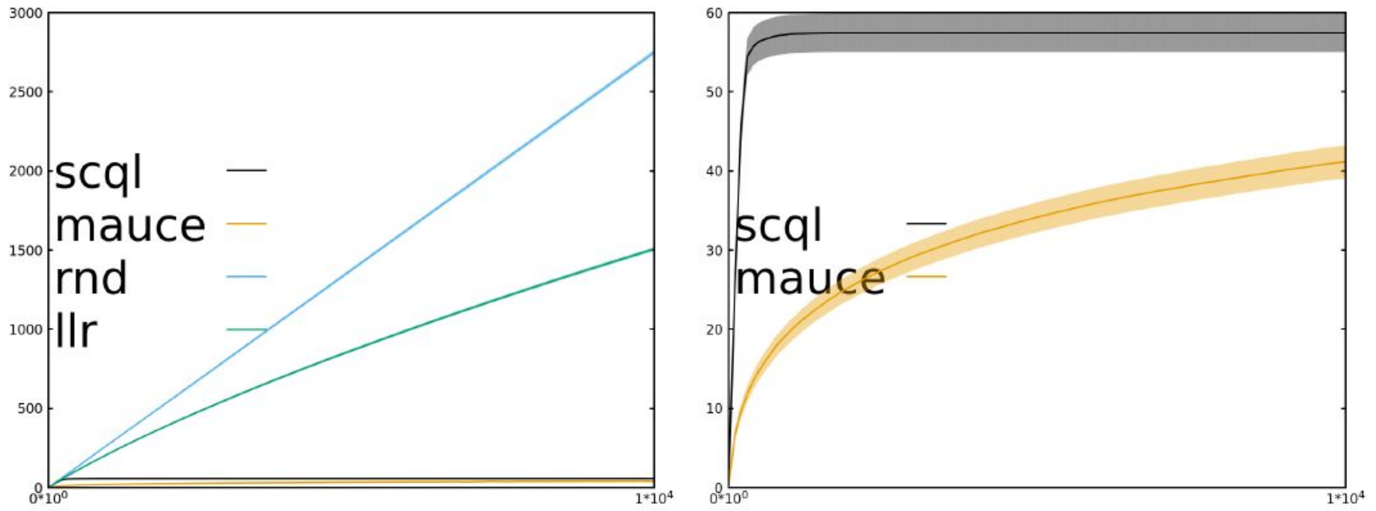


Table 1. The reward table for 0101-Chain. n is the number of agents in the problem. $f(\text{suc}; p)$ is a Bernoulli distribution with success probability p, i.e., $f(1;p) = p$ and $f(0;p) = 1-p$. The table for odd agents is the same but transposed.

Results - 0101-Chain



(a) 0101-Chain, All Algorithms (b) 0101-Ch., MAUCE&SCQL

Experiments - Gem Mining

Mining Day Problem (2015)

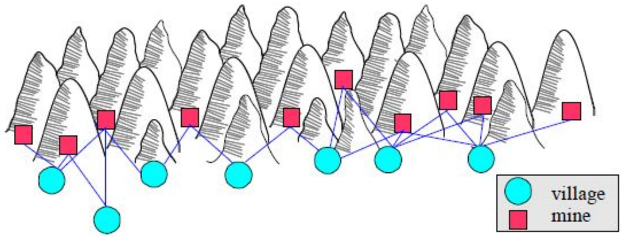
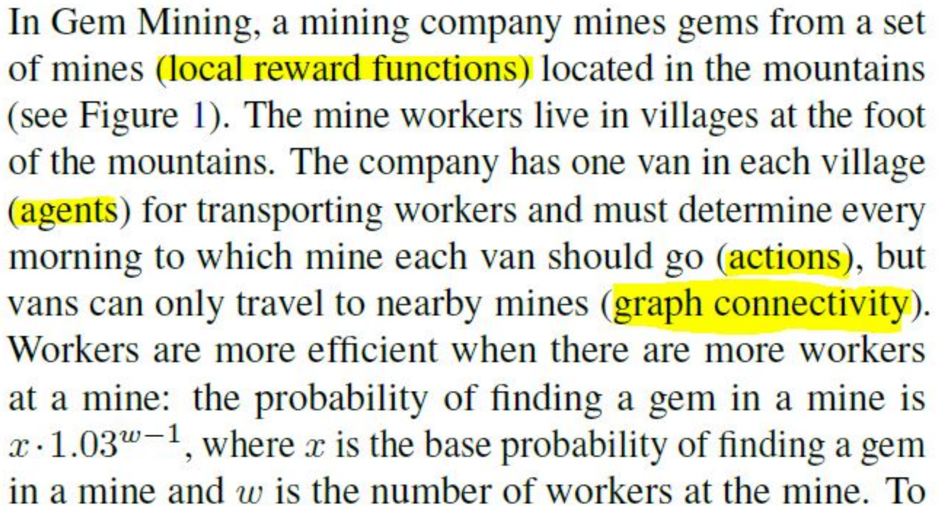


Figure 1. Gem Mining example. Each village represents an agent, while the mines represent the local reward functions.



