Chapter 8

Computational Coalition Formation

Edith Elkind,*[*] *Talal Rahwan,*[*] *and Nicholas R. Jennings

1 Introduction

In many multiagent systems, agents can improve their performance by forming *coalitions*, i.e., pooling their efforts and resources so as to achieve the tasks at hand in a more efficient way. This holds both for *cooperative* agents, i.e., agents who share a common set of goals, and for *selfish* agents who only care about their own payoffs. For cooperative agents, to find the optimal collaboration pattern, it suffices to identify the best way of splitting agents into teams. In contrast, when the agents are selfish, we also have to specify how to distribute the gains from cooperation, since each agent needs to be incentivized to participate in the proposed solution.

In this chapter, we discuss coalition formation in multiagent systems for both selfish and cooperative agents. To deal with selfish agents, we introduce classic solution concepts of coalitional game theory that capture the notions of stability and fairness in coalition formation settings. We then give an overview of existing representation formalisms for coalitional games. For each such formalism, we discuss the complexity of computing the solution concepts defined earlier in the chapter, focusing on algorithms whose running time is polynomial in the number of agents n. In the second half of the chapter, we focus on practical approaches for finding an optimal partition of agents into teams. We present the state-of-the-art algorithms for this problem, and compare their relative strengths and weaknesses.

[*]The first two authors have contributed equally to the chapter.

1.1 Coalitional Games: A Bird's Eye View

The goal of the coalition formation process is to split the set of agents – or *players* – into disjoint teams, or *coalitions*: a partition of the set of agents into coalitions is called a *coalition structure*.[1] Once a coalition structure forms, each coalition chooses its action in a way that results in payoffs to its members. Coalitional games provide a formal model of coalition formation scenarios. They are usually classified according to two orthogonal dimensions: (1) whether agents can make payments to each other and (2) whether the payoff that a coalition obtains by choosing a particular action depends on the actions of other coalitions. We will now discuss this classification in more detail.

In some settings modeled by coalitional games, all agents have comparable utilities and can commit to monetary transfers among the members of a coalition. Whenever this is the case, we can simply assume that the coalitional action generates a single payoff, which is subsequently shared among the members of the coalition; this payoff is referred to as the *value* of this coalition. Such games are known as *transferable utility games*, or *TU games*. However, sometimes agents cannot make side payments to each other, either because their payoffs from the coalitional action are non-monetary in nature, or because there is no suitable infrastructure to transfer the money.

Example 8.1 *If several researchers from different universities write a joint paper, each researcher receives a payoff from its own university: the paper can count toward promotion or tenure, receive an internal prize, or, sometimes, be rewarded with a monetary bonus. However, these payoffs are allocated to individual researchers, and, with the exception of a bonus payment, cannot be transferred from one researcher to another.*

The settings similar to the one in Example 8.1 are modeled by assuming that each coalitional action corresponds to a vector of payoffs – one for each member of the coalition. Games represented in this manner are known as *games with non-transferable utility*, or *NTU games*.

It is important to note that in NTU settings two coalitional actions may be incomparable. For instance, consider the 2-player coalition $\{a_1, a_2\}$ that chooses between actions x and y. Suppose that whenever the players choose x, player a_1 gets a payoff of 5, whereas player a_2 gets a payoff of 1; on the other hand, if players choose y, player a_1 gets 2 and player a_2 gets 7. Obviously, player a_1 prefers x to y, even though action y has a higher total utility, whereas player a_2 prefers y to x. In contrast, in TU games, all players prefer the action(s) that result(s) in the highest sum of payoffs, as they can distribute the total payoff so

[1]Recently, games with overlapping coalitions have also been considered; see, e.g. [12].

that everyone is better off. This intracoalitional competition makes NTU games more difficult to analyze, which may explain why TU games received much more attention in the multiagent literature. We will follow this trend, and for the rest of the chapter focus on TU games only.

Now, in each of the examples considered so far, the payoffs that each coalition could attain were determined by the identities and actions of the coalition members. However, there are cases where a coalition's productivity also depends on the coalition structure that it is a part of, i.e., it may be influenced by the actions of non-members. This is the case, for instance, in market-like environments, where each coalition provides a service, and the payment it can charge for its service depends on the competition it faces. While this phenomenon can be observed both in TU and in NTU settings, traditionally, it has been studied in the transferable utility model only. Transferable utility games where the value of each coalition may depend on the coalition structure it appears in are known as *partition function games* [37]. On the other hand, games where the value of each coalition is the same in every coalition structure are known as *characteristic function games*. Clearly, characteristic function games form a proper subclass of partition function games, and tend to be much easier to work with. Thus, from now on, we will further restrict our attention to characteristic function games.

2 Definitions

In this section, we will formally define characteristic function games as well as several important subclasses of these games.

Definition 8.1 *A* characteristic function game *G is given by a pair* (A, v)*, where* $A = \{a_1, \dots, a_n\}$ *is a finite set of* players*, or* agents*, and* $v : 2^A \to \mathbb{R}$ *is a* characteristic function*, which maps each subset, or* coalition*, of agents C to a real number* $v(C)$*. This number is referred to as the* value *of the coalition C.*

We remark that we can represent a characteristic function game by explicitly listing all coalitions together with their values; the size of this naive representation is exponential in *n*. However, in practice we are usually interested in games that admit a succinct representation and can be analyzed in time polynomial in *n*. A number of such representations have been considered in the literature; we will discuss some of them in Section 4.

We will now present two examples of characteristic function games.

Example 8.2 *Charlie (C), Marcie (M), and Pattie (P) want to pool their savings to buy ice cream. Charlie has c dollars, Marcie has m dollars, Pattie has p dollars, and the ice cream packs come in three different sizes: (1) 500g which costs \$7,* *(2) 750g which costs \$9, and (3) 1000g which costs \$11. The children value ice cream, and assign no utility to money. Thus, the value of each coalition is determined by how much ice cream it can buy.*

This situation corresponds to a characteristic function game with the set of players $A = \{C, M, P\}$. *For* $c = 3$, $m = 4$, $p = 5$, *its characteristic function* v *is given by* $v(\emptyset) = 0$, $v(\{C\}) = v(\{M\}) = v(\{P\}) = 0$, $v(\{C,M\}) = v(\{C,P\}) = 500$, $v(\{M,P\}) = 750$, $v(\{C,M,P\}) = 1000$. *For* $c = 8$, $m = 8$, $p = 1$, *its characteristic function* v *is given by* $v(\emptyset) = 0$, $v(\{C\}) = v(\{M\}) = 500$, $v(\{P\}) = 0$, $v(\{C,P\}) = v(\{M,P\}) = 750$, $v(\{C,M\}) = 1250$, $v(\{C,M,P\}) = 1250$.

Example 8.3 *A fictional country X has a 101-member parliament, where each representative belongs to one of the four parties: Liberal (L), Moderate (M), Conservative (C), or Green (G). The Liberal party has 40 representatives, the Moderate party has 22 representatives, the Conservative party has 30 representatives, and the Green party has 9 representatives. The parliament needs to decide how to allocate \$1 billion of discretionary spending, and each party has its own preferred way of using this money. The decision is made by a simple majority vote, and we assume that all representatives vote along the party lines. Parties can form coalitions; a coalition has value \$1 billion if it can win the budget vote no matter what the other parties do, and value 0 otherwise.*

This situation can be modeled as a 4*-player characteristic function game, where the set of players in* $A = \{L, M, C, G\}$ *and the characteristic function* v *is given by*

$$v(C) = \begin{cases} 0 & \textit{if } |C| \leq 1, \textit{ or } |C| = 2 \textit{ and } G \in C \\ 10^9 & \textit{otherwise.} \end{cases}$$

It is usually assumed that the value of the empty coalition $\emptyset$ is 0, i.e., $v(\emptyset) = 0$. Moreover, it is often the case that the value of each coalition is non-negative (i.e., agents form coalitions to make a profit), or else that the value of each coalition is non-positive (i.e., agents form coalitions to share costs). Throughout this chapter, we will mostly focus on the former scenario, i.e., we assume that $v(C) \geq 0$ for all $C \subseteq A$. However, all our definitions and results can be easily adapted to the latter scenario.

2.1 Outcomes

An outcome of a characteristic function game consists of two parts: (1) a partition of players into coalitions, and (2) a payoff vector, which distributes the value of each coalition among its members.

Formally, a *coalition structure* over A is a collection of non-empty coalitions $CS = \{C_1, \ldots, C_{|CS|}\}$ such that

- $\bigcup_{j=1}^{|CS|} C_j = A$, and
- $C_i \cap C_j = \emptyset$ for any $i, j = 1, \ldots, |CS|$ such that $i \neq j$.

We will denote the space of all coalition structures over A by $\mathcal{P}^A$. Also, given a coalition structure $CS = \{C_1, \ldots, C_{|CS|}\} \in \mathcal{P}^A$, we will say that a vector $\mathbf{x} = (x_1, \ldots, x_n)$ is a *payoff vector* for CS, where x_i specifies the payoff of a_i in CS, if

- $x_i \geq 0$ for all $i = 1, \ldots, n$, and
- $\sum_{i:a_i \in C_j} x_i = v(C_j)$ for any $j = 1, \ldots, |CS|$.

Definition 8.2 *Given a characteristic function game $G = (A, v)$, an* outcome *of G is a pair $(CS, \mathbf{x})$, where $CS \in \mathcal{P}^A$ and $\mathbf{x}$ is a payoff vector for CS.*

A payoff vector $\mathbf{x}$ for a coalition structure $CS \in \mathcal{P}^A$ is said to be an *imputation* if it satisfies the *individual rationality* condition, i.e., $x_i \geq v(\{a_i\})$ for each $a_i \in A$. If a payoff vector is an imputation, each player weakly prefers being in the coalition structure to being on its own. Now, of course, players may still find it profitable to deviate *as a group*; we will discuss the issue of stability against group deviations in Section 3. However, before we do that, let us consider a few important classes of characteristic function games, and discuss the relationship among them.

2.2 Subclasses of Characteristic Function Games

We will now define four important subclasses of coalitional games: monotone games, superadditive games, convex games, and simple games.

2.2.1 Monotone Games

Usually, adding an agent to an existing coalition can only increase the overall productivity of this coalition; games with this property are called *monotone games*.

Definition 8.3 *A characteristic function game $G = (A, v)$ is said to be* monotone *if $v(C') \leq v(C'')$ for every pair of coalitions $C', C'' \subseteq A$ such that $C' \subseteq C''$.*

2.2.2 Superadditive Games

A stronger property, which is also enjoyed by many practically useful games, is *superadditivity*: in a *superadditive game*, it is always profitable for two groups of players to join forces.

Definition 8.4 *A characteristic function game $G = (A, v)$ is said to be* superadditive *if $v(C' \cup C'') \geq v(C') + v(C'')$ for every pair of disjoint coalitions $C', C'' \subseteq A$.*

Since we have assumed that the value of each coalition is non-negative, superadditivity implies monotonicity: if a game $G = (A, v)$ is superadditive, and $C' \subseteq C''$, then $v(C') \leq v(C'') - v(C'' \setminus C') \leq v(C'')$. However, the converse is not necessarily true: consider, for instance, a game where the value of the characteristic function grows logarithmically with the coalition size, i.e., $v(C') = \log|C'|$.

In superadditive games, there is no compelling reason for agents to form a coalition structure consisting of multiple coalitions: the agents can earn at least as much profit by forming the *grand coalition*, i.e., the coalition that contains all agents. Therefore, for superadditive games it is usually assumed that the agents form the grand coalition, i.e., the outcome of a superadditive game is of the form $(\{A\}, \mathbf{x})$ where $\mathbf{x}$ satisfies $\sum_{i=1}^{n} x_i = v(A)$. Conventionally, $\{A\}$ is omitted from the notation, i.e., an outcome of a superadditive game is identified with a payoff vector for the grand coalition.

2.2.3 Convex Games

The superadditivity property places a restriction on the behavior of the characteristic function v on disjoint coalitions. By placing a similar restriction on v's behavior on non-disjoint coalitions, we obtain the class of *convex games*.

Definition 8.5 *A characteristic function game $G = (A, v)$ is said to be* convex *if $v(C \cup C') + v(C \cap C') \geq v(C) + v(C')$ for every pair of coalitions $C, C' \subseteq A$.*

Convex games have a very intuitive characterization in terms of players' marginal contributions: in a convex game, a player is more useful when it joins a bigger coalition.

