Learning Cooperative Games

Maria-Florina Balcan and Ariel D. Procaccia and Yair Zick
Carnegie Mellon University
ninamf,arielpro,yairzick@cs.cmu.edu

Abstract

This paper explores a PAC (probably approximately correct) learning model in cooperative games. Specifically, we are given \( m \) random samples of coalitions and their values, taken from some unknown cooperative game; can we predict the values of unseen coalitions? We study the PAC learnability of several well-known classes of cooperative games, such as network flow games, threshold task games, and induced subgraph games. We also establish a novel connection between PAC learnability and core stability: for games that are efficiently learnable, it is possible to find payoff divisions that are likely to be stable using a polynomial number of samples.

1 Introduction

Cooperative game theory studies the following model. We are given a set of players \( N = \{1, \ldots, n\} \), and \( v : 2^N \rightarrow \mathbb{R} \) is a function assigning a value to every subset (also referred to as a coalition) \( S \subseteq N \).

The game-theoretic literature generally focuses on revenue division: suppose that players have formed the coalition \( N \), they must now divide the revenue \( v(N) \) among themselves in some reasonable manner. However, all of the standard solution concepts for cooperative games require intimate knowledge of the structure of the underlying coalitional interactions. For example, suppose that a department head wishes to divide company bonuses among her employees in a canonically stable manner using the core — a division such each coalition is paid (in total) at least its value. In order to do so, she must know the value that would have been generated by every single subset of her staff. How would she obtain all this information?

Indeed, it is the authors’ opinion that the information required in order to compute cooperative solution concepts (much more than computational complexity) is a major obstacle to their widespread implementation.

Let us therefore relax our requirements. Instead of querying every single coalition value, we would like to elicit the underlying structure of coalitional interactions using a sample of \( m \) evaluations of \( v \) on subsets of \( N \). To be more specific, let us focus on the most common learning-theoretic model: the probably approximately correct (PAC) model [Kearns and Vazirani, 1994]. Briefly, the PAC model studies the following problem: we are given a set of points \( x_1, \ldots, x_m \in \mathbb{R}^n \) and their values \( y_1, \ldots, y_m \). There is some function \( f \) that generated these values, but it is not known to us. We are interested in finding a function \( f^* \) that, given that \( x_1, \ldots, x_m \) were independently sampled from some distribution \( D \), is very likely (‘probably’) to agree with \( f \) on most (‘approximately’) points sampled from the same distribution.

Procaccia and Rosenschein [2006] provide some preliminary results on PAC learning cooperative games, focusing on simple games (this is a technical term, not an opinion!) — where \( v(S) \in \{0, 1\} \) for every \( S \subseteq N \). Their results are mostly negative, showing that simple games require an exponential number of samples in order to be properly PAC learned (with the exception of the trivial class of unanimity games). However, the decade following the publication of their work has seen an explosive growth in the number of well-understood classes of cooperative games, as well as a better understanding of the computational difficulties one faces when computing cooperative solution concepts. This is where our work comes in.

1.1 Our Contribution

We revisit the connection between learning theory and cooperative games, greatly expanding on the results of Procaccia and Rosenschein [2006].

In Section 3, we introduce a novel relaxation of the core: it is likely (but, in contrast to the classic core, not certain) that a coalition cannot improve its payoff by working alone. We show that if a game is learnable, then likely core outcomes can also be learned (in that case we say the game is PAC stabilizable). This result justifies our focus on learning the values of coalitions, by relating this task to the ultimate goal of finding “good” outcomes. Interestingly, we also prove that monotone simple games are efficiently PAC stabilizable even though they are not efficiently PAC learnable.

Motivated by the foregoing connection, in Section 4 we ask whether or not classes of games are efficiently learnable, that is, whether there is a polynomial-time algorithm that receives a polynomial number of samples, and outputs an accurate hypothesis with high confidence. Our main results are that network flow games [Maschler et al., 2013, Chapter 17.9] are efficiently learnable with path queries (but not in general), and
so are threshold task games [Chalkiadakis et al., 2010], and induced subgraph games [Deng and Papadimitriou, 1994]. We also study k-vector weighted voting games [Elkind et al., 2009], MC nets [Ieong and Shoham, 2005], and coalitional skill games [Bachrach and Rosenschein, 2008].

1.2 Related Work
Aside from the closely related work of Procaccia and Rosenschein [2006], there are several papers that study coalitional stability in uncertain environments. Chalkiadakis and Boutilier [2004] and Li and Conitzer [2015] assume that coalition values are drawn from some unknown distribution, and we observe noisy estimates of the values. However, both papers assume full access to the cooperative game, whereas we assume that m independent samples are observed. Other works study coalitional uncertainty: coalition values are known, but agent participation is uncertain due to failures [Bachrach et al., 2012a; 2012b; Bachrach and Shah, 2013].

