An Anytime Algorithm for Finding the ϵ-Core in Nontransferable Utility Coalitional Games

Greg Hines and **Talal Rahwan** and **Nicholas R. Jennings**[1]

Abstract. We provide the first anytime algorithm for finding the ϵ-core in a nontransferable utility coalitional game. For a given set of possible joint actions, our algorithm calculates ϵ, the maximum utility any agent could gain by deviating from this set of actions. If ϵ is too high, our algorithm searches for a subset of the joint actions which leads to a smaller ϵ. Simulations show our algorithm is more efficient than an exhaustive search by up to 2 orders of magnitude.

1 Introduction

Coalitional games are an important tool in multi-agent systems and in AI in general. These games allow self-interested agents to work together to achieve outcomes that could not be achieved by a single agent, even when these agents have different priorities. Coalition formation can be used in applications ranging from management of the smart grid or e-marketplaces to communication networks [4, 5]. We can even use coalition formation to improve disaster response: for example, the lack of coordination between the thousands of aid groups responding to the Haiti earthquake led to a slow and chaotic response [10].

In many of these settings, we can assume the existence of transferable utility (TU) [6]. The outcome achieved by a coalition is some form of currency such as money, units of electricity, or throughput on a network, that can be transferred from one agent to another. Transferable utility can therefore be seen as a useful tool for helping agents reach a consensus.

However, we are interested in settings where transferable utility does not exist. When helping with disaster response, for example, aid groups are (ideally) not making money from saving lives, while coalitions in wireless networks may assign a specific rate for each user in the coalition which is not transferable [4]. These types of situations must instead be represented by nontransferable utility (NTU) games. With these games, the utility an agent receives from an outcome is completely intrinsic to itself and cannot be shared. Without transferable utility, reaching an agreement between agents can be considerably more difficult.

For both TU and NTU games, a commonly considered solution concept is the ϵ-core [6]. The ϵ-core consists of all partitions of agents along with an assigned action for each agent such that no agent can increase its utility by more than ϵ by getting a coalition of agents to defect. While the core is often empty, for a large enough ϵ, the ϵ-core will always be non-empty [6]. For TU games, there has been considerable research on algorithms which can find the ϵ-core or other solution concepts efficiently [3, 8, 9]. For NTU games, while there has been work done on characterising when the core exists in NTU, there has not been any work done on efficiently finding the ϵ-core [7].

Against this background, we present the first anytime algorithm for efficiently finding a coalition structure and joint action in the ϵ-core, for any $\epsilon \geq 0$ such that the ϵ-core is non-empty. Our algorithm is anytime in that it works by repeatedly finding the ϵ-core for smaller and smaller values of ϵ, until the desired ϵ value is reached. With anytime algorithms, it is possible to trade off computation time for solution quality; if finding; if finding the desired ϵ proves to be too time-consuming, we can terminate the algorithm early and take the best ϵ found so far. The anytime property creates an algorithm which is more applicable to the real world. For example, in a disaster response scenario agents may not have time to wait around until the perfect solution is found.

In more detail, our algorithm is based on the idea of *regret*; the loss of utility a group of agents receive for accepting a given joint action instead of choosing a different one. The goal is to recommend a joint action which minimizes the regret for all agents. If the resulting regret is less than ϵ, we have found the ϵ-core. Since there is an exponential number of possible joint actions, two challenges are how to efficiently calculate regret and how to minimize it. To address both of these challenges, we build on two ideas from research into *preference elicitation*: namely, *minimax regret* and *utility independence models*.

The paper is presented as follows. We start by presenting our model and reviewing related work. Next, we present our algorithm and experimental results. Finally we conclude and discuss future work.

2 Model and Related Work

2.1 Coalitional Game Theory

Our general setting is a coalition game with n agents in the set $\mathbb{N}$. An agent can be either a person, a specific group such as an NGO, or a piece of software or hardware. Agent i has a finite set of possible actions, $\mathbb{A}_i$, to choose from. A coalition $p \subseteq \mathbb{N}$ has a set of possible joint actions $\mathbb{A}_p = \times_{i \in p} \mathbb{A}_i$ to choose from. The set of actions over all agents is $\mathbb{A}_\mathbb{N}$. A partition of agents into a set of coalitions is known as a *coalition structure*. Let $p(CS, i)$ be the set of agents in the same coalition as agent i (including itself) given the coalition structure CS. An

[1] University of Southampton, UK, email: {gh2,tr,nrj}@ecs.soton.ac.uk

implementation $(\{p\}, A_{\{p\}})$ is a set of disjoint coalitions and a joint action for each of those coalitions.