Proposition 8.1 *A characteristic function game $G = (A, v)$ is convex if and only if for every pair of coalitions C', C'' such that $C' \subset C''$ and every player $a_i \in A \setminus C''$ it holds that $v(C'' \cup \{a_i\}) - v(C'') \geq v(C' \cup \{a_i\}) - v(C')$.*

Proof. For the "only if" direction, assume that $G = (A, v)$ is convex, and consider two coalitions C', C'' such that $C' \subset C'' \subset A$ and a player $a_i \in A \setminus C''$. By setting $X = C''$, $Y = C' \cup \{a_i\}$, we obtain

$$v(C'' \cup \{a_i\}) - v(C'') = v(X \cup y) - v(X) \geq v(Y) - v(X \cap Y) = v(C' \cup \{a_i\}) - v(C'),$$

which is exactly what we need to prove.

The "if" direction can be proved by induction on the size of $X \setminus Y$; we leave the proof as an exercise for the reader. ■

Any convex game is necessarily superadditive: if a game $G = (A, v)$ is convex, and C' and C'' are two disjoint subsets of A, then we have $v(C' \cup C'') \geq v(C') +$

$v(C'') - v(C' \cap C'') = v(C') + v(C'')$ (here we use our assumption that $v(\emptyset) = 0$). To see that the converse is not always true, consider a game $G = (A, v)$, where $A = \{a_1, a_2, a_3\}$, and $v(C) = 1$ if $|C| \geq 2$ and $v(C) = 0$ otherwise. It is easy to check that this game is superadditive. On the other hand, for $C' = \{a_1, a_2\}$ and $C'' = \{a_2, a_3\}$, we have $v(C') = v(C'') = 1$, $v(C' \cup C'') = 1$, $v(C' \cap C'') = 0$.

2.2.4 Simple Games

Another well-studied class of coalitional games is that of simple games: a game $G = (A, v)$ is said to be a *simple game* if it is monotone and the characteristic function only takes values 0 and 1, i.e., $v(C) \in \{0, 1\}$ for every $C \subseteq A$. For instance, the game in Example 8.3 becomes a simple game if we rescale the payoffs so that they become 0 and 1 (instead of 0 and 10^9). In a simple game, coalitions of value 1 are said to be *winning*, and coalitions of value 0 are said to be *losing*. Such games model situations where there is a task to be completed: a coalition is labeled as winning if and only if it can complete the task.

Note that simple games are superadditive if and only if the complement of each winning coalition is losing. Clearly, there exist simple games that are not superadditive. Nevertheless, it is usually assumed that the outcome of a simple game is a payoff vector for the grand coalition, just as in superadditive games.

3 Solution Concepts

Any partition of agents into coalitions and any payoff vector that respects this partition correspond to an outcome of a characteristic function game. However, not all outcomes are equally desirable. For instance, if all agents contribute equally to the value of a coalition, a payoff vector that allocates the entire payoff to one of the agents is less appealing than the one that shares the profits equally among all agents. Similarly, an outcome that incentivizes all agents to work together is preferable to an outcome that some of the agents want to deviate from.

More broadly, one can evaluate the outcomes according to two sets of criteria: (1) *fairness*, i.e., how well each agent's payoff reflects its contribution, and (2) *stability*, i.e., what the incentives are for the agents to stay in the coalition structure. These two sets of criteria give rise to two families of payoff division schemes, or *solution concepts*. We will now discuss each of them in turn.

3.1 Shapley Value

The best-known solution concept that aims to capture the notion of fairness in characteristic function games is the *Shapley value* [64]. The Shapley value is usually defined for superadditive games. As argued above, for such games an outcome can be identified with a payoff vector for the grand coalition, i.e., the Shapley value prescribes how to share the value of the grand coalition in a fair way.

To present the formal definition of the Shapley value, we need some additional notation. Given a characteristic function game $G=(A,v)$, let Π^A denote the set of all *permutations* of A, i.e., one-to-one mappings from A to itself. Given a permutation $\pi \in \Pi^A$, we denote by $C_\pi(a_i)$ the coalition that consists of all predecessors of a_i in π, i.e., we set $C_\pi(a_i) = \{a_j \in A \mid \pi(a_j) < \pi(a_i)\}$. The *marginal contribution* of an agent a_i with respect to a permutation π in a game $G=(A,v)$ is denoted by $\Delta_\pi^G(a_i)$ and is given by

$$\Delta_\pi^G(a_i) = v(C_\pi(a_i) \cup \{a_i\}) - v(C_\pi(a_i));$$

this quantity measures by how much a_i increases the value of the coalition consisting of its predecessors in π when it joins them. Informally, the Shapley value of a player a_i is its average marginal contribution, where the average is taken over all permutations of A. More formally, we have the following definition.

Definition 8.6 *Given a characteristic function game $G=(A,v)$, the* Shapley value *of a player $a_i \in A$ is denoted by $\varphi_i(G)$ and is given by*

$$\varphi_i(G) = \frac{1}{n!} \sum_{\pi \in \Pi^A} \Delta_\pi^G(a_i).$$

The Shapley value has many attractive properties. In what follows, we list four of them; the proofs of Propositions 8.2–8.5 are left as an exercise for the reader.

First, the Shapley value is *efficient*, i.e., it distributes the value of the grand coalition among all agents.

Proposition 8.2 *For any characteristic function game $G=(A,v)$, we have $\sum_{i=1}^n \varphi_i(G) = v(A)$.*

Second, the Shapley value does not allocate any payoffs to players who do not contribute to any coalition. Formally, given a characteristic function game $G=(A,v)$, a player $a_i \in A$ is said to be a *dummy* if $v(C) = v(C \cup \{a_i\})$ for every $C \subseteq A$. It is not hard to see that the Shapley value of a dummy player is 0.

Proposition 8.3 *If a player $a_i \in A$ is a dummy in a characteristic function game G, then $\varphi_i(G) = 0$.*

Third, if two players contribute equally to each coalition, then their Shapley values are equal. Formally, given a characteristic function game $G=(A,v)$, we say that players a_i and a_j are *symmetric* in G if $v(C \cup \{a_i\}) = v(C \cup \{a_j\})$ for every coalition $C \subseteq A \setminus \{a_i, a_j\}$. It turns out that symmetric players have equal Shapley values.

Proposition 8.4 *If players a_i and a_j are symmetric in a characteristic function game G, then $\varphi_i(G) = \varphi_j(G)$.*

Finally, consider a group of players A that is involved in two coalitional games G' and G'', i.e., $G' = (A, v')$, $G'' = (A, v'')$. The *sum* of G' and G'' is a coalitional game $G^+ = G' + G''$ given by $G^+ = (A, v^+)$, where for every coalition $C \subseteq A$ we have $v^+(C) = v'(C) + v''(C)$. It can easily be seen that the Shapley value of a player a_i in G^+ is the sum of its Shapley values in G' and G''.

Proposition 8.5 *Consider two characteristic function games $G' = (A, v)$ and $G'' = (A, v)$ over the same set of players A. Then for any player $a_i \in A$ we have $\varphi_i(G' + G'') = \varphi_i(G') + \varphi_i(G'')$.*

To summarize, we have argued that the Shapley value possesses four desirable properties:

(1) *Efficiency*: all the profit earned by the agents in the grand coalitions is distributed among them;

(2) *Null player*: players with zero marginal contributions to all coalitions receive zero payoff;

(3) *Symmetry*: all players that have the same marginal contribution to all coalitions receive the same payoff;

(4) *Additivity*: $\varphi_i(G' + G'') = \varphi_i(G') + \varphi_i(G'')$ for all $a_i \in A$.

Interestingly, the Shapley value is the only payoff division scheme that has these four properties simultaneously [64]. In other words, if we view properties (1)–(4) as axioms, then these axioms characterize the Shapley value.

3.2 Banzhaf Index

Another solution concept that is motivated by fairness considerations is the *Banzhaf index* [7]. The difference between the Shapley value and the Banzhaf index can be described in terms of the underlying coalition formation model: the Shapley value measures the agent's expected marginal contribution if agents join the coalition one by one in a random order, whereas the Banzhaf index measures the agent's expected marginal contribution if each agent decides whether to join the coalition independently with probability $1/2$. This intuition is formally captured by the following definition.

Definition 8.7 *Given a characteristic function game $G = (A, v)$, the* Banzhaf index *of a player $i \in A$ is denoted by $\beta_i(G)$ and is given by*

$$\beta_i(G) = \frac{1}{2^{n-1}} \sum_{C \subseteq A \setminus \{a_i\}} [v(C \cup \{a_i\}) - v(C)].$$

It is not hard to verify that the Banzhaf index satisfies properties (2), (3), and (4) in the list above. However, it does not satisfy property (1), i.e., efficiency.

Example 8.4 *Consider a characteristic function game $G = (A, v)$, where $v(A) = 1$ and $v(C) = 0$ for every $C \subset A$. We have $\varphi_i(G) = \frac{1}{n}$, $\beta_i(G) = \frac{1}{2^{n-1}}$ for each $a_i \in A$.*

Since efficiency is a very desirable property of a payoff distribution scheme, some researchers also consider the *normalized Banzhaf index* $\eta_i(G)$, which is defined as

$$\eta_i(G) = \frac{\beta_i(G)}{\sum_{i \in A} \beta_i(G)}.$$

While this version of the Banzhaf index satisfies efficiency, it loses the additivity property.

3.3 Core

We have introduced two solution concepts that attempt to measure the agents' marginal contribution. In contrast, the solution concepts considered in this and subsequent sections are defined in terms of coalitional stability.

Consider a characteristic function game $G = (A, v)$ and an outcome $(CS, \mathbf{x})$ of this game. Let $x(C)$ denote the total payoff of a coalition C under a payoff vector $\mathbf{x}$, i.e., $x(C) = \sum_{i: a_i \in C} x_i$. Now, if $x(C) < v(C)$, then the agents in C have an incentive to deviate since they could do better by abandoning CS and forming a coalition of their own. For example, if the agents were to share the extra profit equally among themselves, every agent $a_i \in C$ would receive a payoff of $x_i + \frac{v(C) - x(C)}{|C|}$ instead of x_i. An outcome where no subset of agents has an incentive to deviate is called *stable*, and the set of all such outcomes is called the *core* of G [29].

Definition 8.8 *The* core *of a characteristic function game $G = (A, v)$ is the set of all outcomes $(CS, \mathbf{x})$ such that $x(C) \geq v(C)$ for any $C \subseteq A$.*

In a superadditive game, the outcomes are payoff vectors for the grand coalition, so for such games the core can be defined as the set of all vectors $\mathbf{x}$ that satisfy: (1) $x_i \geq 0$ for all $a_i \in A$, (2) $x(A) = v(A)$, and (3) $x(C) \geq v(C)$ for all $C \subseteq A$.

The outcomes in the core are stable and therefore they are more likely to arise when a coalitional game is played. However, some games have empty cores.

Example 8.5 *Consider the game $G = (A,v)$, where $A = \{a_1,a_2,a_3\}$, $v(C) = 1$ if $|C| \geq 2$ and $v(C) = 0$ otherwise. We claim that this game has an empty core. Indeed, suppose that the core of G is non-empty. Since G is superadditive, its core contains a vector $\mathbf{x} = (x_1,x_2,x_3)$, where $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, and $x_1 + x_2 + x_3 = 1$. The latter constraint implies that $x_i \geq \frac{1}{3}$ for some $a_i \in A$. But then for $C = A \setminus \{a_i\}$ we have $v(C) = 1$, $x(C) \leq 2/3$, which means that (x_1,x_2,x_3) is not in the core. This contradiction shows that the core of G is empty.*

Observe that the set of all outcomes in the core of a superadditive game can be characterized by the following linear feasibility program (LFP):

$$\begin{aligned} x_i &\geq 0 \quad \text{for each } a_i \in A \\ \sum_{i:a_i\in A} x_i &= v(A) \\ \sum_{i:a_i\in C} x_i &\geq v(C) \quad \text{for each } C \subseteq A \end{aligned} \tag{8.1}$$

This LFP has $2^n + n + 1$ constraints. Therefore, if we want to convert it into an algorithm for checking non-emptiness of the core which runs in time polynomial in n, we need an efficient *separation oracle* for this LFP. Recall that a separation oracle for a linear (feasibility) program is a procedure that, given a candidate solution $(x_1,\dots,x_n)$, determines whether it is feasible, and, if not, outputs the violated constraint. It is well-known that if a linear program over n variables admits a separation oracle that runs in time poly(n), then an optimal feasible solution can be found in time poly(n) [62].

Now, the first $n+1$ constraints in our LFP are straightforward to check. Therefore, the problem of checking non-emptiness of the core for superadditive games can be reduced to checking whether a candidate solution satisfies the last 2^n constraints, i.e., verifying whether a given outcome is in the core (and, if not, computing the coalition that has an incentive to deviate). In general, checking whether a given outcome is in the core and/or deciding whether the core is non-empty is not easy: in Section 4, we will see examples of classes of coalitional games for which these problems are NP-hard. However, we will now see that for some of the classes of games discussed in Section 2.2, these problems are efficiently solvable.