In Gem Mining, a mining company mines gems from a set of mines (local reward functions) located in the mountains (see Figure 1). The mine workers live in villages at the foot of the mountains. The company has one van in each village (agents) for transporting workers and must determine every morning to which mine each van should go (actions), but vans can only travel to nearby mines (graph connectivity). Workers are more efficient when there are more workers at a mine: the probability of finding a gem in a mine is $x \cdot 1.03^{w-1}$, where x is the base probability of finding a gem in a mine and w is the number of workers at the mine. To

Experiments - Wind Farm

A state-of-art simulator (2016)

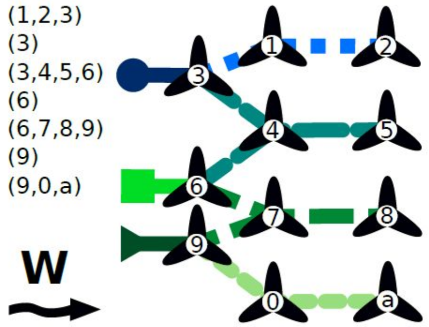


Figure 2. Wind farm setup. The incoming wind is denoted by an arrow. Each local group is denoted by a different color and line type. Groups are listed explicitly on the left. Note the three single-agent groups on the left to handle per-agent rewards.

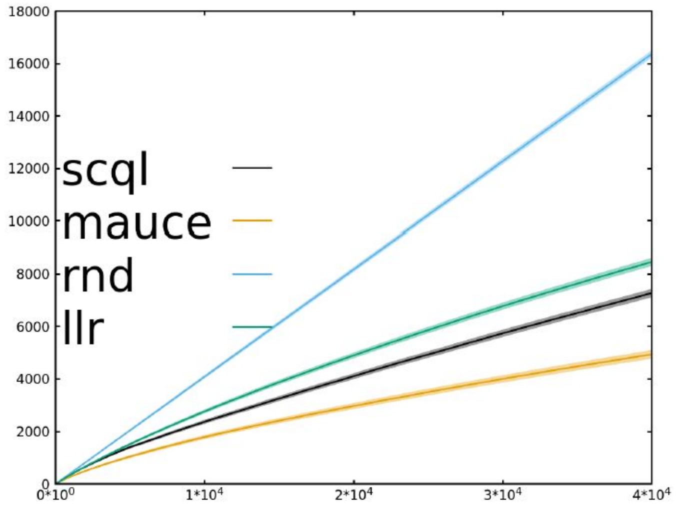
We vary the wind speed in the simulator at each timestep, following a truncated normal distribution with mean 8.1 m/s. The overall reward is normalized to a $[0, 1]$ interval using the maximum possible overall reward at the highest wind strength and the minimum possible reward per turbine at the minimum wind strength. While this makes it impossible to compute the true regret, as choosing the optimal action does not result in a 0 regret in expectation, it avoids having to calculate the true expected reward for all actions in this scenario, which is non-trivial.

Results - Gem Mining & Wind Farm

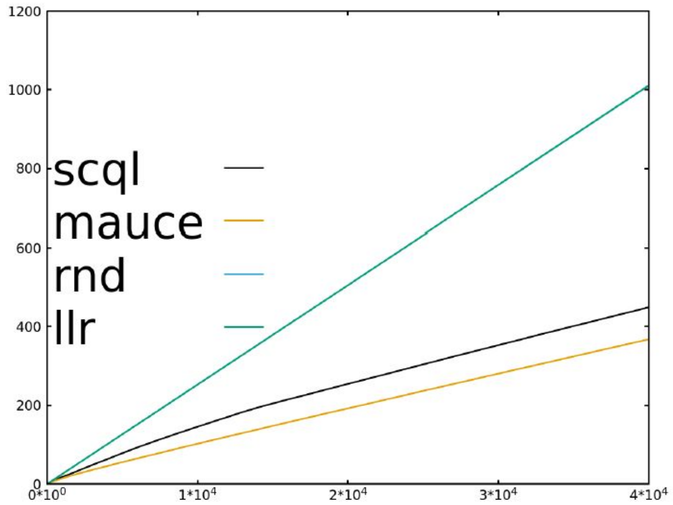
Gem Mining: rewards in this setting s are obtained by a group