Agent i's preferences between joint actions is given by a utility function $u_i : \mathbb{A}_{p(CS,i)} \to \mathbb{R}$. This utility is completely internal to the agent and cannot be transferred from one agent to another. The ordering of the preferences can be completely different for every agent. An agent's utility value is determined only by the joint action which its coalition chooses, *i.e.* the coalitions do not interact with each other.

A *defecting implementation* $(\{p\}, A_{\{p\}})$ *blocks* the implementation (CS, A) (where p may or may not be in CS) if every agent in p either prefers or is indifferent to $(\{p\}, A_{\{p\}})$ over (CS, A) and at least one agent in p strictly prefers $(\{p\}, A_{\{p\}})$. An implementation is in the *core* if no coalition of agents wishes to defect from the implementation. Since the core may be empty or difficult to find, we can also consider the ϵ-core [6]. An implementation is in the ϵ-core if it is blocked by a defecting implementation but no agent in the defecting coalitions increases their utility by more than ϵ. Note that since coalitions do not interact with each other, we need only consider the possible defections and not the possible alternative implementations. For example, if there are two possible defections, then if the first defection blocks the implementation it does so regardless of whether the second defection occurs. This helps since the number of possible defections may be considerably less than the number of possible alternative implementations.

The goal of our work is to provide an algorithm which can help a group of agents that are trying to find an implementation which they can all agree on. We consider a setting where the agents approach a third party who will use our algorithm to find the best possible implementation in a given amount of time. This implementation is the default implementation; agents agree to follow this implementation unless a coalition of agents is willing to defect. The default implementation will be in the ϵ-core, for some $\epsilon \geq 0$. If $\epsilon > 0$, the agents will have to decide if it is worth giving the third party additional time to run our algorithm to try and reduce ϵ. We assume that all agents truthfully report their utility values.

2.2 Preference Elicitation

Our algorithm searches for an implementation in the ϵ-core by finding an implementation which minimizes *regret.* Regret is the loss of utility an agent receives for accepting an implementation instead of defecting. Creating an algorithm based on regret allows us to build on research from preference elicitation on how to efficiently calculate and how to minimize it. We can illustrate the use of preference elicitation by considering a slightly altered coalition formation problem. Suppose we have the mode from Section 2.1 but with an added user who is able to decide on the actions taken by each agent. This user has a utility function $u : \mathbb{A}_{\mathbb{N}} \to \mathbb{R}$ and will choose an action which maximizes u, *i.e.* the user does not care about the agents' individual utilities.

In the worst case, the complexity of representing u is exponential with respect to the number of agents [1]. However, we can often find a more compact representation of u. For example, a common compact representation of u is additive independence where we decompose u as

$$u(A) = \sum_{i \in \mathbb{N}} u(A_i), \tag{1}$$

where $u(A_i) : A_i \to R$ is a utility function with respect to only A_i [1]. The complexity of an additive independence representation is linear.

Suppose additive independence holds and we are helping the user choose between two joint actions A and A'. The downside of additive independence is that we cannot choose the best joint action merely by ordering A_i. Instead, choosing the best joint action requires some information about the actual values for $u(A_i)$ [1]. A standard assumption in preference elicitation is that, due to cognitive limitations, instead of providing specific values for $u(A_i)$, the user can only provide lower and upper bounds [1].

Based on only these lower and upper bounds, we cannot calculate which joint action maximizes the user's utility. Rather, a common alternative is to choose the joint action which minimizes the maximum possible *regret* or loss of utility. The maximum regret for choosing A is

$$MR(A) = u^{upper}(A') - u^{lower}(A)$$

where $u^{upper}(A')$ is an upper bound on $u(A')$ and $u^{lower}(A)$ is a lower bound on $u(A)$. These upper and lower bounds can be calculated using Equation 1 as well as the upper and lower bounds for $u(A_i)$ provided by the user. Suppose that $MR(A) = 0.3$ and $MR(A') = 0.2$. Now, Braziunas and Boutilier argue that, if we know nothing else about the user's preferences, we should choose the joint action A' since it achieves the minimax regret [1]. Since minimax regret and utility independence models are used in preference elicitation to efficiently calculate and minimize regret, we believe there is considerable value in adapting both of them for use with calculating and minimizing regret in a coalition formation setting.