3.3.1 The Core of Simple Games

Recall that for simple games it is standard to assume that the grand coalition forms, even if the game is not superadditive. Under this assumption, it is easy to characterize the outcomes in the core, and provide a simple criterion for checking whether the game has a non-empty core.

A player a_i in a simple game $G = (A, v)$ is said to be a *veto player* if $v(C) = 0$ for any $C \subseteq A \setminus \{a_i\}$; since simple games are monotone, this is equivalent to requiring that $v(A \setminus \{a_i\}) = 0$. Observe that a game may have more than one veto player: for instance, in the unanimity game, where $v(A) = 1$, $v(C) = 0$ for any $C \subset A$, all players are veto players. We will now show that the only way to achieve stability is to share the payoff among the veto players, if they exist.

Theorem 8.1 *A simple game $G = (A, v)$ has a non-empty core if and only if it has a veto player. Moreover, an outcome $(x_1, \dots, x_n)$ is in the core of G if and only if $x_i = 0$ for any player a_i who is not a veto player in G.*

Proof. Suppose G has a veto player a_i. Then the outcome $\mathbf{x}$ with $x_i = 1$, $x_j = 0$ for $j \neq i$ is in the core: any coalition C that contains a_i satisfies $x(C) = 1 \geq v(C)$, whereas any coalition C' that does not contain a_i satisfies $v(C') = 0 \leq x(C')$.

Conversely, suppose that G does not have a veto player. Suppose for the sake of contradiction that G has a non-empty core, and let $\mathbf{x}$ be an outcome in the core of G. Since $x(A) = 1$, we have $x_i > 0$ for some $a_i \in A$, and hence $x(A \setminus \{a_i\}) = 1 - x_i < 1$. However, since a_i is not a veto player, we have $v(A \setminus \{a_i\}) = 1 > x(A \setminus \{a_i\})$, a contradiction with $\mathbf{x}$ being in the core.

The second statement of the theorem can be proved similarly. ■

The characterization of the outcomes in the core provided by Theorem 8.1 suggests a simple algorithm for checking if an outcome is in the core or deciding non-emptiness of the core: it suffices to determine, for each player a_i, whether it is a veto player, i.e., to compute $v(A \setminus \{a_i\})$. Thus, if the characteristic function of a simple game is efficiently computable, we can answer the core-related questions in polynomial time.

We remark that if the simple game is not superadditive, and we use the more general definition of an outcome, i.e., allow the players to form coalition structures, Theorem 8.1 no longer holds. Moreover, deciding whether an outcome is in the core becomes computationally hard even for fairly simple representation formalisms (see Section 4.1).

3.3.2 The Core of Convex Games

Convex games always have a non-empty core. We will now present a constructive proof of this fact, i.e., show how to obtain an outcome in the core of a convex game.

Theorem 8.2 *If $G = (A, v)$ is a convex game, then G has a non-empty core.*

Proof. Fix an arbitrary permutation $\pi \in \Pi^A$, and set $x_i = \Delta_\pi^G(a_i)$. We claim that $(x_1, \dots, x_n)$ is in the core of G.

Indeed, observe first that any convex game is monotone, so $x_i \geq 0$ for all $a_i \in A$. Moreover, we have $\sum_{i=1}^n x_i = \Delta_\pi^G(a_1) + \cdots + \Delta_\pi^G(a_n) = v(A)$. Finally, suppose for the sake of contradiction that we have $v(C) > x(C)$ for some coalition $C = \{a_{i_1}, \dots . a_{i_s}\}$. We can assume without loss of generality that $\pi(a_{i_1}) \leq \cdots \leq \pi(a_{i_s})$, i.e., the members of C appear in π ordered as $a_{i_1}, \dots, a_{i_s}$. We can write $v(C)$ as

$$v(C) = v(\{a_{i_1}\}) - v(\emptyset) + v(\{a_{i_1}, a_{i_2}\}) - v(\{a_{i_1}\}) + \cdots + v(C) - v(C \setminus \{a_{i_s}\}).$$

Now, for each $j = 1, \dots, s$, the supermodularity of v implies

$$v(\{a_{i_1}, \dots, a_{i_j}\}) - v(\{a_{i_1}, \dots, a_{i_{j-1}}\}) \leq v(\{a_1, \dots, a_{i_j}\}) - v(\{a_1, \dots, a_{i_j - 1}\}) = x_{i_j}.$$

By adding up these inequalities, we obtain $v(C) \leq x(C)$, i.e., coalition C does not have an incentive to deviate, which is a contradiction. ■

Observe that the construction used in the proof of Theorem 8.2 immediately implies that in a convex game the Shapley value is in the core: indeed, the Shapley value is a convex combination of outcomes constructed in the proof of Theorem 8.2, and the core can be shown to be a convex set. However, Theorem 8.2 does not, in general, enable us to check whether a given outcome of a convex game is in the core of that game.

3.4 The Least Core

When a given game has an empty core, we may still be interested in finding "the most stable" outcome. In this section, we explore solution concepts that are motivated by this idea. In what follows, we focus on superadditive games; however, many of our definitions also apply to general characteristic function games.

In many situations, a coalition would prefer not to deviate if its gain from a deviation is positive, but tiny. Therefore, we may view outcomes in which no coalition can improve its welfare significantly as stable. This motivates the following definition.

Definition 8.9 *An outcome* $\mathbf{x}$ *is said to be in the* ε*-core of a superadditive game* G *for some* $\varepsilon \in \mathbb{R}$ *if* $x(C) \geq v(C) - \varepsilon$ *for each* $C \subseteq A$.

Of course, in practice we are usually interested in finding the smallest value of ε such that the ε-core is non-empty. The corresponding ε-core is called the *least core* of G [39]. More formally, we have the following definition.

Definition 8.10 *Given a superadditive game G, let*

$$\varepsilon^*(G) = \inf\{\varepsilon \mid \varepsilon\text{-}core\ of\ G\ is\ non\text{-}empty\}.$$

The least core of G is its $\varepsilon^*(G)$*-core. The quantity* $\varepsilon^*(G)$ *is called the* value of the least core *of G.*

To see that the least core is always non-empty, observe that we can modify the linear feasibility program (8.1) so as to obtain a linear program for the value of the least core as well as a payoff vector in the least core. Specifically, we have

$$\begin{aligned}
\min \quad \varepsilon & \qquad \text{subject to:} \\
x_i &\geq 0 \quad \text{for each } a_i \in A \\
\sum_{i:a_i \in A} x_i &= v(I) \\
\sum_{i:a_i \in C} x_i &\geq v(C) - \varepsilon \quad \text{for each } C \subseteq A.
\end{aligned} \tag{8.2}$$

Clearly, if $(\varepsilon, x_1, \ldots, x_n)$ is an optimal solution to this linear program, then ε is the value of the least core and $(x_1, \ldots, x_n)$ is an outcome in the least core. This shows that we can compute the value of the least core of a superadditive game G as long as we have an algorithm for checking if a given outcome is in the core of G (and, if not, finding the deviating coalition).

Observe that if G has a non-empty core, it may happen that $\varepsilon^*(G) < 0$, in which case the least core is a subset of the core. We remark, however, that some authors require the value of the least core to be non-negative, i.e., they define the least core as the smallest *non-negative* value of ε for which the ε-core is non-empty. Under this definition, to compute the value of the least core we need to add the constraint $\varepsilon \geq 0$ to the linear program (8.2).

3.5 Other Solution Concepts

Besides the Shapley value, the Banzhaf index, the core, and the least core, there are several other solution concepts for characteristic function games. The most prominent among them are the nucleolus, the kernel, and the bargaining set. We will not be able to discuss them in full detail in this chapter due to space constraints; instead, we will provide a brief intuitive description of each of these concepts. For a more comprehensive treatment, the interested reader is referred to [47, 48].

The *nucleolus* [61] can be thought of as a refinement of the least core. Specifically, the least core can be defined as the set of all payoff vectors that minimize the maximum *deficit* $d_1 = \max\{v(C) - x(C) \mid C \subseteq A\}$. Now, among all payoff vectors in the least core, we can pick the ones that minimize the second highest deficit $d_2 = \max\{v(C) - x(C) \mid C \subseteq A, v(C) - x(C) < d_1\}$, and remove all other payoff vectors. We can continue this procedure until the set of the surviving payoff vectors stabilizes. The resulting set can be shown to consist of a single payoff vector: this payoff vector is known as the *pre-nucleolus*. If, at each step, we only consider imputations (rather than arbitrary payoff vectors), we obtain the *nucleolus*. The nucleolus is an attractive solution concept, as it arguably identifies the most stable outcome of a game. However, its formal definition involves an exponentially long vector, and therefore the nucleolus is not easy to compute from the first principles. However, some classes of games defined on combinatorial structures (see Section 4) admit efficient algorithms for computing the nucleolus: see, e.g., [19, 26, 36].

The *kernel* [17] consists of all outcomes where no player can credibly demand a fraction of another player's payoff. Formally, for any player a_i we define its *surplus* over the player a_j with respect to a payoff vector $\mathbf{x}$ as the quantity

$$sur_{i,j}(\mathbf{x}) = \max\{v(C) - x(C) \mid C \subseteq A, a_i \in C, a_j \notin C\}.$$

Intuitively, this is the amount that a_i can earn without the cooperation of a_j, by asking a set $C \setminus \{a_i\}$ to join it in a deviation, and paying each player in $C \setminus \{a_i\}$ what it used to be paid under $\mathbf{x}$. Now, if $sur_{i,j}(\mathbf{x}) > sur_{j,i}(\mathbf{x})$, player a_i should be able to demand a fraction of player a_j's payoff – unless player a_j already receives the smallest payment that satisfies the individual rationality condition, i.e., $v(\{a_j\})$. Following this intuition, we say that an imputation $\mathbf{x}$ is in the *kernel* of a superadditive game G if for any pair of players (a_i, a_j) we have either: (1) $sur_{i,j}(\mathbf{x}) = sur_{j,i}(\mathbf{x})$, or (2) $sur_{i,j}(\mathbf{x}) > sur_{j,i}(\mathbf{x})$ and $x_j = v(\{a_j\})$, or (3) $sur_{i,j}(\mathbf{x}) < sur_{j,i}(\mathbf{x})$ and $x_i = v(\{a_i\})$.

The *bargaining set* [38] is defined similarly to the core. However, in contrast to the definition of the core, we only take into account coalitional deviations that are themselves stable, i.e., do not admit a counterdeviation. Consequently, the bargaining set contains the core, and the containment is sometimes strict. In fact, the bargaining set can be shown to contain the least core [22], which implies that the bargaining set is guaranteed to be non-empty.

4 Representation Formalisms

It would be desirable to have a representation language that allows us to encode all coalitional games so that the description size of each game is polynomial in the number of agents n. However, a simple counting argument shows that no representation formalism can encode each coalitional game using poly(n) bits; this is true even if we restrict ourselves to simple games. Therefore, one needs to decide on a trade-off between *expressiveness*, i.e., the formalism's ability to encode many different games, and *succinctness*, i.e., the resulting description size. For instance, one option is to choose a formalism that can only represent games in a certain subclass of coalitional games, but guarantees that each game in this class has a succinct encoding. Alternatively, one can choose a formalism that can represent any coalitional game, but is only guaranteed to produce succinct representation for games that have certain special properties.

In this chapter, we will discuss several formalisms for characteristic function games. We start with *restricted representation languages*, i.e., formalisms that are always succinct, but not fully expressive.

4.1 Weighted Voting Games

In a *weighted voting game*, each player has a certain *weight*, which encodes the amount of resources available to this player. Further, there is a task that can be accomplished by any coalition that has sufficient resources. If a coalition can accomplish the task, it earns a fixed payoff, which can be normalized to 1; otherwise, it earns nothing. Formally, weighted voting games are defined as follows.

Definition 8.11 *A* weighted voting game *G is given by a triple $(A, \mathbf{w}, q)$, where A is the set of players, $|A| = n$, $\mathbf{w} = (w_1, \dots, w_n) \in \mathbb{R}^n$ is a vector of* weights*, and $q \in \mathbb{R}$ is a* quota*. The characteristic function v of a game $G = (A, \mathbf{w}, q)$ is given by $v(C) = 1$ if $\sum_{i:a_i \in C} w_i \geq q$ and $v(C) = 0$ otherwise.*

It is usually assumed that all weights and the quota are integers given in binary; it can be shown that this assumption can be made without loss of generality. Further, most of the work on weighted voting games assumes that all weights are non-negative; observe that in this case weighted voting games are simple games.

Weighted voting games are used to model decision making in voting bodies; for instance, the game described in Example 8.3 is a weighted voting game with quota $q = 51$ and weights 40, 22, 30, 9 for players L, M, C, G, respectively. Indeed, the Shapley value and the Banzhaf index in such games are often viewed as measures of a party's voting power in a parliament and have therefore received significant attention from political scientists. In such settings it is usually assumed that the quota q is at least half of the players' total weight; however, in general task execution scenarios the quota q can take any value between 0 and $\sum_{i=1}^{n} w_i$.