Wind Farm: rewards in this setting s are obtained per-agent from the simulator

Results - Gem Mining & Wind Farm

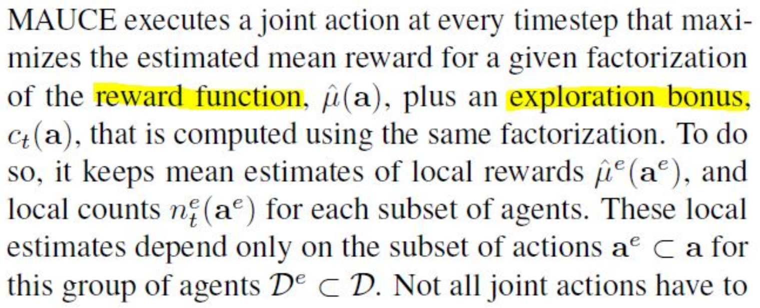


(c) Gem Mining

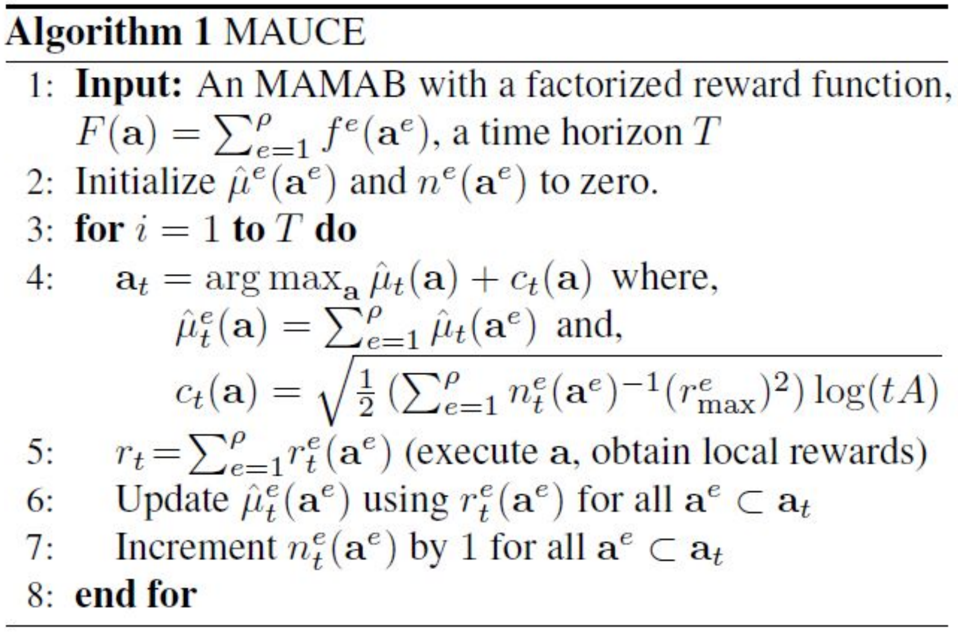


(d) Wind Farm

Approach - MAUCE



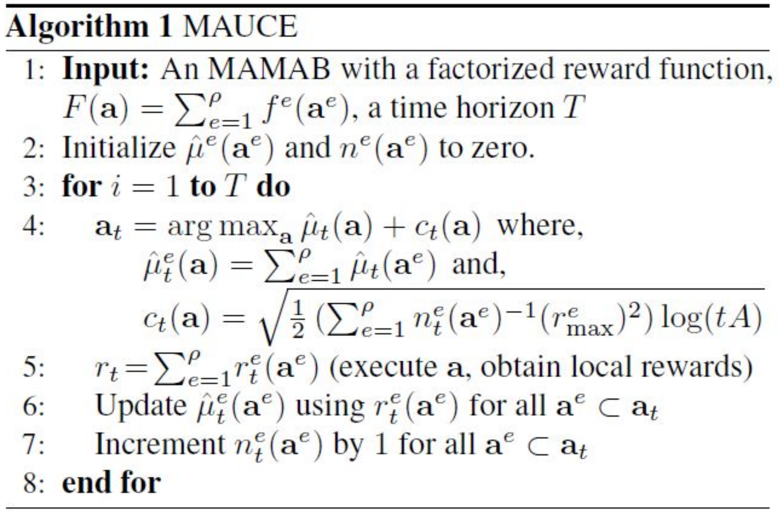
MAUCE executes a joint action at every timestep that maximizes the estimated mean reward for a given factorization of the reward function, $\hat{\mu}(\mathbf{a})$, plus an exploration bonus, $c_t(\mathbf{a})$, that is computed using the same factorization. To do so, it keeps mean estimates of local rewards $\hat{\mu}^e(\mathbf{a}^e)$, and local counts $n^e_t(\mathbf{a}^e)$ for each subset of agents. These local estimates depend only on the subset of actions $\mathbf{a}^e \subset \mathbf{a}$ for this group of agents $\mathcal{D}^e \subset \mathcal{D}$. Not all joint actions have to