3 The Algorithm

In this section we present our algorithm, *Minimax Coalition Formation* (MCF). Our goal is to maximize the agents' utility values by finding an implementation that all agents can agree to, *i.e.* no coalition of agents will wish to defect. If such an implementation exists, it is in the core. Since the core may be empty or may be impractical to find, we also consider the goal of finding an implementation that is in the ϵ-core for a small value of ϵ. Our algorithm can be stopped at any time and will return the best implementation found so far. With enough time, our algorithm will find an implementation in the ϵ-core for any ϵ such that the ϵ-core is nonempty. Our main contribution is achieving these goals by adapting minimax regret for use with coalition formation.

Our first step is to redefine regret in terms of coalition formation. We begin by defining an agent's regret for accepting a given implementation instead of defecting.

Definition 1 (Individual Pairwise Regret)**.** *Individual pairwise regret is the loss of utility that agent i receives from accepting the implementation (CS, A) instead of the defecting implementation $(\{p\}, A'_{\{p\}})$, i.e.,*

$$PR(i, (CS, A), (p, A'_p)) = u_i(p, A'_p) - u_i(CS, A).$$

For brevity, we will omit CS and p.

For example, if $PR(i, A, A'_p) > 0$, then agent i strictly prefers the implementation (p, A'_p) over (CS, A). However, agent i can only defect to (p, A'_p) if all the other agents in p also strictly prefer (p, A'_p) or are at least indifferent. Otherwise at least one agent in p will block the defection. This leads to the idea of a *feasible defection.*

Definition 2 (Feasible defection)**.** *The defecting implementation $(\{p\}, A'_{\{p\}})$ is feasible with respect to (CS, A) if*

$$\min_{i \in p} PR(i, A, A'_p) \geq 0,$$

that is, all agents in p prefer (p, A'_p) over (CS, A) or are at least indifferent.

Using the idea of a feasible defection, we can generalize pairwise regret from one agent to a coalition.

Definition 3 (Coalition Pairwise Regret)**.** *The coalition pairwise regret for the coalition of agents p with respect to the implementation (CS, A) versus (p, A') is*

$$PR(A, A'_p) = \begin{cases} \max_{i \in p} PR(i, A, A'_p) & \text{if } \min_{i \in p} PR(i, A, A'_p) \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

As a result, the coalition pairwise regret is zero as long as at least one agent in p objects to the defection.

We next generalize pairwise regret to being with respect to the set of joint actions $\mathbb{A}_p$ and use this generalization to determine if (CS, A) is in the ϵ-core. For notational brevity, in the remainder of this paper, we assume that only the coalition p is threatening to defect. The generalization to multiple defections is straightforward. Indeed, our experimental results examine this general case.

Definition 4 (Pairwise regret)**.** *For the implementation (CS, A) and possible defection by the coalition p with the set of possible joint actions $\mathbb{A}_p$, the pairwise regret is*

$$PR(A, \mathbb{A}_p) = \max_{A'_p \in \mathbb{A}_p} PR(A, A'_p).$$

Lemma 1. *The implementation (CS, A) is in the ϵ-core if and only if $PR(A, \mathbb{A}_p) \leq \epsilon$.*

Proof omitted for brevity.

Since p can choose its joint action after we have picked A, we must pick a joint action which minimizes the worst-case possible regret that could be caused by p. In this case, the optimal joint action to choose is

$$A^{opt}(CS, p) = \arg\min_{A \in \mathbb{A}} PR(A, \mathbb{A}_p),$$

and the corresponding best worst case regret is

$$r^{opt}(CS, p) = \min_{A \in \mathbb{A}} PR(A, \mathbb{A}_p).$$

Unfortunately, calculating r^{opt} (or finding A^{opt}) requires an exhaustive search of $\mathbb{A}$ and $\mathbb{A}_p$. This is because calculating r^{opt} requires finding the worst case defection which all agents agree to, which in the worst case may require an exhaustive search. We address this problem by instead using an upper bound on r^{opt} based on *relaxed pairwise regret.*