Our work is also related to papers on eliciting and learning combinatorial valuation functions [Zinkevich et al., 2003; Lahaie and Parkes, 2004; Lahaie et al., 2005; Balcan and Harvey, 2011; Balcan et al., 2012; Badanidiyuru et al., 2012]. A player’s valuation function in a combinatorial auction is similar to a cooperative game: it assigns a value to every subset of items (instead of every subset of players). This connection allows us to draw on some of the insights from these papers. For example, as we explain below, learnability results for XOS valuations [Balcan et al., 2012] informed our results on network flow games.

2 Preliminaries
2.1 Cooperative Games
A cooperative game is a tuple G = (N, v), where N = {1, ..., n} is a set of players, and v : 2N → R is called the characteristic function of G. When the player set N is obvious, we will identify G with the characteristic function v, referring to v as the game. A game G is called simple if v(S) ∈ {0, 1} for all S ⊆ N; G is called monotone if v(S) ≤ v(T) whenever S ⊆ T. One of the main objectives in cooperative games is finding “good” ways of dividing revenue: it is assumed that players have generated the revenue v(N), and must find a way of splitting it. An imputation for G is a vector x ∈ Rn that satisfies efficiency: ∑ni=1 xi = v(N), and individual rationality: xi ≥ v({i}) for every i ∈ N. The set of imputations, denoted I(G), is the set of all possible “reasonable” payoff divisions among the players. Given a game G, the core of G is given by

Core(G) = {x ∈ I(G) | ∀S ⊆ N : x(S) ≥ v(S)}.

The core is the set of all stable imputations: no subset of players S can deviate from an imputation x ∈ Core(G) while guaranteeing that every i ∈ S receives at least as much as it gets under x.

2.2 PAC Learning
We provide a brief overview of the PAC learning model; for a far more detailed exposition, see [Kearns and Vazirani, 1994; Shasha, 2009]. PAC learning pertains to the study of the following problem: we are interested in learning an unknown function f : 2N → R. In order to estimate the value of f, we are given m samples (S1, v1), ..., (Sm, vm), where vj = f(Sj). Without any additional information, one could make arbitrary guesses as to the possible identity of f; for example, we could very well guess that f*(Sj) = vj for all j ∈ [m], and 0 everywhere else. Thus, in order to obtain meaningful results, we must make further assumptions. First, we restrict f to be a function from a certain class of functions C: for example, we may know that f is a linear function of the form f(S) = ∑i∈S wi, but we do not know the values w1, ..., wn.

Second, we assume that there is some distribution D over 2N such that S1, ..., Sm were sampled i.i.d. from D. Finally, we require that the estimate that we provide has low error over sets sampled from D.

Formally, we are given a function v : 2N → R+, and two values ε > 0 (the accuracy parameter) and δ > 0 (the confidence parameter). An algorithm A takes as input ε, δ and m samples, (S1, v(S1)), ..., (Sm, v(Sm)), taken i.i.d. from a distribution D. We say that A can properly learn a function f ∈ C from a class of functions C (C is sometimes referred to as the hypothesis class), if by observing m samples — where m can depend only on n (the representation size), 1/δ, and 1/ε — it outputs a function f* ∈ C such that with probability at least 1 − δ,

PrS∼D[f(S) ≠ f*(S)] < ε.

The confidence parameter δ indicates that there is some chance that A will output a bad guess (intuitively, that the m samples given to the algorithm are not representative of the overall behavior of f over the distribution D), but this is unlikely. The accuracy parameter ε indicates that for most sets sampled from D, f* will correctly guess the value of S.

Note that the algorithm A does not know D; that is, the only thing required for PAC learnability to hold is that the input samples independent, and that future observations are also sampled from D. In this paper, we only discuss proper learning; that is, learning a function f ∈ C using only functions from C.

We say that a finite class of functions C is efficiently PAC learnable if the PAC learning algorithm described above runs in polynomial time, and its sample complexity m is polynomial in n, 1/δ, and log 1/ε.

Efficient PAC learnability can be established via the existence of consistent algorithms. Given a class of functions C from 2N to R, suppose that there is some efficient algorithm A that for any set of samples (Sj, v(Sj))j∈[m] is able to output a function f* ∈ C such that f*(Sj) = vj for all j ∈ [m], or determine that no such function exists. Then A is an algorithm that can efficiently PAC learn C given m ≥ 1/δ log |C| samples. Conversely, if no efficient algorithm exists, then f cannot be efficiently PAC learned from C.