Algorithm 1 MAUCE

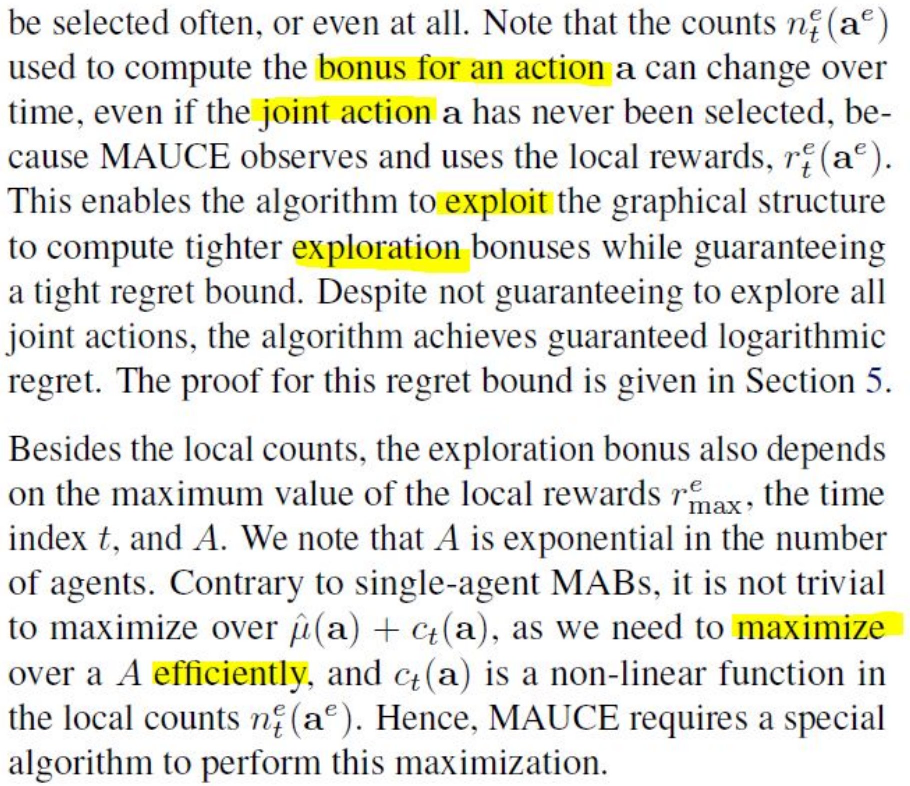
```
1: Input: An MAMAB with a factorized reward function,
   $F(\mathbf{a}) = \sum_{e=1}^{\rho} f^e(\mathbf{a}^e)$, a time horizon $T$
2: Initialize $\hat{\mu}^e(\mathbf{a}^e)$ and $n^e(\mathbf{a}^e)$ to zero.
3: for $i = 1$ to $T$ do
4:     $\mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{\mu}_t(\mathbf{a}) + c_t(\mathbf{a})$ where,
           $\hat{\mu}^e_t(\mathbf{a}) = \sum_{e=1}^{\rho} \hat{\mu}_t(\mathbf{a}^e)$ and,
           $c_t(\mathbf{a}) = \sqrt{\frac{1}{2}\left(\sum_{e=1}^{\rho} n^e_t(\mathbf{a}^e)^{-1}(r^e_{\max})^2\right)\log(tA)}$
5:     $r_t = \sum_{e=1}^{\rho} r^e_t(\mathbf{a}^e)$ (execute $\mathbf{a}$, obtain local rewards)
6:     Update $\hat{\mu}^e_t(\mathbf{a}^e)$ using $r^e_t(\mathbf{a}^e)$ for all $\mathbf{a}^e \subset \mathbf{a}_t$
7:     Increment $n^e_t(\mathbf{a}^e)$ by 1 for all $\mathbf{a}^e \subset \mathbf{a}_t$
8: end for
```

Approach - MAUCE



Algorithm 1 MAUCE

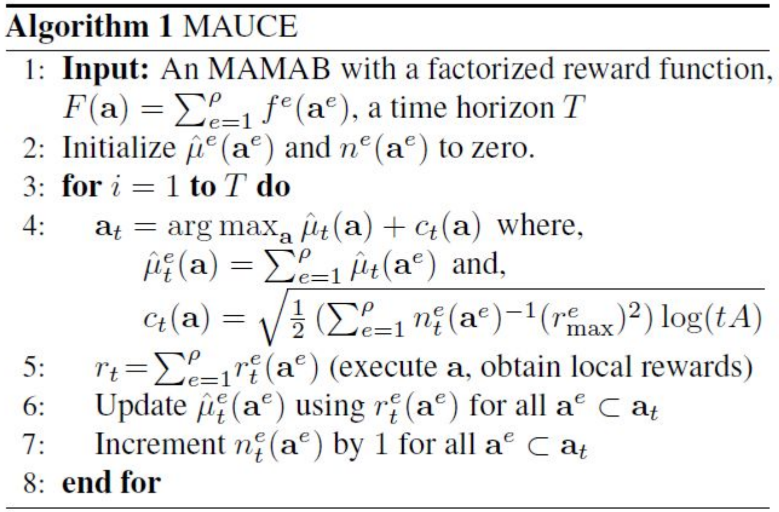
```
1: Input: An MAMAB with a factorized reward function, $F(\mathbf{a}) = \sum_{e=1}^{\rho} f^e(\mathbf{a}^e)$, a time horizon $T$
2: Initialize $\hat{\mu}^e(\mathbf{a}^e)$ and $n^e(\mathbf{a}^e)$ to zero.
3: for $i = 1$ to $T$ do
4:     $\mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{\mu}_t(\mathbf{a}) + c_t(\mathbf{a})$ where,
           $\hat{\mu}_t^e(\mathbf{a}) = \sum_{e=1}^{\rho} \hat{\mu}_t(\mathbf{a}^e)$ and,
           $c_t(\mathbf{a}) = \sqrt{\frac{1}{2}\left(\sum_{e=1}^{\rho} n_t^e(\mathbf{a}^e)^{-1}(r_{\max}^e)^2\right)\log(tA)}$
5:     $r_t = \sum_{e=1}^{\rho} r_t^e(\mathbf{a}^e)$ (execute $\mathbf{a}$, obtain local rewards)
6:     Update $\hat{\mu}_t^e(\mathbf{a}^e)$ using $r_t^e(\mathbf{a}^e)$ for all $\mathbf{a}^e \subset \mathbf{a}_t$
7:     Increment $n_t^e(\mathbf{a}^e)$ by 1 for all $\mathbf{a}^e \subset \mathbf{a}_t$
8: end for
```