Definition 5 (Relaxed Pairwise Regret)**.** *For the implementation (CS, A) and defecting implementation $(\{p\}, A'_{\{p\}})$, the relaxed pairwise regret*

$$PR^+(A, A'_p) = \max_{i \in p} PR(i, A, A'_p).$$

That is, the relaxed regret is the maximum regret over all agents in p, regardless of whether or not all agents would agree to the joint action A'_p. We can similarly define the relaxed pairwise regret *with respect to $\mathbb{A}_p$ as*

$$PR^+(A, \mathbb{A}_p) = \max_{A'_p \in \mathbb{A}_p} PR^+(A, A'_p).$$

Since $PR^+ \geq PR$, PR^+ is always a valid upper bound on the possible regret. Thus, trying to minimize PR^+ is a reasonable alternative to trying to minimize PR. However, as the following derivation shows, we encounter another problem when trying to efficiently minimize regret:

$$\begin{aligned} &\min_{A \in \mathbb{A}} PR^+(A, \mathbb{A}_p) \\ &= \min_{A \in \mathbb{A}} \max_{A'_p \in \mathbb{A}_p} \max_{i \in p} PR(i, A, A'_p) \\ &= \min_{A \in \mathbb{A}} \max_{i \in p} \max_{A'_p \in \mathbb{A}_p} PR(i, A, A'_p) \\ &= \min_{A \in \mathbb{A}} \max_{i \in p} \max_{A'_p \in \mathbb{A}_p} \left[u_i(p, A'_p) - u_i(CS, A) \right] \\ &= \min_{A \in \mathbb{A}} \max_{i \in p} \left[\max_{A'_p \in \mathbb{A}_p} u_i(p, A'_p) - u_i(CS, A) \right] \end{aligned}$$

Since $\min_y \max_x f(x, y) \neq \max_x \min_y f(x, y)$, we cannot further simplify this equation. To overcome this, we consider a relaxed goal of finding an upper bound on the optimal regret using *maximum regret.*

Definition 6 (Maximum regret)**.** *Given a coalition structure CS, the maximum regret is*

$$MR(\mathbb{A}, \mathbb{A}_p) = \max_{A \in \mathbb{A}} PR^+(A, \mathbb{A}_p). \quad (2)$$

Calculating the maximum regret is straightforward:

$$\begin{aligned} MR(\mathbb{A}, \mathbb{A}_p) &= \max_{A \in \mathbb{A}} \max_{A'_p \in \mathbb{A}_p} \max_{i \in p} PR(i, A, A'_p) \\ &= \max_{A \in \mathbb{A}} \max_{i \in p} \left[\max_{A'_p \in \mathbb{A}_p} u_i(p, A'_p) - u_i(CS, A) \right] \\ &= \max_{i \in p} \left[\max_{A'_p \in \mathbb{A}_p} u_i(p, A'_p) - \min_{A \in \mathbb{A}} u_i(CS, A) \right] \end{aligned}$$

Using additive independence from Equation 1 we can further simplify this equation so that it can be calculated efficiently. Letting $A'_{p,j}$ be the action for agent j in A'_p:

$$\begin{aligned} MR(\mathbb{A}, \mathbb{A}_p) &= \max_{i \in p} \left[\max_{A'_p \in \mathbb{A}_p} \sum_{j \in p} u(A'_{p,j}) - \min_{A \in \mathbb{A}} \sum_{j \in p(CS,i)} u(A_j) \right] \\ &= \max_{i \in p} \left[\sum_{j \in p} \max_{A_j \in \mathbb{A}_j} u(A_j) - \sum_{j \in p(CS,i)} \min_{A_j \in \mathbb{A}_j} u(A_j) \right]. \end{aligned} \quad (3)$$

Since Equation 3 is maximizing over the sets of actions for each agent as opposed to over all joint actions, Equation 3 can be calculated in polynomial time even when there is an exponential number of possible joint actions.

We calculate the maximum regret for every possible coalition structure. The coalition structure with the lowest maximum regret is the *minimax coalition structure* (CS^*) and the corresponding regret is the *minimax regret* (MMR). We should choose this coalition structure since it guarantees the lowest worst-case regret. If $MMR \leq \epsilon$, then any implementation based on CS^* will result in the maximum regret being less than or equal to ϵ. If ϵ is below some desired threshold, we can save considerable time by not having to figure out which joint action gives the best implementation, since all will work.

If ϵ is too large, we reduce it by using our algorithm *Minimax Coalition Formation* (MCF). Our first step is defining a relaxation of (joint) actions, *(joint) partial actions.*

Definition 7 ((Joint) Partial Actions)**.** *A joint partial action $J_p \subseteq \mathbb{A}_p$ is given by*

$$J_p = \times_{i \in p} J_i,$$

where $J_i \subseteq \mathbb{A}_i$ is a partial action.