To conclude, in order for a class C to be efficiently PAC learnable, we must have polynomial bounds on the sample complexity — i.e. the number of samples required in order to obtain a good estimate of functions in C — as well as a polynomial algorithm that finds a function in C which is a perfect
match for the samples. We observe that in many of the settings described in this paper, the sample complexity is low, but finding consistent functions in \(C\) is computationally intractable (it would entail that \(P = NP\) or that \(NP = RP\)). In contrast, the result of Procaccia and Rosenschein [2006] establishes lower bounds on the sample complexity for PAC learning monotone simple games, but there exists a simple algorithm that outputs a hypothesis consistent with any sample.

When the hypothesis class \(C\) is finite, it suffices to show that \(\log |C|\) is bounded by a polynomial in order to establish a polynomial sample complexity. In the case of an infinite class of hypotheses, this bound becomes meaningless, and other measures must be used. When learning a function that takes values in \(\{0, 1\}\), the VC dimension [Kearns and Vazirani, 1994] captures the learnability of \(C\). Given a class \(C\), and a list \(S\) of \(m\) sets \(S_1, \ldots, S_m\), we say that \(C\) shatters \(S\) if for every \(b \in \{0, 1\}^m\) there exists some \(v_b \in C\) such that \(v(b)(S_j) = b_j\) for all \(j\). We write

\[
V\text{Cdim}(C) = \max\{m \mid \exists S, |S| = m, C \text{ can shatter } S\}.
\]

When learning hypotheses that output real numbers (as opposed to functions that take on values in \(\{0, 1\}\)), the notion of pseudo dimension is used in order to bound the complexity of a function class. Given a sample \(S\) of \(m\) sets \(S = S_1, \ldots, S_m \subseteq N\), we say that a class \(C\) shatters \(S\) if for every \(i \in \{0, 1\}^m\) there exists some \(v_i \in C\) such that \(v_i(b)(S_j) \geq r_j\) if \(b_j = 1\), and \(v_i(b)(S_j) < r_j\) if \(b_j = 0\). We write

\[
P\text{dim}(C) = \max\{m \mid \exists S : |S| = m, C \text{ can shatter } S\}.
\]

It is known [Anthony and Bartlett, 2009] that if \(P\text{dim}(C)\) is polynomial, then the sample complexity of \(C\) is polynomial as well.

### 3 PAC Stability

In the context of cooperative games, one could think of PAC learning as the following process. A central authority wishes to find a stable outcome, but lacks information about agents’ abilities. It solicits the independent valuations of \(m\) subsets of agents, and outputs an outcome that, with probability \(1 - \delta\), is likely to be stable against any unknown valuations.

More formally, given \(\varepsilon, \delta \in (0, 1)\), we say that an imputation \(x \in I(G)\) is \(\varepsilon, \delta\)-probably stable under \(D\) if

\[
\Pr_{S \sim D}[|x(S) - v(S)| \geq \varepsilon] \leq \delta.
\]

An algorithm \(A\) can PAC stabilize a class of functions \(C\) from \(2^N\) to \(\mathbb{R}_+\) if, given \(\varepsilon, \delta \in (0, 1)\), and \(m\) i.i.d. samples \((S_1, v(S_1)), \ldots, (S_m, v(S_m))\) of some \(v \in C\), with probability \(1 - \delta\), \(A\) outputs an outcome \(x\) that is \(\varepsilon\), \(\delta\)-probably stable under \(D\), or outputs that the core is empty. If \(m\) is polynomial in \(n\), \(\varepsilon\) and \(\frac{1}{\delta}\), and \(A\) runs in polynomial time, we say that \(C\) is efficiently PAC stabilizable.

There is an immediate relation between PAC learnable and PAC stabilizable function classes:

**Proposition 3.1.** Let \(v, v^* : 2^N \rightarrow \mathbb{R}_+\) be two functions such that \(\Pr_{S \sim D}[v(S) = v^*(S)] \geq 1 - \varepsilon\), if \(x \in \text{Core}(v^*)\), then \(x\) is \(\varepsilon\)-probably stable under \(D\) for \(v\).

**Proof.** Suppose that \(v^*\) is a function that satisfies

\[
\Pr_{S \sim D}[v(S) = v^*(S)] \geq 1 - \varepsilon,
\]

and suppose that \(x \in \text{Core}(v^*)\). Then,

\[
\Pr_{S \sim D}[x(S) \geq v(S)] \\
\geq \Pr_{S \sim D}[x(S) \geq v(S) \mid v(S) = v^*(S)] \cdot \Pr_{S \sim D}[v(S) = v^*(S)] \\
\geq 1 \cdot (1 - \varepsilon) = 1 - \varepsilon
\]

We mention that Proposition 3.1 says nothing about the efficiency of finding probably stable outcomes. In order to find a PAC-stable outcome using a PAC learned function, it is essential that \(v\) belong to a class of functions that can be learned in polynomial time, and for which a core outcome can be found in polynomial time.