be selected often, or even at all. Note that the counts $n_t^e(\mathbf{a}^e)$ used to compute the bonus for an action $\mathbf{a}$ can change over time, even if the joint action $\mathbf{a}$ has never been selected, because MAUCE observes and uses the local rewards, $r_t^e(\mathbf{a}^e)$. This enables the algorithm to exploit the graphical structure to compute tighter exploration bonuses while guaranteeing a tight regret bound. Despite not guaranteeing to explore all joint actions, the algorithm achieves guaranteed logarithmic regret. The proof for this regret bound is given in Section 5.

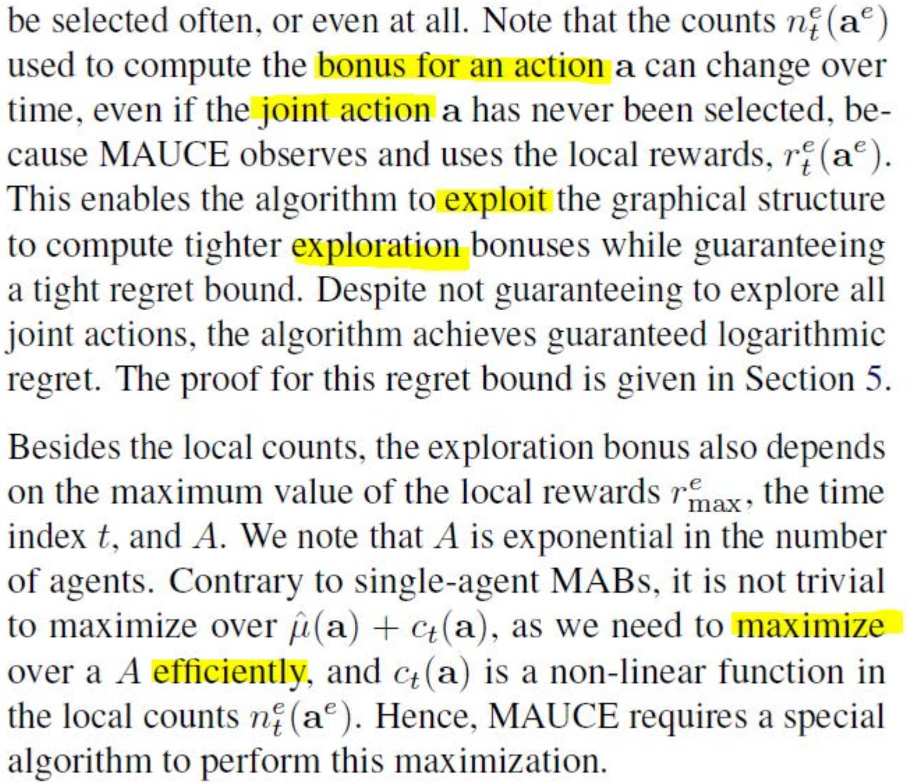
Besides the local counts, the exploration bonus also depends on the maximum value of the local rewards $r_{\max}^e$, the time index t, and A. We note that A is exponential in the number of agents. Contrary to single-agent MABs, it is not trivial to maximize over $\hat{\mu}(\mathbf{a}) + c_t(\mathbf{a})$, as we need to maximize over a A efficiently, and $c_t(\mathbf{a})$ is a non-linear function in the local counts $n_t^e(\mathbf{a}^e)$. Hence, MAUCE requires a special algorithm to perform this maximization.