Given a joint partial action $J \subseteq \mathbb{A}$ and a set of joint partial actions $\mathbb{J}_p$ over $\mathbb{A}_p$, we can define a generalization of maximum regret (Equation 2):

$$MR(J, \mathbb{J}_p) = \max_{A \in J} \max_{J_p \in \mathbb{J}_p} \max_{A'_p \in J_p} PR^+(A, A'_p). \qquad (4)$$

For example, our original definition of maximum regret can be written as $MR(\mathbb{A}, \{\mathbb{A}_p\})$. Similarly to Equation 3, we can calculate Equation 4 in polynomial time with respect to the number of partial joint actions in $\mathbb{J}_p$.

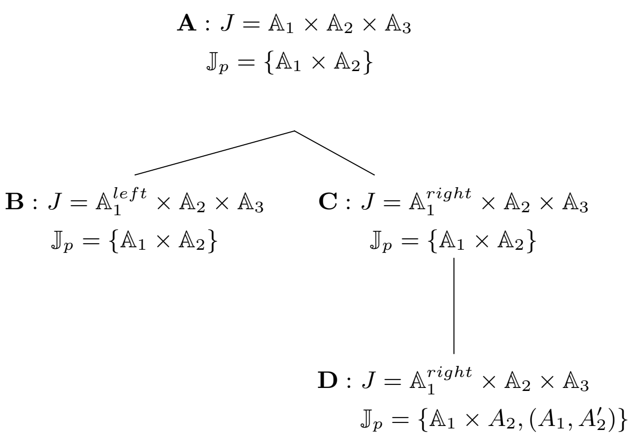
A preliminary step in decreasing the maximum regret is to make sure that we only consider the Pareto optimal joint actions in J. A Pareto optimal joint action is one which cannot be changed in any way without decreasing the utility for at least one agent. If $A \in J$ is not a Pareto optimal joint action, then there is at least one joint action in J which every agent either strictly prefers over A or is indifferent to. As a result, including A in J can increase the maximum regret. With additive independence, we can use joint partial actions to efficiently express all Pareto optimal joint actions. If we let $\mathbb{A}^{PO}_{\mathbb{N}}$ be the set of all Pareto optimal actions and $\mathbb{A}^{PO}_i$ be the set of Pareto optimal actions with respect to $\mathbb{A}_i$ then

$$\mathbb{A}^{PO}_{\mathbb{N}} = \times_{i \in \mathbb{N}} \mathbb{A}^{PO}_i. \qquad (5)$$

Proof omitted for brevity. The set of Pareto optimal joint actions is thus a joint partial action. As a result, we decrease the maximum regret by setting J in Equation 4 to Equation 5.

Decreasing the maximum regret means refining either J or $\mathbb{J}_p$ in Equation 4. MCF refines J and $\mathbb{J}_p$ using two methods, *action branching* and *defection pruning*, respectively. Action branching splits J into smaller joint partial actions and chooses the subset which minimizes Equation 4. Defection pruning removes joint actions in $\mathbb{J}_p$ which do not lead to feasible defections.

MCF uses a search tree to find the lowest possible maximum regret. An example search tree is shown in Figure 1 where some coalition structure CS where $n = 3$ and we are concerned with a possible defection of $p = \{1, 2\}$. There will



A : $J = \mathbb{A}_1 \times \mathbb{A}_2 \times \mathbb{A}_3$
$\mathbb{J}_p = \{\mathbb{A}_1 \times \mathbb{A}_2\}$
B : $J = \mathbb{A}^{left}_1 \times \mathbb{A}_2 \times \mathbb{A}_3$
$\mathbb{J}_p = \{\mathbb{A}_1 \times \mathbb{A}_2\}$
C : $J = \mathbb{A}^{right}_1 \times \mathbb{A}_2 \times \mathbb{A}_3$
$\mathbb{J}_p = \{\mathbb{A}_1 \times \mathbb{A}_2\}$
D : $J = \mathbb{A}^{right}_1 \times \mathbb{A}_2 \times \mathbb{A}_3$
$\mathbb{J}_p = \{\mathbb{A}_1 \times A_2, (A_1, A'_2)\}$

Figure 1: *An example of a search tree for the coalition structure CS against a possible defection by agents 1 and 2. Nodes* **B** *and* **C** *were created using an action branch, as shown in Algorithm 1, on* **A**. *Node* **D** *was created using defection pruning, as shown in Algorithm 2, on node* **C**.