Moreover, there is an important subtlety here. Let \(G^* = (N, v^*)\) be the PAC learned hypothesis of \(G = (N, v)\). Proposition 3.1 states that if \(x \in \text{Core}(G^*)\), then the probability that \(x\) violates a core constraint in \(G\) is small. However, there are two potential risks: first, it is possible that \(\text{Core}(G^*) = \emptyset\), but \(\text{Core}(G) \neq \emptyset\). This is not a concern if the learned hypothesis is guaranteed to have a non-empty core, or if \(\text{Core}(G^*)\) contains \(\text{Core}(G)\).

Second, even if \(\text{Core}(G^*) \neq \emptyset\), we are not guaranteed that \(x \in \text{Core}(G^*)\) is a valid payoff division for \(G\), if \(v^*(N) \neq v(N)\). In our motivating setting, we assume that \(v(N)\) is known, so the latter is not a major concern.

These issues do not arise if \(v^*(S) \leq v(S)\) for all \(S \subseteq N\). In that case, \(\text{Core}(G^*)\) contains \(\text{Core}(G)\); thus, if the latter is non-empty, so is the former.

While it may be generally hard to find a core outcome for a cooperative game, it is easy to do so for monotone simple games, where the core has a very simple characterization (see e.g. [Maschler et al., 2013, Chapter 17]) via veto players. We say that a player \(i \in N\) is a veto player if it belongs to all winning coalitions; in other words, if \(v(S) = 1\), then \(i \in S\).

**Fact 3.2.** Let \(G = (N, v)\) be a monotone simple game, and let \(V \subseteq N\) be the set of veto players for \(G\). If \(V = \emptyset\) then \(\text{Core}(G) = \emptyset\); otherwise, \(\text{Core}(G)\) consists of all imputations that assign a payoff of 0 to any \(i \in N \setminus V\); in particular, if \(x \in \text{Core}(G)\) then \(\sum_{i \in V} x_i = 1\).

Leveraging Fact 3.2, we obtain the following result.

**Theorem 3.3.** The class of monotone simple games is efficiently PAC stabilizable.

The theorem is especially interesting because the class of monotone simple games is not efficiently PAC learnable [Procaccia and Rosenschein, 2006].

**Proof of Theorem 3.3.** First, if our samples contain two disjoint winning coalitions, then the core of \(G\) is surely empty, and we report that the core is empty. Thus, let us assume that all winning coalitions intersect. This implies that our input samples correspond to an input from a unanimity game [Maschler et al., 2013].
A unanimity game $\mathcal{U}_V = \langle N, u_V \rangle$, has $u_V(S) = 1$ if and only if $V \subseteq S$. According to Fact 3.2,

$$\text{Core}(\mathcal{G}) = \text{Core}(\mathcal{U}_V) = \left\{ x \in \mathbb{R}^n_+ \mid \sum_{i \in V} x_i = 1 \right\}.$$ 

Thus, finding a probably stable outcome for $\mathcal{G}$ amounts to finding a probably stable outcome for $\mathcal{U}_V$. Procaccia and Rosenschein [2006] show that unanimity games can be efficiently PAC learned; thus, according to Proposition 3.1, some of our computational in- 

ative games. We show when can one leverage efficient PAC 

learnability and PAC stability in common classes of cooper-

able to PAC stabilize it. In what follows, we explore both PAC 

checking whether $\text{Core} \setminus \text{Core}_{\mathcal{G}}$.

4.1 Network Flow Games

A network flow game is given by a weighted, directed graph $\Gamma = (V, E)$, with $w : E \rightarrow \mathbb{R}_+$ being the weight function for the edges. Here, $N = E$, and $v(S) = \text{flow}(\Gamma_{S})$, where $\text{flow}$ denotes the maximum $s$-$t$ flow through $\Gamma$, where edge weights are given by $w$, and $s, t \in V$.

We begin by showing that a similar class of functions is not efficiently learnable. We define the following family of functions, called min-sum functions which are defined as follows: there exists a list of vectors $w_1, \ldots, w_m$. For every $S \subseteq N$, $f(S) = \min_{e \in \text{[m]}} w_k(S)$, where $w_k(S) = \sum_{j \in S} w_{kj}$. If $m = 1$, we say that the min-sum function is trivial. We note that Balcan et al. [2012] study the learnability of XOS valuations, where the min is replaced with a max.

Lemma 4.1. The set of non-trivial min-sum functions is not efficiently PAC learnable unless $NP = RP$.