Approach - MAUCE



Algorithm 1 MAUCE

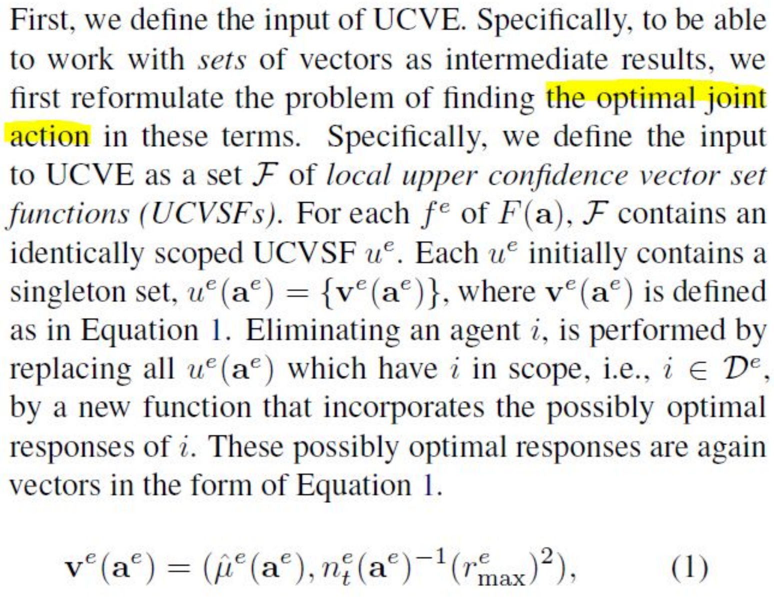
```
1: Input: An MAMAB with a factorized reward function,
   $F(\mathbf{a}) = \sum_{e=1}^{\rho} f^e(\mathbf{a}^e)$, a time horizon $T$
2: Initialize $\hat{\mu}^e(\mathbf{a}^e)$ and $n^e(\mathbf{a}^e)$ to zero.
3: for $i = 1$ to $T$ do
4:     $\mathbf{a}_t = \arg\max_{\mathbf{a}} \hat{\mu}_t(\mathbf{a}) + c_t(\mathbf{a})$ where,
           $\hat{\mu}_t^e(\mathbf{a}) = \sum_{e=1}^{\rho} \hat{\mu}_t(\mathbf{a}^e)$ and,
           $c_t(\mathbf{a}) = \sqrt{\frac{1}{2}\left(\sum_{e=1}^{\rho} n_t^e(\mathbf{a}^e)^{-1}(r_{\max}^e)^2\right)\log(tA)}$
5:     $r_t = \sum_{e=1}^{\rho} r_t^e(\mathbf{a}^e)$ (execute $\mathbf{a}$, obtain local rewards)
6:     Update $\hat{\mu}_t^e(\mathbf{a}^e)$ using $r_t^e(\mathbf{a}^e)$ for all $\mathbf{a}^e \subset \mathbf{a}_t$
7:     Increment $n_t^e(\mathbf{a}^e)$ by 1 for all $\mathbf{a}^e \subset \mathbf{a}_t$
8: end for
```



be selected often, or even at all. Note that the counts $n_t^e(\mathbf{a}^e)$ used to compute the bonus for an action $\mathbf{a}$ can change over time, even if the joint action $\mathbf{a}$ has never been selected, because MAUCE observes and uses the local rewards, $r_t^e(\mathbf{a}^e)$. This enables the algorithm to exploit the graphical structure to compute tighter exploration bonuses while guaranteeing a tight regret bound. Despite not guaranteeing to explore all joint actions, the algorithm achieves guaranteed logarithmic regret. The proof for this regret bound is given in Section 5.

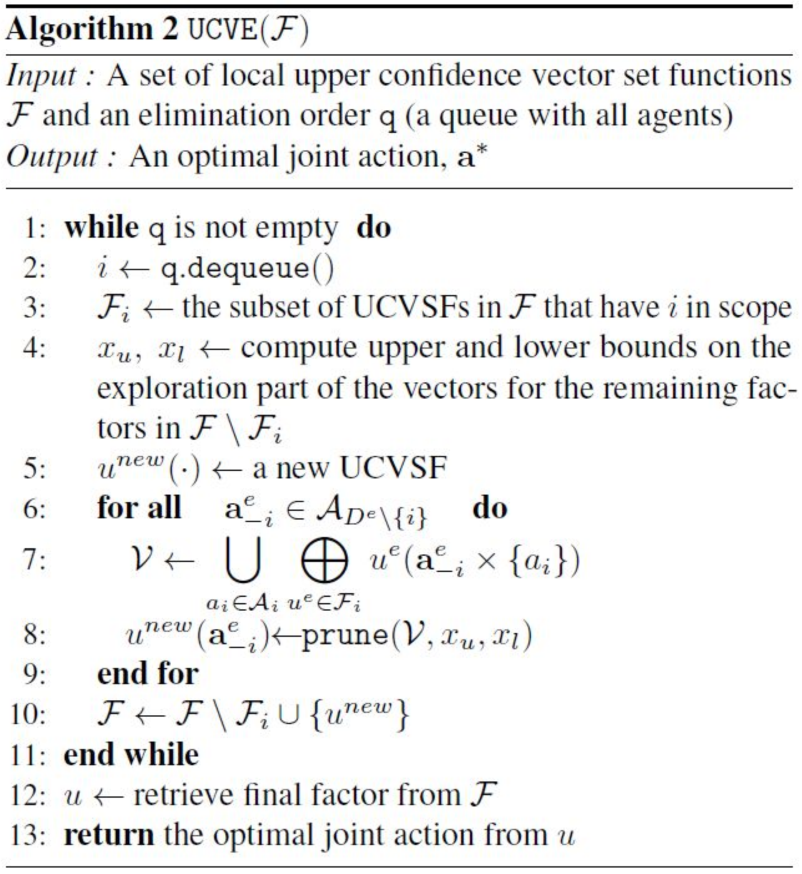
Besides the local counts, the exploration bonus also depends on the maximum value of the local rewards $r_{\max}^e$, the time index t, and A. We note that A is exponential in the number of agents. Contrary to single-agent MABs, it is not trivial to maximize over $\hat{\mu}(\mathbf{a}) + c_t(\mathbf{a})$, as we need to maximize over a A efficiently, and $c_t(\mathbf{a})$ is a non-linear function in the local counts $n_t^e(\mathbf{a}^e)$. Hence, MAUCE requires a special algorithm to perform this maximization.