It is important to note that a player's power in a weighted voting game is not necessarily proportional to its weight. Indeed, in Example 8.3, the Liberal party and the Moderate party have the same Shapley value (namely, $1/3$), even though their weights differ by almost a factor of 2. Moreover, the Green party is a dummy and thus its Shapley value is 0, even though it has a non-zero weight. Observe also that if we changed the quota to, say, $q' = 60$, the balance of power would change: for instance, we would have $v(\{M,C\}) = 0$, but $v(\{M,C,G\}) = 1$, so G would no longer be a dummy.

4.1.1 Computational Issues

The complexity of computing fair and stable outcomes in weighted voting games has received significant attention in the literature.

For instance, the complexity of determining the players' Shapley values has been analyzed by a variety of authors [21, 41, 50]. An easy reduction from the SUBSET SUM problem shows that deciding whether a player is a dummy is coNP-complete; this implies that deciding whether a player's Shapley value is equal to 0 is coNP-complete as well. In fact, one can strengthen this result to show that computing the Shapley value is #P-complete.

Fortunately, the situation is considerably less bleak if we can assume that all weights are at most polynomial in the number of players n, or, equivalently, are given in unary. Under this assumption, we would be satisfied with algorithms whose running time is polynomial in n and the maximum weight, i.e., $\max_{i:a_i \in A} w_i$. It is not too hard to show that such algorithms do exist: a dynamic programming-based approach has been described by Matsui and Matsui [40].

The same easiness and hardness results hold for the Banzhaf index: it is #P-complete to compute when weights are given in binary, but admits an efficient dynamic programming-based algorithm for small weights.

The core-related questions are easy to answer if we make the standard assumption that the grand coalition always forms: indeed, since weighted voting games are simple games, there is a stable way of dividing the payoffs of the grand coalition if and only if the game has veto players. Now, determining if a player a_i is a veto player in a weighted voting game $G = (A, \mathbf{w}, q)$ is easy: it suffices to check whether $\sum_{j:a_j \neq a_i} w_j \geq q$. This implies that there are polynomial-time algorithms for checking if an outcome is in the core or determining whether the core is non-empty.

However, if $q < \sum_{i=1}^{n} w_i/2$, then forming the grand coalition may be inefficient, and therefore there may exist stable outcomes in which the agents form a non-trivial coalition structure. Indeed, consider the weighted voting game $G = \big(\{a_1, a_2, a_3, a_4\}, (2,2,2,2), 4\big)$. This game does not have a veto player, and therefore any outcome in which the grand coalition forms is not stable. On the other hand, it is easy to see that the outcome $(\{\{a_1, a_2\}, \{a_3, a_4\}\}, (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}))$ is stable: any winning coalition contains at least two players, and therefore its payoff is at least 1.

Now, when arbitrary coalition structures are allowed, checking whether a stable outcome exists, or even whether a given outcome is stable, becomes difficult.

Specifically, Elkind et al. [23] showed that that former problem is NP-hard, while the latter problem is coNP-complete. On the positive side, they also showed that if all weights are polynomially bounded, one can check in polynomial time whether an outcome is in the core. It is currently open whether a similar easiness result holds for the problem of checking the non-emptiness of the core; although it is conjectured that this problem remains hard even for small weights [23].

Elkind et al. analyze the complexity of computing the value of the least core and the nucleolus [24, 26]. Again, a familiar picture emerges: both problems are hard when weights are given in binary, but easy when weights are given in unary. Moreover, even for large weights, the value of the least core admits a fully polynomial-time approximation scheme (FPTAS), i.e., an algorithm that, given a weighted voting game $G = (A, \mathbf{w}, q)$ and a parameter δ, outputs a value ε' that satisfies $\varepsilon \leq \varepsilon' \leq (1+\delta)\varepsilon$, where ε is the true value of the least core of G, and runs in time that is polynomial in the number of players n, the maximum weight, and $1/\delta$.

4.1.2 Expressivity and Vector Weighted Voting Games

When all weights are non-negative, weighted voting games are simple games. However, one may wonder if the converse is also true, i.e., whether given a simple game $G = (A, v)$ with $|A| = n$ we can always find a vector of weights $\mathbf{w} = \{w_1, \ldots, w_n\}$ and a quota q such that G is equivalent to the game $(A, \mathbf{w}, q)$, i.e., for every $C \subseteq A$ it holds that $v(C) = 1$ if and only if $\sum_{i:a_i \in C} w_i \geq q$.

It is not hard to show that the answer to this question is “no.” Indeed, consider a simple game $G = (A, v)$ with $A = \{a_1, a_2, a_3, a_4\}$, where a coalition is winning if it contains both an even-numbered agent and an odd-numbered agent, or, in symbols, $v(C) = 1$ if and only if $C \cap \{a_1, a_3\} \neq \emptyset$ and $C \cap \{a_2, a_4\} \neq \emptyset$. Suppose that this game can be represented as a weighted voting game $(A, \mathbf{w}, q)$ for some real weights and quota (note that we do not assume that the weights are positive or rational). Since $\{a_1, a_2\}$ and $\{a_3, a_4\}$ are winning coalitions, we have $w_1 + w_2 \geq q$, $w_3 + w_4 \geq q$, and hence $w_1 + w_2 + w_3 + w_4 \geq 2q$. On the other hand, $\{a_1, a_3\}$ and $\{a_2, a_4\}$ are losing coalitions, so we have $w_1 + w_3 < q$, $w_2 + w_4 < q$, and hence $w_1 + w_2 + w_3 + w_4 < 2q$. This contradiction shows that G is not equivalent to any weighted voting game.

Interestingly, the game G discussed in the previous paragraph can be viewed as an *intersection* of two weighted voting games: to win the first game, the coalition must contain an odd-numbered player (this corresponds to the weighted voting game $\big(A, (1, 0, 1, 0), 1\big)$, whereas to win the second game, the coalition must contain an even-numbered player (this corresponds to the weighted voting game $\big(A, (0, 1, 0, 0), 1\big)$. To win the overall game, the coalition must win both of the component games. Such games are known as *vector weighted voting games*, or

k-weighted voting games, where k is the number of component games.

Definition 8.12 *A game $G = (A, v)$ with $|A| = n$ is said to be a* k-weighted voting game *for some $k \in \mathbb{N}$ if there exists a collection of k weighted voting games $G^1 = \big(A, (w_1^1, \dots, w_n^1), q^1\big), \dots, G^k = \big(A, (w_1^k, \dots, w_n^k), q^k\big)$ over the set of players A such that $v(C) = 1$ if and only if $\sum_{i:a_i \in C} w_i^j \geq q^j$ for every $j = 1, \dots, k$. The games $G^1, \dots, G^k$ are called the* component games *of G; we will write $G = G^1 \wedge \dots \wedge G^k$.*

Vector weighted voting games are widely used in practice: for instance, the European Union decision-making system is a 27-player 3-weighted voting game, where the three component games correspond to the commissioners, countries, and population [9].

From the computational perspective, vector weighted voting games are similar to the ordinary weighted voting games if k is bounded by a constant, but become harder to deal with if k is viewed as part of the input: for instance, Elkind et al. [27] show that deciding whether a player is a dummy in a k-weighted voting game is coNP-complete even if all weights are in $\{0, 1\}$ (recall that, in contrast, for weighted voting games this problem is easy as long as all weights are polynomially bounded).

Now, we have seen that vector weighted voting games are more expressive than weighted voting games; but are they fully expressive? We will now show that the answer is "yes," i.e., *any* simple game can be represented as a k-weighted voting game for a suitable value of k; this holds even if all weights are required to be in $\{0, 1\}$.

Theorem 8.3 *Any simple game $G = (A, v)$ with $|A| = n$ can be represented as a k-weighted voting game $G^1 \wedge \dots \wedge G^k$, where $k \leq 2^n$ and all weights in each component game are either* 0 *or* 1.

Proof. Let $C_1, \dots, C_k$ be the list of all losing coalitions in G. For each coalition C_j in this list, we construct a weighted voting game $G^j = \Big(A, (w_1^j, \dots, w_n^j), q^j\Big)$, where $q^j = 1$ and $w_i^j = 1$ if $a_i \notin C_j$, $w_i^j = 0$ if $a_i \in C_j$. Observe that a coalition C is a winning coalition in G^j if and only if it contains some agent $a_i \in A \setminus C_j$.

We claim that G is equivalent to $G' = G^1 \wedge \dots \wedge G^k$. Indeed, if $C \subseteq A$ is a losing coalition in G, then $C = C_j$ for some $j = 1, \dots, k$, and therefore C loses in the corresponding component game and hence in G'. On the other hand, if $C \subseteq A$ is a winning coalition in G, then, by monotonicity, C is not contained in any losing coalition, i.e., for any coalition C_j in our list we have $C \setminus C_j \neq \emptyset$ and hence C is a winning coalition in C_j. Since this holds for any $j = 1, \dots, k$, C is a winning coalition in G'. To complete the proof, it remains to observe that $k \leq 2^n$. ■

The minimum number of component games in the representation of a given simple game G as a weighted voting game is called the *dimension* of G. Theorem 8.3 shows that the dimension of any simple n-player game G does not exceed 2^n; on the other hand, there are explicit constructions of simple games whose dimension is exponential in the number of players [69]. Thus, vector weighted voting games are *universally expressive* for the class of all simple games, but are only succinct for some of the games in this class (namely, the games with polynomially small dimension, which includes all weighted voting games). This situation is typical of the universally expressive representation formalisms; we will see some further examples in Section 4.3.

4.2 Combinatorial Optimization Games

Several classes of cooperative games that have been studied in the operations research and theoretical computer science community are defined via a combinatorial structure, such as, for example, a graph. The value of each coalition is obtained by solving a combinatorial optimization problem on the substructure that corresponds to this coalition. We will refer to such games as *combinatorial optimization games*. Just like weighted voting games, such representations are succinct, but not complete. An excellent (though somewhat outdated) survey of combinatorial optimization games can be found in [10]. In this section, we give several examples of the games in this family.

4.2.1 Induced Subgraph Games

In *induced subgraph games* [21], players are vertices of a weighted graph, and the value of a coalition is the total weight of its internal edges. It can be checked that if all weights are non-negative, this game is convex and therefore has a non-empty core. However, if we allow negative weights, the core may be empty, and, moreover, checking whether an outcome is in the core becomes coNP-complete. In contrast, the Shapley value in this game is easy to compute even if the weights can be negative: the Shapley value of a vertex x is half of the total weight of the edges that are incident to x.

4.2.2 Network Flow Games

In *network flow games* [33, 34], the players are edges of a network with a *source* and a *sink*. Each edge has a positive integer capacity, indicating how much flow it can carry. The value of a coalition C is the maximum amount of flow that can be sent from the source to the sink using the edges in C only. Various stability-related solution concepts for this class of games were studied in [31] and subsequently in [19].

One can also consider a variant of network flow games where the value of a coalition is 1 if it can carry at least k units of flow from the source to the sink, and 0 otherwise. Such games are called *threshold network flow games*, and have been studied in [6] and subsequently in [2].

4.2.3 Matching and Assignment Games

In *assignment games* [65], agents are vertices of a weighted bipartite graph. The value of each coalition is the size of its maximum-weight induced matching. *Matching games* [20] are a generalization of assignment games, where the graph is not required to be bipartite. The complexity of the core, the least core, and the nucleolus in these games has been studied in [36, 68].

4.3 Complete Representation Languages

In this section, we will discuss four representation formalisms for coalitional games that are *complete*, i.e., can be used to describe any coalitional game.