Proof Sketch. Our proof relies on the fact that CNF formulas with more than two clauses are not efficiently PAC learnable unless $NP = RP$ [Pitt and Valiant, 1988]. The reduction shows that for any CNF formula $\phi$ and any distribution $D$ over the inputs to $\phi$ there is some min-sum function $f_\phi$ with player set $N^e$, and a distribution $D'$ over $N^e$ such that $f_\phi(S) = 1$ if and only if $\phi(\text{var}(S)) = 1$, where var$(S)$ is a variable assignment corresponding to the players chosen in $S$. Thus, efficient PAC learnability of $f_\phi$ implies the PAC learnability of $\phi$.

Theorem 4.2. Network flow functions are not efficiently learnable unless $NP = RP$.

Proof. Our proof reduces the problem of learning min-sum functions to the problem of learning network flow functions. Given a min-sum target function $\bar{f}$, defined by $w_1, \ldots, w_k$, and a distribution $D$ over samples of $N$, we construct the di-

rected graph $\Gamma = (V, E)$ as follows.

For every weight vector $w_{\ell} = (w_{1,\ell}, \ldots, w_{k,\ell})$, we define vertices $\ell, \ell + 1, \ldots, n$ and edges from $\ell$ to $\ell + 1$, where the capacity of the edge $e_{\ell,\ell+1}$ is $w_{\ell}$. Finally, we denote the vertex $k + 1$ as the target $t$, and the vertex $1$ as the source $s$. Given a set $S \subseteq N$, we write $E_S = \{e_{\ell,\ell+1} \mid \ell \in [k], i \in S \}$. We observe that the flow from $s$ to $t$ in the constructed graph using only edges in $E_S$ equals $f(S)$; in other words, $\text{flow}(\Gamma_{E_S}) = f(S)$ for all $S \subseteq N$. Now, given a probability distribution $D$ over $2^N$, we define a probability distribution over $E$ as follows: $\text{Pr}_D[E_S] = \text{Pr}_D[S]$ for all $S \subseteq N$, and is $0$ for all other subsets of $E$.

We conclude that efficiently PAC learning $\text{flow}_E$ under the distribution $D'$ is equivalent to PAC learning $f$, which cannot be done efficiently by Lemma 4.1.

Learning network flow games is thus generally a difficult task. In order to obtain some notions of tractability, let us study a variant of network flow games, where we limit our attention to sets that constitute paths in $\Gamma$. In other words, we limit our attention to distributions $D$ such that if $D$ assigns some positive probability to a set $S$, then $S$ must be an $s$-$t$ path in $\Gamma$. One natural example of such a distribution is the following: we make graph queries on $\Gamma$ by performing a random walk on $\Gamma$ until we either reach $t$ or have traversed more than $|V|$ vertices.

Given a directed path $\pi = (w_1, \ldots, w_k)$, we let $w(\pi)$ be the flow that can pass through $\pi$; that is, $w(\pi) = \min_{e \in \pi} w_e$.

Theorem 4.3. Network flow games are efficiently PAC learnable if we limit $D$ to be a distribution over paths in $\Gamma$.

Proof. Given an input $((p_1, v_1), \ldots, (p_m, v_m))$, we let $\bar{w}_e = \max_{e \in [\pi]} v_j$. We observe that the weights $(\bar{w}_e)_{e \in E}$ are such that $w(p_j) = w(p_j)$ for all $j \in [m]$. This is because for any $e \in p_j$, $\bar{w}_e \geq v_j$, so $\min_{e \in p_j} \bar{w}_e \geq v_j$. On the other hand, $w_e \leq \bar{w}_e$ for all $e \in E$, since $w_e \geq v_j$ for all $v_j$ such that $e \in p_j$, and in particular $w_e \geq \max_{j \in [\pi]} v_j$ is $\bar{w}_e$. Thus, $\min_{e \in p_j} \bar{w}_e \leq \min_{e \in \pi} w_e = v_j$. In other words, by simply taking edge weights to be the maximum flow that passes through them in the samples, we obtain a graph that is consistent with the sample in polynomial time.
Now, suppose that the set of weights on the edges of the graph according to the target weights \( w_e \) is given by \( \{a_1, \ldots, a_k\} \), where \( k \leq n \). Then there are \( (k+1)^n \leq (n+1)^n \) possible ways of assigning values \( (\bar{w}_e)_{e \in E} \) to the edges in \( E \). In other words, there are at most \( (n+1)^n \) possible hypotheses to test. Thus, in order to \((\varepsilon, \delta)\)-learn \( (w_e)_{e \in E} \), where the hypothesis class \( C \) is of size \( \leq (n+1)^n \), we need a number of samples polynomial in \( \frac{1}{\varepsilon}, \log \frac{1}{\delta} \) and \( \log |C| \in O(n \log n) \).

**Corollary 4.4.** Network flow games are efficiently PAC stabilizable if we limit \( D \) to be a distribution over paths in \( \Gamma \).