Algorithm 1 Action Branching Algorithm

```
Choose node $(J, \mathbb{J}_p)$ to split
Choose agent $i$ and split $J_i$ into $J^{right}_i$ and $J^{left}_i$
Create child nodes $(J^{right}_i \times J_{j \neq i}, \mathbb{J}_p)$ and $(J^{left}_i \times J_{j \neq i}, \mathbb{J}_p)$
Use defection pruning on both child nodes
```

be one search tree for each possible coalition structure. Each node is identified by the tuple $(J, \mathbb{J}_p)$. The root node, Node **A** in Figure 1 is $(\mathbb{A}, \{\mathbb{A}_p\})$. Our first method for reducing the maximum regret, an action branch, shown in Algorithm 1, simply splits J into two smaller joint partial actions. For example, in Figure 1, Nodes **B** and **C** are created by an action branch on the root node, **A**.

There are many possible heuristics that can be used to choose which node to split and how to split it. We used a simple greedy heuristic which always branched the minimax node and split a joint partial action by the partial action corresponding to the the greatest utility range over all agents. In a few cases, with this approach our algorithm would become stuck and additional iterations of our algorithm would not decrease the minimax regret. We identified such cases when the minimax regret had not decreased after a certain number of iterations. Then we would branch on the node with the second lowest maximum regret.

After each action branch, we use defection pruning on both child nodes to remove infeasible joint actions from $\mathbb{J}_p$. Consider, for example, Node **C** in Figure 1 where $\mathbb{J}_p = \{\mathbb{A}_1 \times \mathbb{A}_2\}$. Suppose that

$$(A'_1, A'_2) = \arg \max_{(A_1, A_2) \in \{\mathbb{A}_1 \times \mathbb{A}_2\}} MR(J, \{(A_1, A_2)\}),$$

that is, the joint action (A'_1, A'_2) maximizes $MR(J, \mathbb{J}_p)$. But if either agent 1 or agent 2 objects to (A'_1, A'_2), then it will not result in a feasible defection. This means that $MR(J, \mathbb{J}_p)$ may be an overestimation. If this is the case, we can lower $MR(J, \mathbb{J}_p)$ by removing (A'_1, A'_2) from $\mathbb{J}_p$. The first step in defection pruning is to decompose $\mathbb{J}_p$ in such a way that (A'_1, A'_2) is isolated. For example, if $\mathbb{A}_1 = \{A_1, A'_1\}$ and $\mathbb{A}_2 = \{A_2, A'_2\}$, then we can decompose $\mathbb{J}_p$ as $\{\mathbb{A}_1 \times A_2, (A_1, A'_2), (A'_1, A'_2)\}$.

Algorithm 2 Defection Pruning Algorithm

```
Require: Node $(J, \mathbb{J}_p)$
  loop
    $A_p^w = \max_{J_p \in \mathbb{J}_p} \arg\max_{A_p \in J_p} MR(J, \{(A)\})$
    Split $\mathbb{J}_p$ into $(A_p^w) \cup \mathbb{J}_p'$
    if $\min_i \max_{A \in J} PR(i, A, A_p^w) < 0$ then
      Remove $A_p^w$ from $\mathbb{J}_p$ {There is at least one agent who will always object to $A_p^w$}
    else
      Break
    end if
  end loop
```

If (A_1', A_2') is not feasible, we remove it from $\mathbb{J}_p$ (as demonstrated in Figure 1) and repeat this process until no more joint partial actions can be pruned.

After we have split a node and pruned both of its children, we then search for a node which minimizes the maximum regret. This node is called the *minimax node* and the corresponding joint partial action offers the lowest maximum regret found so far for the given coalition structure. Since splitting and pruning will never increase the minimax regret, we only need to search through the leaf nodes to find the minimax node. The overall minimax regret is the lowest minimax regret over all coalition structures. The corresponding coalition structure and joint partial action give the best implementation found so far. Since we can stop MCF after any iteration and go with the best implementation yet found, MCF is an anytime algorithm.