Approach - UCVE



First, we define the input of UCVE. Specifically, to be able to work with *sets* of vectors as intermediate results, we first reformulate the problem of finding the optimal joint action in these terms. Specifically, we define the input to UCVE as a set $\mathcal{F}$ of *local upper confidence vector set functions (UCVSFs)*. For each f^e of $F(\mathbf{a})$, $\mathcal{F}$ contains an identically scoped UCVSF u^e. Each u^e initially contains a singleton set, $u^e(\mathbf{a}^e) = \{\mathbf{v}^e(\mathbf{a}^e)\}$, where $\mathbf{v}^e(\mathbf{a}^e)$ is defined as in Equation 1. Eliminating an agent i, is performed by replacing all $u^e(\mathbf{a}^e)$ which have i in scope, i.e., $i \in \mathcal{D}^e$, by a new function that incorporates the possibly optimal responses of i. These possibly optimal responses are again vectors in the form of Equation 1.

$$\mathbf{v}^e(\mathbf{a}^e) = (\hat{\mu}^e(\mathbf{a}^e), n_t^e(\mathbf{a}^e)^{-1}(r_{\max}^e)^2), \qquad (1)$$

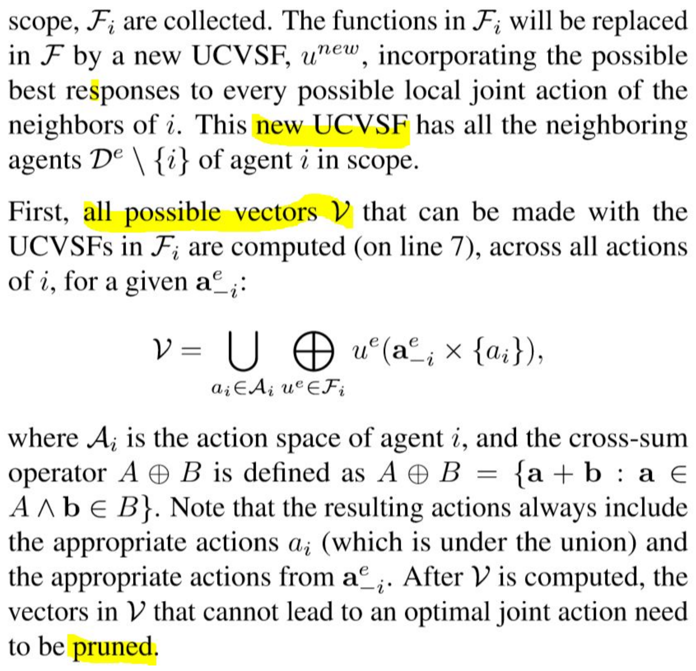


Algorithm 2 $\texttt{UCVE}(\mathcal{F})$

Input : A set of local upper confidence vector set functions $\mathcal{F}$ and an elimination order q (a queue with all agents)
Output : An optimal joint action, $\mathbf{a}^*$

```
1: while q is not empty do
2:     $i \leftarrow$ q.dequeue()
3:     $\mathcal{F}_i \leftarrow$ the subset of UCVSFs in $\mathcal{F}$ that have $i$ in scope
4:     $x_u, x_l \leftarrow$ compute upper and lower bounds on the exploration part of the vectors for the remaining factors in $\mathcal{F} \setminus \mathcal{F}_i$
5:     $u^{new}(\cdot) \leftarrow$ a new UCVSF
6:     for all $\mathbf{a}^e_{-i} \in \mathcal{A}_{\mathcal{D}^e \setminus \{i\}}$ do
7:         $\mathcal{V} \leftarrow \bigcup_{a_i \in \mathcal{A}_i} \bigoplus_{u^e \in \mathcal{F}_i} u^e(\mathbf{a}^e_{-i} \times \{a_i\})$
8:         $u^{new}(\mathbf{a}^e_{-i}) \leftarrow \texttt{prune}(\mathcal{V}, x_u, x_l)$
9:     end for
10:    $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_i \cup \{u^{new}\}$
11: end while
12: $u \leftarrow$ retrieve final factor from $\mathcal{F}$
13: return the optimal joint action from $u$
```