**Proof.** Theorem 4.3 establishes that if one is limited to path queries, network flow games are efficiently PAC learnable. In order to prove PAC stabilizability, we need to show that core outcomes can be found in polynomial time.

It is well-known that computing core outcomes in network flow games is easy: given an edge set \( C \subseteq E \) that is a minimum cut in the graph, pay each \( e \in C \) an amount equal to the flow that passes through it. We conclude that finding probably stable outcomes for network flow games can be done in polynomial time if we limit ourselves to path queries on \( \Gamma \).

**4.2 Threshold Task Games**

In **Threshold tasks games (TTG)** each player \( i \in N \) has an integer weight \( w_i \); there is a finite list of tasks \( T \), and each task \( t \in T \) is associated with a threshold \( q(t) \) and a payoff \( V(t) \). Given a coalition \( S \subseteq N \), we let \( T_S = \{t \in T \mid q(t) \leq w(S)\} \). The value of \( S \) is given by \( v(S) = \max_{t \in T_S} V(t) \). In other words, \( v(S) \) is the value of the most valuable task that \( S \) can accomplish. Weighted voting games (WVGs) are the special case of TTGs with a single task, whose value is 1; that is, they describe linear classifiers.

Without loss of generality we can assume that all tasks in \( T \) have strictly monotone thresholds and values: if \( q(t) > q(t') \) then \( V(t) > V(t') \). Otherwise, we will have some redundant tasks. For ease of exposition, we assume that there is some task whose value is 0 and whose threshold is 0. Let \( C_{ttg}(Q) \) be the class of \( k \)-TTGs for which the set of task values \( Q \subseteq \mathbb{R} \) of size \( k \). The first step of our proof is to show that \( C_{ttg}(Q) \) is PAC learnable.

**Lemma 4.5.** The class \( C_{ttg}(Q) \) is PAC learnable.

**Proof Sketch.** In order to show this, we first bound the sample complexity of \( C_{ttg}(Q) \). We claim that \( Pdim(C_{ttg}(Q)) < (k+1)(n+2) \). The proof relies on the fact that the VC dimension of linear functions is \( n+1 \).

Assume by contradiction that there exists some \( S \in S \) of size \( L \), where \( L = (k+1)(n+2) \), and some values \( r_1, \ldots, r_L \in \mathbb{R}^+ \) such that for all \( b \in \{0, 1\}^L \) there is some TTG \( f_b \in C_{ttg}(Q) \) such that \( f_b(S_b) \geq r_i \) when \( b_i = 0 \), and \( f_b(S_b) \leq r_i \) when \( b_i = 0 \). We assume that \( 0 \leq r_1 \leq \cdots \leq r_L \). Let us write the task set to \( T = \{t_1, \ldots, t_k\} \), each with a value \( V(t_i) \in Q \); we order our tasks by increasing value. Now, by the pigeonhole principle, there exists some task \( t_i \) and some \( j \) such that \( r_j, \ldots, r_{j'} \in \{V(t_i), V(t_{i+1})\} \). In particular, if we write \( S^* = \{S_j, \ldots, S_{j'}\} \), then for every \( b \in \{0, 1\}^L \), there is some \( f_b \in C_{ttg}(Q) \) (defined by an agent weight vector \( w_b \), and task thresholds \( T_1^b, \ldots, T_k^b \)), such that for all \( S_j \in S^* \), if \( f_b(S_j) \geq r_j \) it must be that \( f_b(S_j) \geq V(t_i) \), i.e., \( w_b(S_j) > T_k^b \). If \( f_b(S_j) \leq r_j \) then \( w_b(S_j) \leq T_k^b \). Thus, \( (w_b, T_k^b) \) is a set of \( n \)-dimensional linear classifiers that is able to shatter a set of size \( n+2 \), a contradiction. To conclude, \( Pdim(C_{ttg}(Q)) \leq (k+1)(n+2) \), which implies that the sample complexity for PAC learning TTGs is polynomial.

It is easy to construct an efficient algorithm that is consistent with any sample from \( C_{ttg}(Q) \) via linear programming (details omitted).

The following lemma (proof omitted) shows that if we take a sufficient number of samples, a game \( v \in C_{ttg} \) can be PAC approximated by a game \( \bar{v} \in C_{ttg}(Q) \), where \( Q \) are the observed values of \( v \).

**Lemma 4.6.** Given \( m \geq (k+1)\frac{1}{\varepsilon} \) independent samples \( (v(S_1), (S_1, v(S_1)), \ldots, (S_m, v(S_m))) \) from \( v \in C_{ttg} \); let \( Q = \bigcup_{j=1}^m \{v(S_j)\} \). The event \( Pr_{S \sim D}[\bar{v}(S) \not\in Q] < \varepsilon \) occurs with probability at least \( 1 - \delta \).