4.3.1 Marginal Contribution Nets

Marginal contribution nets, or *MC-nets* [32], is a rule-based representation; it describes a game with a set of players $A = \{a_1, \cdots, a_n\}$ by a collection of rules $\mathcal{R}$. Each rule $r \in \mathcal{R}$ is of the form $\mathcal{B}_r \to \vartheta_r$, where $\mathcal{B}_r$ is a Boolean formula over a set of variables $\{b_1, \ldots, b_n\}$ and ϑ_r is a real value. We say that a rule $r \in \mathcal{R}$ is *applicable* to a coalition C if $\mathcal{B}_r$ is satisfied by the truth assignment given by $b_i = \top$ if $a_i \in C$ and $b_i = \bot$ if $a_i \notin C$. Let $\mathcal{R}^C$ denote the set of rules that are applicable to C. Then, the characteristic function of the game described by $\mathcal{R} = \{\mathcal{B}_1 \to \vartheta_1, \ldots, \mathcal{B}_k \to \vartheta_k\}$ is computed as follows:

$$v(C) = \sum_{r \in \mathcal{R}^C} \vartheta_r.$$

Example 8.6 *The MC-net that consists of the rules $\mathcal{R} = \{b_1 \wedge b_2 \to 5, b_2 \to 2\}$, corresponds to a coalitional game $G = (A, v)$, where $A = \{a_1, a_2\}$, $v(\{a_1\}) = 0$, $v(\{a_2\}) = 2$, $v(\{a_1, a_2\}) = 7$.*

An MC-net is said to be *basic* if the left-hand side of any rule is a conjunction of *literals*, i.e., variables and their negations. In this case, we can write a rule $r \in \mathcal{R}$ as $(P_r, N_r) \to \vartheta_r$, where P_r and N_r are the sets of agents that correspond to positive and negative literals in $\mathcal{B}_r$, respectively. Thus, r is applicable to coalition

C if C contains every agent in P_r and none of the agents in N_r. It is not hard to see that any coalitional game $G = (A, v)$ with $|A| = n$ can be represented by a basic MC-net with $2^n - 1$ rules: for each non-empty coalition $C \subseteq A$ we create a rule

$$(\wedge_{i:a_i \in C} b_i) \bigwedge (\wedge_{i:a_i \notin C} \neg b_i) \rightarrow v(C).$$

However, many interesting games admit a more succinct representation, especially if we allow MC-nets that are not basic.

For basic MC-nets, the players' Shapley values can be computed efficiently. The algorithm proceeds by decomposing a game given by k rules into k games – one for each rule; in a game described by a single basic rule, the Shapley value of each player is given by a closed-form expression. This argument extends to *read-once* MC-nets, where in each rule each literal appears at most once [25]. However, if the formulas in the rules can be arbitrary, the Shapley value becomes hard to compute. On the other hand, the core-related questions are NP-hard even for basic MC-nets [32].

4.3.2 Synergy Coalition Groups

Synergy Coalition Group (SCG) Representation [14] is a complete language for superadditive games that is obtained by trimming down the naive representation, i.e., one that lists all coalitions together with their values. It is based on the following idea. Suppose that a game $G = (A, v)$ is superadditive, and consider a coalition $C \subseteq A$. Then we have

$$v(C) \geq \max_{CS \in \mathcal{P}^C \setminus \{C\}} \sum_{C' \in CS} v(C'). \tag{8.3}$$

Now, if the inequality (8.3) holds with equality, then there is no need to store the value of C as it can be computed from the values of the smaller coalitions. Therefore, we can represent G by listing the values of all coalitions of size 1 as well as the values of the coalitions for which there is a *synergy*, i.e., the inequality (8.3) is strict.

By construction, the SCG representation is complete. Moreover, it is succinct when there are only a few groups of agents that can collaborate productively. Further, it allows for an efficient procedure for checking whether an outcome is in the core: it can be shown that if an outcome is not in the core, then there is a "synergetic" coalition, i.e., one whose value is given explicitly in our representation, which can profitably deviate. However, the SCG representation has a major drawback: computing the value of a coalition may involve finding an optimal partition of the players into subcoalitions, and is therefore NP-hard.

4.3.3 Skill-Based Representations

In many settings, the value of a coalition can be defined in terms of the skills possessed by the agents. A simple representation formalism that is based on this idea has been proposed in [46]: there is a set of *skills* S, each agent $a_i \in A$ has a subset of the skills $S^{a_i} \subseteq S$, and there is a function $u : 2^S \to \mathbb{R}$, which for every subset of skills $S' \subseteq S$ specifies the payoff that can be obtained by a coalition that collectively possesses all the skills in S'. The value of a coalition $C \subseteq A$ is then

$$v(C) = u(\cup_{i:a_i \in C} S^{a_i}).$$

Clearly, this representation is complete, as we can identify each agent a_i with a unique skill s^{a_i} and set $u(S') = v(\{a_i \mid s^{a_i} \in S'\})$ for any subset S' of the skill set. It is succinct when the performance of each coalition can be expressed in terms of a small number of skills possessed by the members of the coalition. Ohta et al. [46] discuss such representations in the context of anonymous environments, where agents can hide skills or split them among multiple identifiers.

A more structured representation was proposed in [5], where coalition values are expressed in terms of skills and tasks. Specifically, in addition to the set of skills S, there is a set of *tasks* Γ, and every task $\tau \in \Gamma$ has a *skill requirement* $S^{\tau} \subseteq S$ and a payoff. As before, each agent $a_i \in A$ has a set of skills $S^{a_i} \subseteq S$. A coalition $C \subseteq A$ *achieves* a task τ if it has all skills that are required for τ, i.e., if $S^{\tau} \subseteq \cup_{i:a_i \in C} S^{a_i}$. Finally, there is a *task value function* $F : 2^{\Gamma} \to \mathbb{R}$, which for every subset $\Gamma' \subseteq \Gamma$ of tasks specifies the payoff that can be obtained by a coalition that achieves all tasks in Γ'. A *coalitional skill game* [4] is then defined as the coalitional game $\langle A, v \rangle$ where:

$$v(C) = F(\{\tau \mid S^{\tau} \subseteq \cup_{i:a_i \in C} S^{a_i}\}).$$

This representation is more compact than that of [46] when the number of skills is large (so that the domain of the function u is very large), but the game can be described in terms of a small number of tasks, or if the function F can be encoded succinctly.

4.3.4 Agent-Type Representation

Shrot et al. [67] and Ueda et al. [71] study coalition formation scenarios where agents can be classified into a small number of *types* so that the agents of the same type are symmetric, i.e., make the same contribution to any coalition they belong to. In such settings, the characteristic function can often be specified more succinctly.

More formally, suppose that the set of agents A admits a partition $\{A^1, \dots, A^T\}$ such that for every $i = 1, \dots, T$, every $a_j, a_k \in A^i$ and every coalition C such that $a_j, a_k \notin C$ it holds that $v(C \cup \{a_j\}) = v(C \cup \{a_k\})$. We will refer to the sets $A^1, \dots, A^T$ as *agent types*. Then the value of any coalition depends solely on how many agents of each type it contains. More precisely, given a coalition $C \subseteq A$, we define the *coalition-type* of C as a vector $\psi = \langle n^1, \dots, n^T \rangle$, where $n^i = |C \cap A^i|$. It is immediate that two coalitions of the same coalition-type have the same value. This means that the conventional characteristic function $v : 2^A \to \mathbb{R}$ can be replaced with the more concise *type-based characteristic function*, $v^t : \Psi \to \mathbb{R}$, which is defined on the set

$$\Psi = \{\langle n^1, \dots, n^T \rangle \mid 0 \le n^i \le |A^i|\}$$

of all possible coalition-types. To represent this function, we only need to store $O(n^T)$ coalitional values, since $|\Psi| = (|A^1| + 1) \times \cdots \times (|A^T| + 1) < n^T$. Thus, for small values of T, this representation is significantly more succinct than the standard one. On the other hand, it is obviously complete: in the worst case, all agents have different types and v^t coincides with v.

5 Coalition Structure Generation

While the focus so far has been on how to *distribute* the gains from cooperation, in this section we focus on how to *maximize* those gains. To state our computational problem formally, we need some additional notation. Recall that $\mathcal{P}^A$ denotes the space of all coalition structures over the set of agents A; we extend this notation to subsets of A, and write $\mathcal{P}^C$ to denote the space of all coalition structures over a set $C \subseteq A$. Given a set $C \subseteq A$ and a coalition structure $CS \in \mathcal{P}^C$, let $V(CS)$ denote the *value* of CS, which is calculated as follows: $V(CS) = \sum_{C' \in CS} v(C')$. The *coalition structure generation* problem is then to find an *optimal* coalition structure $CS^* \in \mathcal{P}^A$, i.e., an (arbitrary) element of the set

$$\text{argmax}_{CS \in \mathcal{P}^A} V(CS).$$

This problem is computationally hard. It resists brute-force search, as the number of possible coalition structures over n players, which is known as the *Bell number* B_n [8], satisfies $\alpha n^{n/2} \le B_n \le n^n$ for some positive constant α (see, e.g., Sandholm et al. [60] for proofs of these bounds and de Bruijn [18] for an asymptotically tight bound). Moreover, it is NP-hard to find an optimal coalition structure given oracle access to the characteristic function [60]. To date, therefore, a number of algorithms have been developed to try and combat this complexity. In what follows, we will present these algorithms and discuss their relative strengths and weaknesses. However, before we do that, we will present the two main representations of the space of the possible coalition structures as they will provide insight into the way some of these algorithms work.

5.1 Space Representation

To date, there are two main representations of the space of possible coalition structures. The first, proposed by Sandholm et al. [60], is called the *coalition structure graph*. In this undirected graph, every node represents a coalition structure. These nodes are categorized into levels $\mathcal{P}_1^A, \ldots, \mathcal{P}_n^A$, where level $\mathcal{P}_i^A$ contains the nodes that represent all coalition structures containing exactly *i* coalitions. An edge connects two coalition structures if and only if: (1) they belong to two consecutive levels $\mathcal{P}_i^A$ and $\mathcal{P}_{i-1}^A$, and (2) the coalition structure in $\mathcal{P}_{i-1}^A$ can be obtained from the one in $\mathcal{P}_i^A$ by merging two coalitions into one. A four-agent example can be seen in Figure 8.1.

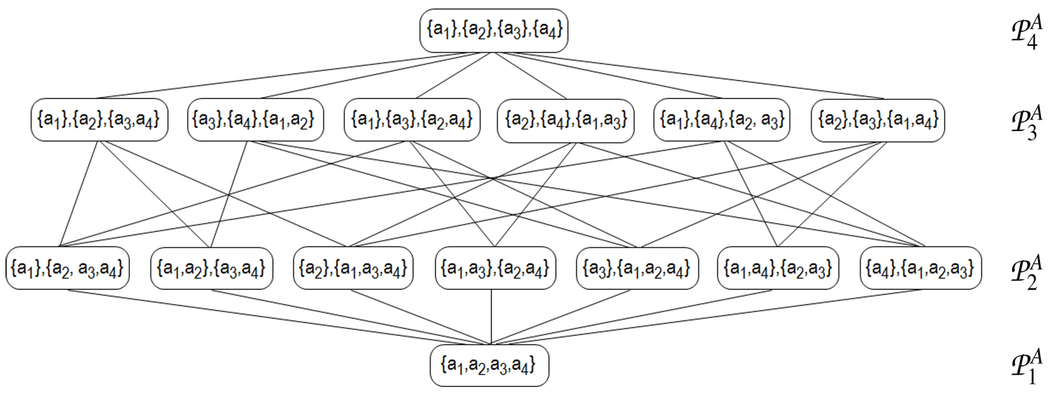
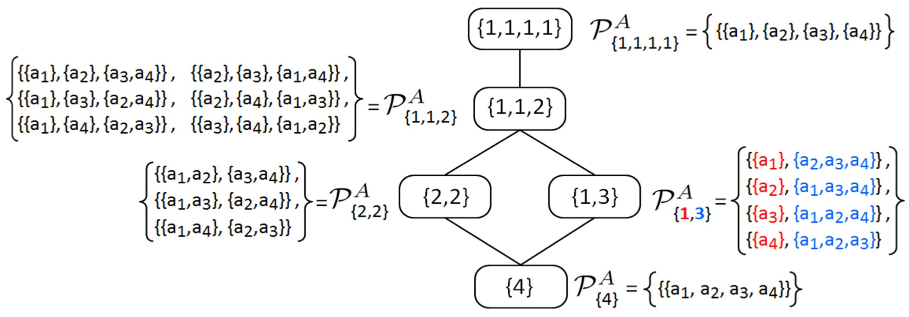


Figure 8.1: The coalition structure graph for four agents.

While the above representation categorizes the coalition structures according to the *number of coalitions* they contain, a different representation was proposed by Rahwan et al. [56] to categorize them based on the *sizes of the coalitions* they contain. More specifically, this representation divides the space of coalition structures into disjoint subspaces that are each represented by an *integer partition* of *n*. Recall that an *integer partition* of *n* is a multiset of positive integers, or *parts*, whose sum (with multiplicities) equals to *n* [1]. For instance, $n = 4$ has five distinct integer partitions, namely, $\{4\}$, $\{1,3\}$, $\{2,2\}$, $\{1,1,2\}$, and $\{1,1,1,1\}$. Each of these partitions corresponds to the subspace of $\mathcal{P}^{\{a_1,a_2,a_3,a_4\}}$, which consists of all the coalition structures within which the coalition sizes match the parts of the integer partition. We denote by $\mathcal{I}^n$ the set of integer partitions of *n*, and by $\mathcal{P}_I^A$ the subspace that corresponds to $I \in \mathcal{I}^n$. For instance, $\mathcal{P}_{\{1,1,2\}}^{\{a_1,a_2,a_3,a_4\}}$ is the subspace containing all the coalition structures consisting of two coalitions of size 1 and one coalition of size 2. This representation can be encoded by an *integer partition* *graph* [52]. This is an undirected graph, where every subspace is represented by a node, and two nodes representing $I, I' \in \mathcal{I}^n$ are connected by an edge if and only if there exist two parts $i, j \in I$ such that $I' = (I \setminus \{i,j\}) \uplus \{i+j\}$ (here $\uplus$ denotes the multiset union operation). For example, Figure 8.2 shows the integer partition graph for four agents, as well as the subspaces that correspond to every node in the graph.