4 Experimental Results

We next present our experimental results. While the theory presented in the previous section allows for arbitrary coalitions, for our experiments we wanted to consider scenarios that could be plausible in real-world examples. In the real world, many possible coalitions may be unable to form: for example, communication infrastructure limitations may limit the maximum size of coalitions or geographical constraints may prevent specific agents from working together. To model such situations, we rely on the language for *constrained coalition formation* provided by Rahwan *et. al.* [9]. The basic form of this language models constraints on the individual coalitions, as opposed to constraints over the whole coalition structure. We used constrained coalition formation to create tractable test cases where only coalitions of a specific size are allowed to form.

To demonstrate the difficulty of finding the ϵ-core, we began with a simulation consisting of 4 agents, each with 5 possible actions, and only coalitions of size 2 allowed. For each coalition structure, there are 4 possible defecting coalitions (it is possible for different defections to result in the same coalition structure); these settings result in 1875 possible implementations and for each implementation, 100 possible defecting implementations. For this scenario, as well as all subsequent ones, we tested 100 cases to help ensure statistical significance of the results. Due to a lack of alternative algorithms to compare ours against, our benchmark algorithm was an exhaustive search (for brevity, the details of the exhaustive search are omitted). Figure 2 shows the ϵ-core versus the median number of iterations of the exhaustive search. Note the logarithmic x axis. The results, as well as subsequent ones, showed a heavy-tailed distribution since the difficult test cases tended to be especially difficult. As a result, all of our figures show the median instead of the mean since the median is more robust against outliers. Our error ranges were calculated using *median absolute deviation*, the median absolute difference from the median, which is also robust against outliers. Even in this simple setting, the exhaustive search takes, on average, almost 8000 iterations to find an implementation in the core. We also see that initially ϵ decreases quickly, while later on the decrease is much slower. This slowing down is due to the nature of random sampling; the smaller ϵ is the more iterations are needed to decrease it. For this initial simulation, as well as the subsequent ones, for roughly 1% to 2% of the test cases, the core was empty. Given the low percentages, these test cases were ignored.

We next ran MCF in the same setting. Since the majority of the work in our algorithm is done by the pruning, we counted the number of iterations for our algorithm by the number of iterations of the defection pruning method. We identified MCF as being stuck if the minimax regret had not decreased after 10 iterations. The results are also shown in Figure 2. As with the exhaustive search, with our algorithm the minimax regret

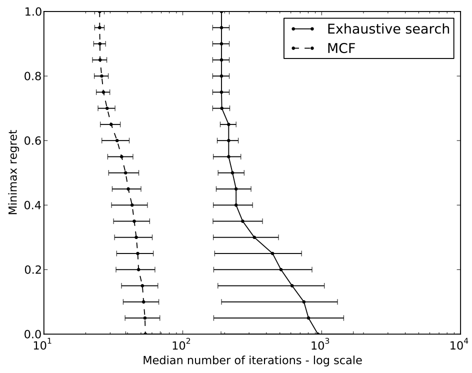


Figure 2: *The median number of iterations to reach a given minimax regret or ϵ value for both an exhaustive search and our MCF algorithm.*

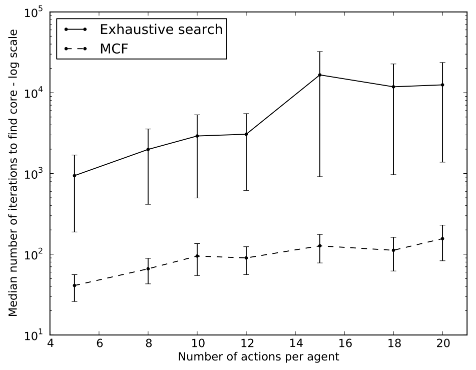


Figure 3: *The median number of iterations to find an implementation in the core as the number of actions per agent varies for both an exhaustive search and our MCF algorithm.*

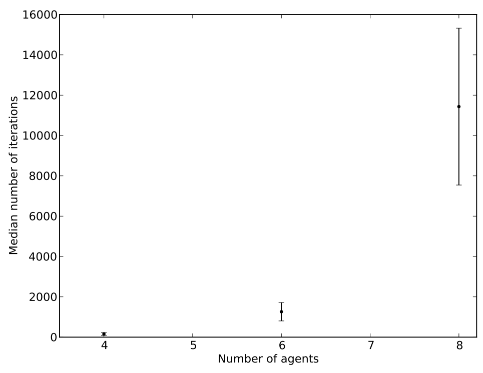


Figure 4: *The median number of iterations to find an implementation in the ϵ-core for $\epsilon = 0.05$ as the number of agents varies using the MCF algorithm.*

initially decreases very quickly but with additional iterations the rate of decrease slows. This is because as joint partial actions get smaller, the range of utility values they give gets smaller. Thus, action branches on smaller joint partial actions will produce child nodes with similar maximum regret values, which slows the decrease in the maximum regret.