**Theorem 4.7.** Let \( C_{ttg} \) be the class of \( k \)-TTGs; then \( C_{ttg} \) is PAC learnable.

**Proof Sketch.** Let \( (S_1, v(S_1)), \ldots, (S_m, v(S_m)) \) be our set of samples. According to Lemma 4.6, we can choose \( m \) such that with probability \( \geq 1 - \frac{\delta}{2} \), \( Pr_{S \sim D}[\bar{v}(S) \not\in Q] < \frac{\delta}{2} \). We let \( \bar{v} \) be the TTG \( v \) with the set of tasks reduced to \( Q \); that is \( \bar{v}(S) = v(S) \) if \( v(S) \in Q \), and the value of the best task that \( S \) can complete whose value is in \( Q \) otherwise. Thus, we can pretend that our input is from \( \bar{v} \in C_{ttg}(Q) \). According to Lemma 4.5, if \( m \) is sufficiently large, then with probability \( \geq 1 - \frac{\delta}{2} \), we will output some \( v^* \in C_{ttg}(Q) \) such that \( Pr_{S \sim D}[\bar{v}(S) = v^*(S)] \geq 1 - \frac{\delta}{2} \). Thus, with probability \( \geq 1 - \delta \), we have that both \( Pr_{S \sim D}[\bar{v}(S) = v^*(S)] \geq 1 - \frac{\delta}{2} \) and \( Pr_{S \sim D}[\bar{v}(S) = v(S)] \geq 1 - \frac{\delta}{2} \). We claim that \( v^* \) PAC approximates \( v \). Indeed, it is easy to verify that \( Pr_{S \sim D}[\bar{v}(S) \not\in v^*(S)] < \varepsilon \). Thus, \( \bar{v} \) is at most \( \varepsilon \).

We observe that the output of the algorithm described in Theorem 4.7 is a TTG for which coalition values do not exceed values in the original TTG. However, this does not guarantee that we can obtain a stable outcome using this method, unless we assume that the value of the original game is known. Indeed, it is possible that the core of the original game is not empty, whereas the learned game has an empty core.

Finally, even if we are able to PAC learn and output a TTG with a non-empty core, it is not necessarily the case that a core outcome can be computed in polynomial time. This is because computing the core of a TTG is known to be NP-hard [Chalkiadakis et al., 2010], unless weights are given in unary (i.e., the bit precision is polylogarithmic).

**Corollary 4.8.** If the value \( v(N) \) is known, the class of TTGs with poly-size weights and values is PAC stabilizable.
4.3 Induced Subgraph Games

An induced subgraph game (ISG) is given by a weighted graph $\Gamma = (N,E)$, where for every pair $i,j \in N$, $w_{ij} \in \mathbb{Z}$ denotes the weight of the edge between $i$ and $j$. We let $W$ be the weighted adjacency matrix of $\Gamma$. The value of a coalition $S \subseteq N$ is given by $v(S) = \sum_{i \in S} \sum_{j \in S \setminus \{i\}} w_{ij};$ i.e. the value of a set of nodes is the weight of their induced subgraph.

**Theorem 4.9.** The class of induced subgraph games is efficiently PAC learnable.

**Proof Sketch.** Let $W$ be the (unknown) weighted adjacency matrix of $\Gamma$. Let us write $e_S$ to be the indicator vector for the set $S \in \mathbb{R}^n$. That is, the $i$-th coordinate of $e_S$ is 1 if $i \in S$, and is 0 otherwise. We observe that in an ISG, $v(S) = e_S^TW e_S$.

In other words, learning the coefficients of an ISG is equivalent to learning a linear function with $O(n^2)$ variables (one per vertex pair), which is known to have polynomial sample complexity [Anthony and Bartlett, 2009].

Now, given observations $(S_1,v_1), \ldots, (S_m,v_m)$, we need to solve a linear system with $m$ constraints (one per sample), and $O(n^2)$ variables (one per vertex pair, as above), which is solvable in polynomial time. The output of this linear optimization is guaranteed to be consistent, and since a solution exists (namely, $W$), we have a simple consistent poly-time algorithm, and conclude that the class of ISGs is efficiently PAC learnable.

It is well known that computing a core outcome for ISGs is NP-hard [Deng and Papadimitriou, 1994], unless all weights are non-negative (in which case the core is never empty). In order to ensure that we find a PAC stable outcome for the latter case, we can slightly modify the solution by searching for a non-negative solution. If a solution exists, we have obtained an outcome that is PAC stable; if not, we drop the non-negativity assumption, but are not guaranteed a poly-time algorithm for finding a core outcome, nor its existence.