{1,1,1,1}
$\mathcal{P}^A_{\{1,1,1,1\}} = \{\{\{a_1\},\{a_2\},\{a_3\},\{a_4\}\}\}$
{1,1,2}
$\{\{\{a_1\},\{a_2\},\{a_3,a_4\}\}, \{\{a_2\},\{a_3\},\{a_1,a_4\}\}, \{\{a_1\},\{a_3\},\{a_2,a_4\}\}, \{\{a_2\},\{a_4\},\{a_1,a_3\}\}, \{\{a_1\},\{a_4\},\{a_2,a_3\}\}, \{\{a_3\},\{a_4\},\{a_1,a_2\}\}\} = \mathcal{P}^A_{\{1,1,2\}}$
{2,2}
$\{\{\{a_1,a_2\},\{a_3,a_4\}\}, \{\{a_1,a_3\},\{a_2,a_4\}\}, \{\{a_1,a_4\},\{a_2,a_3\}\}\} = \mathcal{P}^A_{\{2,2\}}$
{1,3}
$\mathcal{P}^A_{\{1,3\}} = \{\{\{a_1\},\{a_2,a_3,a_4\}\}, \{\{a_2\},\{a_1,a_3,a_4\}\}, \{\{a_3\},\{a_1,a_2,a_4\}\}, \{\{a_4\},\{a_1,a_2,a_3\}\}\}$
{4}
$\mathcal{P}^A_{\{4\}} = \{\{\{a_1,a_2,a_3,a_4\}\}\}$

Figure 8.2: The integer partition-based representation for four agents.

Having described the main representations of the search space, in the remaining subsections we will present different approaches to the coalition structure generation problem, some of which are built upon those representations.

5.2 Dynamic Programming Algorithms

The first dynamic programming algorithm, called DP, was proposed by Yeh [72]. This algorithm is based on the following theorem.

Theorem 8.4 *Given a coalition $C \subseteq A$, let $f(C)$ be the value of an optimal partition of C, i.e., $f(C) = \max_{P \in \mathcal{P}^C} V(P)$. Then*

$$f(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\left\{ v(C)\ ,\ \max_{\{C',C''\} \in \mathcal{P}^C} \left(f(C') + f(C'') \right) \right\} & \text{otherwise.} \end{cases} \tag{8.4}$$

Proof. The proof is trivial when $|C| = 1$. Thus, for the remainder of the proof we will assume that $|C| > 1$. Let $opt(C)$ be some optimal partition of C, i.e., $opt(C) \in \text{argmax}_{P \in \mathcal{P}^C} V(P)$. We will make use of the following lemma.

Lemma 8.1 *For any coalition $C \subseteq A$, if $P^* = \{P_1, \ldots, P_k\}$ is an optimal partition of C and $k > 1$, then for any $j = 1, \ldots, k$ it holds that $P' = \{P_1, \ldots, P_j\}$ is an* *optimal partition of* $C' = \cup P'$*, and* $P'' = \{P_{j+1}, \dots, P_k\}$ *is an optimal partition of* $C'' = \cup P''$.

Proof of Lemma 8.1 To prove the lemma, observe that $P^* = P' \cup P''$ and $V(P^*) = V(P') + V(P'')$. Suppose for the sake of contradiction that P' was not an optimal partition of C'. Then there exists another partition $\widehat{P'} \in \mathcal{P}^{C'}$ such that $V(\widehat{P'}) > V(P')$. However, since $\widehat{P'} \cup P''$ is a partition of C, and since $V(\widehat{P'} \cup P'') = V(\widehat{P'}) + V(P'') > V(P^*)$, it follows that P^* cannot be an optimal partition of C, a contradiction. Assuming that P'' is not an optimal partition of C'' leads to a contradiction as well, by a similar argument. Thus, the proof of the lemma is complete. ■

Lemma 8.1 shows that if $|opt(C)| > 1$, then there exists a coalition structure $\{C', C''\} \in \mathcal{P}^C$ such that $opt(C) = opt(C') \cup opt(C'')$. On the other hand, if $|opt(C)| = 1$, then surely we would have $opt(C) = \{C\}$ and $V(opt(C)) = v(C)$. Equation (8.4) covers both possibilities by taking the maximum over $v(C)$ and $\max_{\{C',C''\} \in \mathcal{P}^C} \big(f(C') + f(C'')\big)$. ■

The way DP works is by iterating over all the coalitions of size 1, and then over all those of size 2, and then size 3, and so on until size n: for every such coalition C, it computes $f(C)$ using equation (8.4). As can be seen, whenever $|C| > 1$, the equation requires comparing $v(C)$ with $\max_{\{C',C''\} \in \mathcal{P}^C} \big(f(C') + f(C'')\big)$. The result of this comparison is stored in a table, t, which has an entry for every coalition. In particular, if $v(C)$ was greater, then the algorithm sets $t[C] = C$, so that it can later on remember that it is not beneficial to split C into two coalitions. Otherwise, it sets $t[C] = \text{argmax}_{\{C',C''\} \in \mathcal{P}^C} \big(f(C') + f(C'')\big)$ to remember the best way of splitting C into two coalitions. By the end of this process, $f(A)$ will be computed, which is by definition equal to $V(CS^*)$. It remains to compute CS^* itself. This is done recursively using the table t. The running time of this algorithm can be shown to be $O(3^n)$.

The execution of the algorithm is illustrated by the following example.

Example 8.7 *Given* $A = \{a_1, a_2, a_3, a_4\}$*, suppose that* $t[A] = \{\{a_1, a_2\}, \{a_3, a_4\}\}$*, i.e., it is most beneficial to split A into* $\{a_1, a_2\}$ *and* $\{a_3, a_4\}$*. Moreover, suppose that* $t[\{a_1, a_2\}] = \{\{a_1\}, \{a_2\}\}$*, while* $t[\{a_3, a_4\}] = \{a_3, a_4\}$*, i.e., it is most beneficial to split* $\{a_1, a_2\}$ *into* $\{a_1\}$ *and* $\{a_2\}$*, but it is not beneficial to split* $\{a_3, a_4\}$*. In this case,* $CS^* = \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$.

Although DP is guaranteed to find an optimal coalition structure, Rahwan and Jennings [53] showed that many of its operations are in fact redundant. Based on this, they developed an improved dynamic programming algorithm (IDP) that avoids all redundant operations. To date, IDP is the fastest algorithm that can find an optimal solution in $O(3^n)$ time. This is significantly less than $\omega(n^{n/2})$ – the time required to exhaustively enumerate all coalition structures. However, the disadvantage is that IDP provides no interim solution before completion, meaning that it is not possible to trade computation time for solution quality.

5.3 Anytime Algorithms

Generally speaking, an *anytime algorithm* is one whose solution quality improves gradually as computation time increases [73]. In our case, this is particularly desirable as the agents might not always have sufficient time to run the algorithm to completion due to the exponential size of the search space. Moreover, being anytime makes the algorithm robust against failure; if the execution is stopped before the algorithm would normally have terminated, then it can still return a solution that is better than the initial – or any intermediate – one.

In this subsection, we will focus on anytime algorithms that return optimal solutions, or at least provide worst-case guarantees on the quality of their solutions.

5.3.1 Identifying Subspaces with Worst-Case Guarantees

A number of researchers have attempted to answer the following question:

If the solution space is too large to be fully searched, can we search through only a subset of this space, and be guaranteed to find a solution CS^{**} that is within a certain bound β from optimum, that is, $\frac{V(CS^*)}{V(CS^{**})} \leq \beta$?

This problem can be approached by (1) dividing the space into subsets, and (2) identifying a sequence in which these subsets are searched so that the worst-case bound on solution quality is guaranteed to improve after each subset. The first such algorithm was developed by Sandholm et al. [60], and is mainly based on the following theorem.

Theorem 8.5 *To establish a worst-case bound β, it is sufficient to search the lowest two levels of the coalition structure graph, i.e., $\mathcal{P}_1^A$ and $\mathcal{P}_2^A$. With this search, the bound is $\beta = n$, and the number of searched coalition structures is 2^{n-1}. Furthermore, no algorithm can establish any bound by searching a different set of at most 2^{n-1} coalition structures.*

Proof. For a partial search to establish a bound on solution quality, every coalition $C \subseteq A$ must appear in at least one of the searched coalition structures. This is due to the possibility of having a single coalition whose value is arbitrarily greater than the values of other coalitions. Now, since the grand coalition appears in $\mathcal{P}_1^A$, and every other coalition $C \subset A$ appears in $\{C, A \backslash C\} \in \mathcal{P}_2^A$, the value of the best coalition structure in $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ is at least $\max_{C \subseteq A} v(C)$. On the other hand, since CS^* can include at most n coalitions, its value cannot be greater than $n \times \max_{C \subseteq A} v(C)$. This means $\frac{V(CS^*)}{\max_{CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A} V(CS^*)} \leq n$.

As for the number of searched coalition structures, the reader can check that $\left|\mathcal{P}_1^A \cup \mathcal{P}_2^A\right| = 2^{n-1}$. What remains is to show that no bound can be established by searching a different set of at most 2^{n-1} coalition structures. This is done by proving that $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ is the unique subset of $\mathcal{P}^A$ of size at most 2^{n-1} in which every coalition appears in some coalition structure. We leave this as an exercise for the reader. ■

Based on this theorem, the algorithm starts by searching the bottom two levels. After that, if additional time is available, the algorithm searches the remaining levels one by one, starting from the top level and moving downward. Sandholm et al. proved that the bound improves with this search. In particular, once the algorithm completes searching level $\mathcal{P}_i^A$, the bound becomes $\beta = \lfloor n/h \rfloor$, where $h = \lfloor (n-i)/2 \rfloor + 2$. The only exception is when $n \equiv h - 1 (mod\ h)$ and $n \equiv i (mod\ 2)$, in which case the bound becomes $\beta = \lceil n/h \rceil$. Importantly, this means that after searching the bottom two levels and establishing the bound $\beta = n$, one can very easily drop (i.e., improve) the bound to $\beta = \lceil n/2 \rceil$ by searching the top level, which only contains one coalition structure.

A different approach was proposed by Dang and Jennings [16]. Their algorithm starts by searching the bottom two levels, as well as the top one (as Sandholm et al.'s algorithm does). After that, however, instead of searching the remaining levels one by one (as Sandholm et al. do), the algorithm searches through certain subsets of all remaining levels. Specifically, it searches the coalition structures that have at least one coalition of size at least $\lceil n(d-1)/d \rceil$ (with d running from $\lfloor (n+1)/4 \rfloor$ down to 2). Dang and Jennings proved that, for any given value of d, the algorithm establishes a bound of $2d - 1$.

So far, we have seen how certain bounds can be established by searching certain subsets of the search space. However, with the exception of $\beta = n$ and $\beta = \lceil n/2 \rceil$, we still do not know the *minimum* subset that must be searched in order to establish a desired bound. To this end, let us introduce the following notation. For any integer partition $I \in \mathcal{I}^n$, let $\mathcal{P}^I$ denote the set of possible partitions of I. For instance, $\mathcal{P}^{\{1,1,2\}}$ consists of the following four partitions: $\{\{1,1,2\}\}$, $\{\{1,1\},\{2\}\}$, $\{\{1,2\},\{1\}\}$, and $\{\{1\},\{1\},\{2\}\}$. Moreover, for any set of integer partitions $\mathcal{I}' \subseteq \mathcal{I}^n$, let $\mathcal{S}(\mathcal{I}')$ be the set that consists of every non-empty subset of every integer partition in $\mathcal{I}'$, i.e.,

$$\mathcal{S}(\mathcal{I}') = \bigcup_{I \in \mathcal{I}'} \bigcup_{J \subseteq I, J \neq \emptyset} \{J\}.$$

For example, given $\mathcal{I}' = \{\{1,1,2\},\{1,3\}\}$, the set $\mathcal{S}(\mathcal{I}')$ consists of the following subsets: $\{\{1\}\}$, $\{\{2\}\}$, $\{\{3\}\}$, $\{\{1,1\}\}$, $\{\{1,2\}\}$, $\{\{1,3\}\}$, $\{\{1,1,2\}\}$. Finally, for any integer partition $I \in \mathcal{I}^n$ and any set of integer partitions $\mathcal{I}' \subseteq \mathcal{I}^n$, let $\eta(\mathcal{I}',I)$ denote the minimum number of subsets in $\mathcal{S}(\mathcal{I}')$ that are required to construct a partition in $\mathcal{P}^I$. Formally,

$$\eta(I,\mathcal{I}') = \begin{cases} \min_{S \subseteq \mathcal{S}(\mathcal{I}'):S \in \mathcal{P}^I} |S| & \text{if } \exists S \subseteq \mathcal{S}(\mathcal{I}') : S \in \mathcal{P}^I \\ \\ +\infty & \text{otherwise.} \end{cases}$$

For example, given $I = \{1,1,1,3\}$ and $\mathcal{I}' = \{\{1,1,2\},\{1,3\}\}$, the minimum number of subsets in $\mathcal{S}(\mathcal{I}')$ that are required to construct a partition of I is 2, and those subsets are $\{1,1\}$ and $\{1,3\}$. Therefore, we have $\eta(\mathcal{I}',I) = 2$. Rahwan et al. [55] showed that this definition is crucial when determining the minimum subset that must be searched in order to establish a certain bound. Specifically, they prove the following theorem.