Approach - UCVE

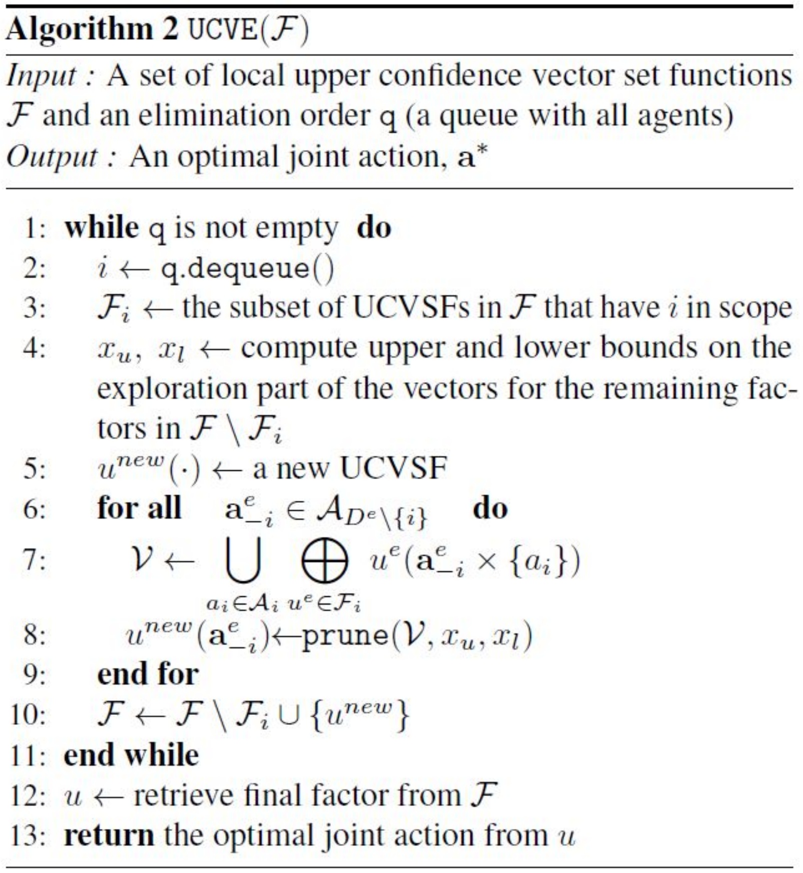


scope, $\mathcal{F}_i$ are collected. The functions in $\mathcal{F}_i$ will be replaced in $\mathcal{F}$ by a new UCVSF, u^{new}, incorporating the possible best responses to every possible local joint action of the neighbors of i. This new UCVSF has all the neighboring agents $\mathcal{D}^e \setminus \{i\}$ of agent i in scope.

First, all possible vectors $\mathcal{V}$ that can be made with the UCVSFs in $\mathcal{F}_i$ are computed (on line 7), across all actions of i, for a given $\mathbf{a}^e_{-i}$:

$$\mathcal{V} = \bigcup_{a_i \in \mathcal{A}_i} \bigoplus_{u^e \in \mathcal{F}_i} u^e(\mathbf{a}^e_{-i} \times \{a_i\}),$$

where $\mathcal{A}_i$ is the action space of agent i, and the cross-sum operator $A \oplus B$ is defined as $A \oplus B = \{\mathbf{a} + \mathbf{b} : \mathbf{a} \in A \wedge \mathbf{b} \in B\}$. Note that the resulting actions always include the appropriate actions a_i (which is under the union) and the appropriate actions from $\mathbf{a}^e_{-i}$. After $\mathcal{V}$ is computed, the vectors in $\mathcal{V}$ that cannot lead to an optimal joint action need to be pruned.



Algorithm 2 UCVE($\mathcal{F}$)

Input : A set of local upper confidence vector set functions $\mathcal{F}$ and an elimination order q (a queue with all agents)
Output : An optimal joint action, $\mathbf{a}^*$

```
1:  while q is not empty do
2:      $i \leftarrow$ q.dequeue()
3:      $\mathcal{F}_i \leftarrow$ the subset of UCVSFs in $\mathcal{F}$ that have $i$ in scope
4:      $x_u, x_l \leftarrow$ compute upper and lower bounds on the
        exploration part of the vectors for the remaining fac-
        tors in $\mathcal{F} \setminus \mathcal{F}_i$
5:      $u^{new}(\cdot) \leftarrow$ a new UCVSF
6:      for all $\mathbf{a}^e_{-i} \in \mathcal{A}_{D^e \setminus \{i\}}$ do
7:          $\mathcal{V} \leftarrow \bigcup_{a_i \in \mathcal{A}_i} \bigoplus_{u^e \in \mathcal{F}_i} u^e(\mathbf{a}^e_{-i} \times \{a_i\})$
8:          $u^{new}(\mathbf{a}^e_{-i}) \leftarrow$ prune($\mathcal{V}, x_u, x_l$)
9:      end for
10:     $\mathcal{F} \leftarrow \mathcal{F} \setminus \mathcal{F}_i \cup \{u^{new}\}$
11: end while
12: $u \leftarrow$ retrieve final factor from $\mathcal{F}$
13: return the optimal joint action from $u$
```

Thanks