4.4 Additional Classes of Cooperative Games

Before we conclude, we present a brief overview of additional results obtained for other classes of cooperative games.

**k-WVGs:** In weighted voting games (WVGs), each player $i \in N$ has an integer weight $w_i$; the weight of a coalition $S \subseteq N$ is defined as $w(S) = \sum_{i \in S} w_i$. A coalition is winning (has value 1) if $w(S) \geq q$, and has a value of 0 otherwise. Here, $q$ is a given threshold, or quota. The class of $k$-vector WVGs is a simple generalization of weighted voting games given by Elkind et al. [2009]. A $k$-vector WVG is given by $k$ WVGs: $(w_1;q_1), \ldots, (w_k;q_k)$. A set $S \subseteq N$ is winning if it is winning in every one of the $k$ WVGs.

Learning a weighted voting game is equivalent to learning a separating hyperplane, which is known to be easy [Kearns and Vazirani, 1994]. However, learning $k$-vector WVGs is equivalent to learning the intersection of $k$-hyperplanes, which is known to be NP-hard even when $k = 2$ [Alekhnovich et al., 2004; Blum and Rivest, 1992; Klivans et al., 2002]. Thus, $k$-WVGs are not efficiently PAC learnable, unless $P=NP$; however, since they are simple and monotone, they are PAC stabilizable according to Theorem 3.3.

**Coalitional Skill Games:** Coalitional Skill Games (CSGs) [Bachrach and Rosenschein, 2008] are another well-studied class of cooperative games. Here, each player $i$ has a skill-set $K_i$; additionally, there is a list of tasks $T$, each with a set of required skills $K_i$. Given a set of players $S \subseteq N$, let $K(S)$ be the set of skills that the players in $S$ have. Let $T(S)$ be the set of tasks $\{t \in T \mid K_i \subseteq K(S)\}$. The value of the set $T(S)$ can be determined by various utility models; for example, setting $v(S) = |T(S)|$, or assuming that there is some subset of tasks $T^* \subseteq T$ such that $v(S) = 1$ iff $T^* \subseteq T(S)$; the former class of CSGs is known as conjunctive task skill games (CTSGs).

PAC learnability of coalitional skill games is generally computationally hard. This holds even if we make some simplifying assumptions; for example, even if we know the set of tasks and their required skills in advance, or if we know the set of skills each player possesses, but the skills required by tasks are unknown. However, we can show that CTSGs are efficiently PAC learnable if player skills are known.

**MC-nets:** Marginal Contribution Nets (MC-nets) [Ieong and Shoham, 2005] provide compact representation for cooperative games. Briefly, an MC-net is given by a list of rules over the player set $N$, along with values. A rule is a Boolean formula $\phi_j$ over $N$, and a value $v_j$. For example, $r = x_1 \lor x_2 \lor \neg x_3 \rightarrow 7$ assigns a value of 7 to all coalitions containing players 1 and 2, but not player 3. Given a list of rules, the value of a coalition is the sum of all values of rules that apply to it. PAC learning of MC-nets can be reduced to PAC learning of DNF formulas, which is believed to be intractable [Klivans and Servedio, 2001].

5 Discussion

Our work is limited to finding outcomes that are likely to be stable for an unknown function. However, learning approximately stable outcomes is a promising research avenue as well. Such results naturally relate approximately stable outcomes — such as the $\varepsilon$ and least core [Peleg and Sudhölter, 2007], or the cost of stability [Bachrach et al., 2009] — with PMAC learning algorithms [Balcan and Harvey, 2011], which seek to approximate a target function (M stands for “mostly”) with high accuracy and confidence.

This work has focused on the core solution concept; however, learning other solution concepts is a natural extension. While some solution concepts, such as the nucleolus or the approximate core variants mentioned above, can be naturally extended to cases where only a subset of the coalitions is observed, it is less obvious how to extend solution concepts such as the Shapley value or Banzhaf power index. These concepts depend on the marginal contribution of player $i$ to coalition $S$, i.e., $v(S \cup \{i\}) - v(S)$. Under the Shapley value, we are interested in the expected marginal contribution when a permutation of the players is drawn uniformly at random, and $i$ joins previous players in the permutation. According to Banzhaf, $S$ is drawn uniformly at random from all subsets that do not include $i$. Both solution concepts are easy to approximate if we are allowed to draw coalition values from the appropriate distribution [Bachrach et al., 2010] — this is a good way to circumvent computational complexity when the
game is known. It would be interesting to understand what guarantees we obtain for arbitrary distributions.

Acknowledgments: This research is partially supported by NSF grants CCF-1451177, CCF-1422910, and CCF-1101283, CCF-1215883 and IIS-1350598, a Microsoft Research Faculty Fellowship, and Sloan Research Fellowships.

References