Theorem 8.6 *For any real value b, $1 \leq b \leq n$, and for any $\mathcal{I}' \subseteq \mathcal{I}^n$, we can establish a bound $\beta = b$ by searching $\cup_{I \in \mathcal{I}'} \mathcal{P}_I^A$ if and only if the following holds:*

$$\forall I \in \mathcal{I}^n, \eta(\mathcal{I}',I) \leq b. \tag{8.5}$$

Furthermore, the minimum set of coalition structures that must be searched in order to establish a bound $\beta = b$ is $\cup_{I \in \mathcal{I}^n(b)} \mathcal{P}_I^A$, where $\mathcal{I}^n(b)$ is defined as follows:

$$\mathcal{I}^n(b) \in \operatorname*{arg\,min}_{\mathcal{I}' \subseteq \mathcal{I}^n : \forall I \in \mathcal{I}^n, \eta(\mathcal{I}',I) \leq b} \left| \cup_{I \in \mathcal{I}'} \mathcal{P}_I^A \right|.$$

In other words, to establish a bound $\beta = b$, all we need to do is to find a set of integer partitions $\mathcal{I}' \subseteq \mathcal{I}^n$ such that, if we take every possible subset of every $I \in \mathcal{I}'$, then with these subsets we can partition every $I \in \mathcal{I}^n$ into at most b parts. One can optimize this by looking for the set of integer partitions that minimizes $\left|\cup_{I \in \mathcal{I}'} \mathcal{P}_I^A\right|$.

We omit the proof of Theorem 8.6 due to space constraints. However, the intuition is similar to the proof of Theorem 8.5, where we proved that a bound $\beta = n$ can be established by searching $\mathcal{P}_1^A \cup \mathcal{P}_2^A$. This was done by showing that CS^* contains at most n coalitions, and that every possible coalition appears in some $CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A$. The proof of Theorem 8.6 generalizes this idea by replacing *"coalitions"* with *"combinations of coalitions"*. More specifically, equation (8.5) means that CS^* contains at most b combinations, and that every one of those combinations appears in some $CS \in \cup_{I \in \mathcal{I}'} \mathcal{P}_I^A$.

Theorem 8.6 enables us to describe the set to be searched when establishing a given bound in terms of subspaces that are represented by integer partitions.

Therefore, it would be useful to have an algorithm that can efficiently search those subspaces. In what follows, we present an algorithm that does exactly that.

5.3.2 Integer Partition-Based Search

An anytime algorithm, called IP, was developed by Rahwan et al. [58] based on the integer partition-based representation from Section 5.1. In particular, it uses the observation that, for any subspace $\mathcal{P}_I^A$, it is possible to compute upper and lower bounds on the value of the best coalition structure in that subspace. More formally, let Max_s^A and Avg_s^A be the maximum and average values of all coalitions of size s, respectively. It turns out that one can compute the average value of the coalition structures in $\mathcal{P}_I^A$ without inspecting these coalition structures [58].

Theorem 8.7 *For any $I \in \mathcal{I}^n$, let $I(i)$ be the multiplicity of i in I. Then:*

$$\frac{\sum_{CS\in\mathcal{P}_I^A} V(CS)}{\left|\mathcal{P}_I^A\right|} = \sum_{i\in I} I(i)\cdot Avg_i^A. \tag{8.6}$$

Proof. For any $C \subseteq A$, the number of coalition structures in $\mathcal{P}_I^A$ that contain C depends solely on the size of C. In other words, this number is equal for any two coalitions that are of the same size. Let us denote this number by $\mathcal{N}_I^{|C|}$. Formally, for every $C \subseteq A$ we set $\mathcal{N}_I^{|C|} = \left|\{CS \in \mathcal{P}_I^A \mid C \in CS\}\right|$. Then we have

$$\sum_{CS\in\mathcal{P}_I^A} V(CS) = \sum_{i\in I}\sum_{C:|C|=i} \mathcal{N}_I^i \cdot v(C) = \sum_{i\in I}\mathcal{N}_I^i \sum_{C:|C|=i} v(C) = \sum_{i\in I}\mathcal{N}_I^i\cdot\binom{n}{i}\cdot Avg_i^A,$$

where $\binom{n}{i}$ is the binomial coefficient (i.e., the number of possible coalitions of size i). Thus, to prove (8.6) it suffices to prove that

$$\frac{\sum_{i\in I}\mathcal{N}_I^i\cdot\binom{n}{i}\cdot Avg_i^A}{\left|\mathcal{P}_I^A\right|} = \sum_{i\in I} I(i)\cdot Avg_i^A.$$

This can be done by proving that the following holds for all $i \in I$:

$$\mathcal{N}_I^i\cdot\binom{n}{i} = I(i)\cdot\left|\mathcal{P}_I^A\right|. \tag{8.7}$$

Observe that every $CS \in \mathcal{P}_I^A$ contains exactly $I(i)$ coalitions of size i. Thus:

$$\sum_{C:|C|=i}\mathcal{N}_I^{|C|} = \sum_{C:|C|=i}\;\sum_{CS\in\mathcal{P}_I^A:C\in CS} 1 = \sum_{CS\in\mathcal{P}_I^A}\;\sum_{C\in CS:|C|=i} 1 = \sum_{CS\in\mathcal{P}_I^A} I(i) = |\mathcal{P}_I^A|\cdot I(i).$$

We have shown that $\sum_{C:|C|=i} \mathcal{N}_I^{|C|} = |\mathcal{P}_I^A| \cdot I(i)$. On the other hand, since $\mathcal{N}_I^{|C|}$ is equal for all coalitions of size $|C|$, we obtain $\sum_{C:|C|=i} \mathcal{N}_I^{|C|} = \binom{n}{i} \cdot \mathcal{N}_I^i$. Thus, equation (8.7) holds. ■

Based on this theorem, for every $I \in \mathcal{I}^n$, it is possible to compute a lower bound LB_I on the value of the best coalition structure in $\mathcal{P}_I^A$ as follows: $LB_I = \sum_{s \in I} I(s) Avg_s^A$. This is simply because the best value is always greater than, or equal to, the average one. Similarly, it is possible to compute an upper bound UB_I on the value of the best coalition structure in $\mathcal{P}_I^A$ as $UB_I = \sum_{s \in I} I(s) Max_s^A$. Using these bounds, the algorithm computes an upper bound $UB^* = \max_{I \in \mathcal{I}^n} UB_I$ and a lower bound $LB^* = \max_{I \in \mathcal{I}^n} LB_I$ on the value of the optimal coalition structure CS^*. Computing UB^* allows for establishing a bound on the quality of the best coalition structure found at any point in time, denoted CS^{**}; this bound is $\beta = UB^*/V(CS^{**})$. On the other hand, computing LB^* allows for identifying subspaces that have no potential of containing an optimal coalition structure, which are $\mathcal{P}_I^A : UB_I < LB^*$. These subspaces are pruned from the search space. As for the remaining subspaces, the algorithm searches them one at a time, unless a coalition structure is found that has a value greater than, or equal to, the upper bound of some subspace, in which case that subspace no longer needs to be searched. Searching a subspace is done using an efficient process that applies a *branch-and-bound* technique to avoid examining every coalition structure in that subspace whenever possible. A distributed version of IP has also been developed, see [43] for more details.

The IP algorithm can, in the worst case, end up searching the entire space, i.e., it runs in $O(n^n)$ time. In practice, however, IP has been shown to be significantly faster than IDP given popular coalition-value distributions, and the bound that it generates, i.e., $\beta = UB^*/V(CS^{**})$, has been shown to be significantly better than those obtained by searching particular subsets as per the previous subsection.

An extended version of IP, called IDP-IP, was developed by Rahwan and Jennings [52]. As the name suggests, this algorithm is a combination of IDP and IP; it is based on the observation that IDP, even if not run to completion, can still provide useful information. Thus, the basic idea is to partially run IDP, and then use a modified version of IP that can continue the search from where IDP has stopped. This results in a hybrid performance that can be controlled by simply setting the point at which IDP stops. In so doing, one can control the trade-off between the desired features of both IDP and IP. For more details, see [52].

5.3.3 Integer Programming

A different anytime approach, compared to what we have discussed so far, is to formulate the coalition structure generation problem as an integer program. More specifically, let $C_1, C_2, \dots, C_{2^n}$ denote the possible coalitions. Let z be an $n \times 2^n$ binary matrix, where every row represents an agent and every column represents a coalition, so that $z_{i,j} = 1$ if and only if $a_i \in C_j$. Finally, let us have 2^n decision variables, $x_1, x_2, \dots, x_{2^n}$, where $x_j = 1$ corresponds to C_j being selected in the solution. The coalition structure generation problem can then be modeled as:

$$\begin{array}{lrcll} \max & \sum_{j=1,\dots,2^n} v(C_j) \cdot x_j & & & \text{subject to:} \\ & \sum_{j=1,\dots,2^n} z_{i,j} \cdot x_j & = & 1 & \text{for } i = 1,2,\dots,n \\ & x_j & \in & \{1,0\} & \text{for } j = 1,2,\dots,2^n \end{array}$$

With this formulation, it is possible to apply any integer programming solver. However, this approach has been shown to be inefficient, e.g., even an industrial-strength solver such as ILOG's CPLEX was shown to be significantly slower than both IDP and IP, and quickly runs out of memory as the number of agents increases [57].

5.4 Metaheuristic Algorithms

In all the algorithms that were presented so far, the focus was on finding an optimal solution, or a solution that is within a bound from optimum. However, as the number of agents increases, the problem becomes too hard, and the only practical option would be to use metaheuristic algorithms. Such algorithms do not guarantee that an optimal solution is ever found, nor do they provide any guarantees on the quality of their solutions. However, they can usually be applied for very large problems. Next, we outline some of these algorithms.

Sen and Dutta [63] developed a genetic algorithm for coalition structure generation. This algorithm starts with an initial, randomly generated set of coalition structures, called a *population*. After that, the algorithm repeats the following three steps: (1) evaluation, (2) selection, and (3) recombination. More specifically, the algorithm evaluates every member of the current population, selects members based on the outcome of the evaluation, and constructs new members from the selected ones by exchanging and/or modifying their contents.

Keinänen [35] proposed an algorithm based on Simulated Annealing – a generic stochastic local search technique. At each iteration, the algorithm moves from the current coalition structure to a coalition structure in its neighborhood, where neighborhoods can be defined using a variety of criteria. More specifically, the algorithm starts by generating a random coalition structure CS. Then, at every iteration, it samples a random coalition structure CS' in the neighborhood of CS. If CS' is better than CS, then the algorithm sets $CS = CS'$. Otherwise, it sets

$CS = CS'$ with a probability $e^{\frac{V(CS')-V(CS)}{\tau}}$, where τ is the *temperature* parameter that decreases after each iteration according to an *annealing schedule* $\tau = \alpha\tau$, where $0 < \alpha < 1$.

A decentralized, greedy algorithm was proposed by Shehory and Kraus [66]. This algorithm ignores coalitions containing more than a certain number of agents. It returns a coalition structure *CS* that is constructed iteratively in a greedy manner; at every iteration, the best of all candidate coalitions is added to *CS*, where a candidate coalition is one that does not overlap with any of the coalitions that were added to *CS* in previous iterations. The search for the best candidate coalition is done in a distributed fashion; the agents negotiate over which one of them searches which coalitions. A significantly improved distribution mechanism was later on proposed in [51].