Our algorithm provides a substantial increase in efficiency over the exhaustive search. For larger values of ϵ, our algorithm can find an implementation in the ϵ-core 8 times faster. This difference only increases for smaller values of ϵ: on average, our algorithm finds an implementation in the core 23 times faster than the exhaustive search.

We next examined how both of these algorithms fared as we increased the number of actions per agent. Figure 3 shows the median number of iterations for both algorithms to find the core versus the number of actions per agent. Note the log axis on the y axis. While the average number of iterations increases for both algorithms, the number of iterations for our algorithm remains significantly below that needed for an exhaustive search. More importantly, the difference in performance between the two algorithms increases. While our algorithm is initially 23 times faster, at its best it is over 130 times faster. This shows that our algorithm is better able to handle an increase in complexity.

Finally, we examined how our algorithm performed as we increased the number of agents. We examined a setting where coalitions were only of size 2 and each agent had 20 possible actions. Figure 4 shows the median number of iterations for MCF to find the ϵ-core for ϵ=0.05 versus the number of agents. We see that increasing from 6 to 8 agents results in a substantial increase in the number of iterations. This is due mostly to the increase in the number of possible coalition structures. With 6 agents, we have 15 possible coalition structures which gives an average of 84 iterations per coalition structure. With 8 agents, we have 105 possible coalition structures giving an average of 109 iterations per coalition structure. Thus, for each individual coalition structure, our algorithm scales very well despite the number of joint actions increasing by a factor of 400. With 4 agents, our algorithm was unsuccessful 2% of the time, with 6 agents 5% of the time and with 8 agents 20% of the time. Our algorithm is thus able to solve the vast majority of the test cases with 8 agents efficiently despite having over $2.6 \cdot 10^{12}$ possible implementations and over 11,000 possible defecting implementations.

5 Conclusions

The goal of this paper was to help a group of agents coordinate in such a way that each agent maximized their own utility by efficiently finding an implementation in the ϵ-core. Our algorithm is an anytime algorithm; if finding the ϵ-core for a specific ϵ turned out to be too time-consuming, our algorithm can terminate and return the best ϵ found so far. For any ϵ such that the ϵ-core is nonempty, our algorithm will eventually find an implementation in that ϵ-core. We achieved these goals by minimizing regret and adapting methods from preference elicitation.

There are many interesting possibilities for expanding on this work. One goal is developing better heuristics for helping to select and split nodes during an action branch. Another goal is dealing with the quickly growing number of coalition structures. Since coalition structures may have many coalitions in common, there may be possibilities for increasing efficiencies. Building on the work done on constrained coalition formation would allow our algorithm to scale efficiently with even more complex coalitions [9]. Finally, we would like to generalize our setting by examining other utility independence models such as *generalized additive independence* or the model proposed by Deng and Papadimitriou [1, 2].

REFERENCES

[1] D. Braziunas and C. Boutilier. Minimax Regret-based Elicitation of Generalized Additive Utilities In Proceedings of UAI '07, Vancouver, 2007.

[2] X. Deng and C. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, pages 257–266, 1994.

[3] K. Larson and T. Sandholm. Anytime coalition structure generation: An average case study. *Journal of Experimental and Theoretical AI*, 11:1–20, 2000.

[4] S. Mathur, L. Sankar, and N.B. Mandayam. Coalitions in cooperative wireless network. *IEEE Journal on Selected Areas in Communications*, 2008.

[5] R.-C. Mihailescu, M. Vasirani, and S. Ossowski. Dynamic coalitional formation and adaptation for virtual power stations in smart grids. In Proceedings of ATES '11, Taipei, 2011.

[6] M. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[7] A. Predtetchinski and P. Herings. A necessary and sufficient condition for nonemptiness of the core of a nontransferable utility game. *Journal of economic theory*, 116:84–92, 2004.

[8] T. Rahwan and N. R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In Proceedings ofAAMAS '08, Estoril, 2008.

[9] T. Rahwan, T. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N.R. Jennings. Constrained coalition formation. In Proceedings of AAAI '11, San Francisco, 2011.

[10] C. Renois. Haiti faces NGO quandary, 2011.