Another greedy algorithm, which was put forward by Di Mauro et al. [42], is based on GRASP – a general purpose greedy algorithm, which after each iteration performs a quick local search to try and improve its solution [28]. In the coalition structure generation version of GRASP, a coalition structure *CS* is constructed iteratively. Every iteration consists of two steps. The first step is to add the best candidate coalition to *CS*, resulting in a set of pairwise disjoint, but not necessarily exhaustive, coalitions, i.e., $\cup CS \subseteq A$. The second step is to explore different neighborhoods of *CS*. These two steps are repeated until $\cup CS = A$. Furthermore, the whole process of constructing *CS* is repeated over and over to try and find better solutions. This algorithm has been shown to work particularly well, with empirical results suggesting that it is the best metaheuristic algorithm for coalition structure generation to date.

5.5 Coalition Structure Generation under Compact Representations

So far, we focused on the coalition structure generation problem under the characteristic function representation (where the input consists of a value for every possible coalition). In what follows, we briefly discuss several papers that consider alternative, often more concise, representations.

5.5.1 Distributed Constraint Optimization

The Distributed Constraint Optimization Problem (DCOP) framework has recently become a popular approach for modeling cooperative agents [44]. In this framework: (1) each agent has a choice of actions, (2) reward is determined by the combination of actions, and (3) the goal is for every agent to choose an action so as to maximize the sum of the rewards. Ueda et al. [70] considered the coalition structure generation problem where the multiagent system is represented as one big DCOP, and every coalition's value is computed as the optimal solution of the DCOP among the agents of that coalition.

At first glance, this might seem too computationally expensive since there are 2^n possible coalitions. Thus, to find the optimal coalition structure, one might need to solve 2^n instances of the NP-hard DCOP problem. Interestingly, however, Ueda et al. showed that the process of finding an optimal, or near optimal, coalition structure does not have to be divided into two independent stages: (1) computing all coalition values, and (2) finding an optimal combination of disjoint and exhaustive coalitions. Instead, the big DCOP that represents the multiagent system can be modified so that those two stages are merged. This means the desired coalition structure can be obtained by solving a single, modified DCOP.

The modification is controlled by a single parameter, called σ, which specifies the maximum number of coalitions that are allowed to contain more than one agent. We will call these *multiagent* coalitions. The basic idea behind the modification is to change every agent's *domain*, i.e., set of possible actions. Specifically, every action d_j in the original domain is replaced by σ actions, $d_{j,1},\dots,d_{j,\sigma}$, where $d_{j,i}$ means that the agent performs action d_j while joining the i^{th} multiagent coalition. The new domain also contains an action called *"independent"*, which means that the agent acts independently. The modified DCOP can be solved using any existing algorithm that can obtain an optimal solution, e.g., ADOPT [44] or DPOP [49]. Assuming that the original number of possible actions per agent is d, the search space size for the original DCOP is d^n, while for the modified DCOP it is $(\sigma d+1)^n$. The following theorem implies that the optimal solution of the modified DCOP is within a bound $\beta = \left\lfloor \frac{n}{2} \right\rfloor / \sigma$ from optimum.

Theorem 8.8 *Let $\mathcal{I}^n_k \subseteq \mathcal{I}^n$ be a set in which every integer partition contains at most k integers that are greater than 1. Then, the best coalition structure in $\cup_{I\in\mathcal{I}^n_k}\mathcal{P}^A_I$ is within a bound $\beta = \left\lfloor \frac{n}{2} \right\rfloor / k$ from optimum.*

Proof. Assume that CS^* contains ℓ multiagent coalitions, where $\ell > k$. Let $C_1,\dots,C_{\ell-k}$ be the $\ell-k$ coalitions with the smallest values in CS^*. Let us split each coalition C_i, $i=1,\dots,\ell-k$, into single-agent coalitions; denote the resulting coalition structure by CS'_k. Clearly, $CS'_k \in \cup_{I\in\mathcal{I}^n_k}\mathcal{P}^A_I$. Furthermore, the total value of $C_1,\dots,C_{\ell-k}$ is at most $\frac{\ell-k}{\ell}V(CS^*)$, and the values of the single-agent coalitions are non-negative. Hence, we have $V(CS'_k) \geq \frac{k}{\ell}V(CS^*)$. It remains to observe that $\ell \leq \left\lfloor \frac{n}{2} \right\rfloor$. ■

5.5.2 Marginal Contribution Nets

Ohta et al. [45] studied the coalition structure generation problem under the basic MC-net representation (see Section 4.3.1). Recall that a basic MC-net rule can be written as $(P_r, N_r) \to \vartheta_r$: the interpretation is that a coalition that contains all agents in P_r and none of the agents in N_r can earn a profit of ϑ_r. Ohta et al. consider a restricted class of basic MC-nets, where for each r we have $P_r \neq \emptyset$ and $\vartheta_r > 0$; it can be shown that any characteristic function can be represented by such a restricted MC-net. They define a set of rules $\mathcal{R}' \subseteq \mathcal{R}$ to be *feasible* if all the rules in $\mathcal{R}'$ are applicable *at the same time* to some coalition structure. In other words, $\mathcal{R}'$ is feasible if there exists a coalition structure CS such that every rule $r \in \mathcal{R}'$ is applicable to some $C \in CS$. The problem of finding an optimal coalition structure is then equivalent to the problem of finding a feasible set of rules $\mathcal{R}'$ such that $\sum_{r \in \mathcal{R}'} \vartheta_r$ is maximized. While this problem is NP-hard, Ohta et al. showed that it admits a mixed integer programming (MIP) formulation. Their MIP is based on the observation that, for any two rules r, r', the possible relations between r and r' can be classified into the following four cases:

- **Compatible on different coalitions (CD):** This is when $P_r \cap P_{r'} = \emptyset$ and ($P_r \cap N_{r'} \neq \emptyset$ or $P_{r'} \cap N_r \neq \emptyset$). For example, $(\{a_1, a_2\}, \emptyset) \to \vartheta_1$ and $(\{a_3, a_4\}, \{a_1\}) \to \vartheta_2$ are applicable at the same time in some CS as long as a_1, a_2 appear in a coalition $C \in CS$ and a_3, a_4 appear in a different coalition $C' \in CS$.

- **Incompatible (IC):** This is when $P_r \cap P_{r'} \neq \emptyset$ and ($P_r \cap N_{r'} \neq \emptyset$ or $P_{r'} \cap N_r \neq \emptyset$). For example, $(\{a_1, a_2\}, \emptyset) \to \vartheta_1$ and $(\{a_2, a_3\}, \{a_1\}) \to \vartheta_2$ are not applicable at the same time, because the first requires a_1 and a_2 to appear together in a coalition, while the second requires a_2 and a_3 to appear together in a coalition that does not contain a_1.

- **Compatible on the same coalition (CS):** This is when $P_r \cap P_{r'} \neq \emptyset$ and $P_r \cap N_{r'} = P_{r'} \cap N_r = \emptyset$. For example, $(\{a_1, a_2\}, \emptyset) \to \vartheta_1$ and $(\{a_2, a_3\}, \{a_4\}) \to \vartheta_2$ are applicable at the same time to some coalition structure CS as long as there exists $C \in CS$ such that $\{a_1, a_2, a_3\} \subseteq C$ and $a_4 \notin C$. Note that both rules apply to the same coalition.

- **Independent (ID):** This is when $P_r \cap P_{r'} = P_r \cap N_{r'} = P_{r'} \cap N_r = \emptyset$.

Consider a graphical representation of an MC-net in which every node is a rule, and between any two nodes there exists an edge whose type is one of the four cases described above. Then, the following holds:

Theorem 8.9 *A set of rules $\mathcal{R}'$ is feasible if and only if (1) it includes no pair of rules that are connected by an edge of type* **IC**, *and (2) for any two rules in $\mathcal{R}'$ that are connected by an edge of type* **CD**, *it is not possible to reach one from the other via a series of edges of type* **CS**.

To understand the intuition behind the proof, consider an example of three rules, r_1, r_2, r_3. Suppose that for $i = 1,2,3$ we have $r_i = (P_i, N_i) \to \vartheta_i$, where $P_1 = \{a_1, a_2\}$, $N_1 = \emptyset$, $P_2 = \{a_2, a_3\}$, $N_2 = \emptyset$, and $P_3 = \{a_3, a_4\}$, $N_3 = \{a_1\}$. Here, r_1 and r_2 are connected by an edge of type **CS**. Thus, they must be applicable to a single coalition in CS, say C', such that $P_1 \cup P_2 \subseteq C'$. Similarly, an edge of type **CS** connects r_2 and r_3, and so they must be applicable to a single coalition in CS, say C'', such that $P_2 \cup P_3 \subseteq C''$. Now, since $P_1 \cup P_2$ overlaps with $P_2 \cup P_3$, and since the coalitions in CS are pairwise disjoint, we must have $C' = C''$. This means that r_1, r_2, r_3 must all be applicable to the same coalition, i.e., the edge between r_1 and r_3 must *not* be of the type **IC** or **CD**. However, in our example, we happen to have an edge of type **CD** between r_1 and r_3. Therefore, any rule set containing r_1, r_2, r_3 is not feasible.

Based on Theorem 8.9, Ohta et al. proposed the following MIP formulation.

$$\max \sum_{r \in R} \vartheta_r \cdot x_r \quad \text{subject to:}$$

$$\begin{aligned}
x_{r_i} + x_{r_j} &\leq 1 \quad \text{for each edge } (r_i, r_j) \text{ of type } \mathbf{IC} && (8.8)\\
y^e_{r_i} &= 0 \quad \text{for each edge } e = (r_i, r_j) \text{ of type } \mathbf{CD} \text{ with } j > i && (8.9)\\
y^e_{r_j} &\geq 1 \quad \text{for each edge } e = (r_i, r_j) \text{ of type } \mathbf{CD} \text{ with } j > i && (8.10)\\
y^e_{r_k} &\leq y^e_{r_\ell} + (1 - x_{r_k}) + (1 - x_{r_\ell})\\
&\qquad \text{for each edge } (r_k, r_\ell) \text{ of type } \mathbf{CS} && (8.11)\\
y^e_{r_\ell} &\leq y^e_{r_k} + (1 - x_{r_k}) + (1 - x_{r_\ell})\\
&\qquad \text{for each edge } (r_k, r_\ell) \text{ of type } \mathbf{CS} && (8.12)\\
x_r &\in \{0,1\} \text{ for each } r \in R
\end{aligned}$$

Here, we have a binary variable x_r for every rule r, where $x_r = 1$ means that r is selected in the solution. Thus, condition (1) of Theorem 8.9 is enforced by the constraint (8.8), which ensures that two rules connected by an edge of type **IC** are never selected at the same time. Moreover, for every edge e of type **CD** or **CS** and every rule r that is adjacent to this edge we define a variable y^e_r. These variables are used in constraints (8.9)–(8.12) to enforce condition (2) of Theorem 8.9. In more detail, for every edge $e = (r_i, r_j)$ of type **CD** with $j > i$ constraints (8.9) and (8.10) ensure that $y^e_{r_i} \neq y^e_{r_j}$. Furthermore, for every edge (r_k, r_ℓ) of type **CS** the constraints (8.10) and (8.11) ensure that, if both r_k and r_ℓ are selected, then $y^e_{r_k} = y^e_{r_l}$. Thus, by enforcing both conditions of Theorem 8.9, we guarantee that every solution to this MIP is a feasible rule set.

5.5.3 Coalitional Skill Games

Bachrach et al. [4] considered the coalition structure generation problem in *coalitional skill games* (see Section 4.3.3). While this problem is, in general, very hard computationally, Bachrach et al. showed that it admits an efficient algorithm as long as the number of tasks m and the *treewidth* of a certain associated hypergraph are small. To describe their algorithm, we need a few additional definitions.

Given a skill game with a skill set S, its *skill graph* is a hypergraph $g = \langle V, E \rangle$ in which every agent corresponds to a vertex, and every skill $s_i \in S$ is represented as a hyperedge $e_{s_i} \in E$ that connects all agents that possess this skill. The "complexity" of a hypergraph can be measured using the notion of *treewidth*. The following definition is reproduced from [30] (an illustration is provided in Figure 8.3).

Definition 8.13 *Given a hypergraph $g = \langle V, E \rangle$, a* tree decomposition *of g is a tuple $(Q, \mathbf{B})$, where $\mathbf{B}$ is a family of subsets of V (each such subset $B_i \in \mathbf{B}$ is called a* bag*), and Q is a tree whose node set is $\mathbf{B}$ such that: (1) for each $e \in E$ there is a bag $B_i \in \mathbf{B}$ such that $e \in B_i$; (2) for each $v_j \in V$ the set $\{B_i \in \mathbf{B} \mid v_j \in B_i\}$ is non-empty and connected in Q. The* width *of $(Q, \mathbf{B})$ is* $\max_{B_i \in \mathbf{B}} |B_i| - 1$. *The* treewidth *of g is the minimum width of $(Q, \mathbf{B})$ over all possible tree decompositions $(Q, \mathbf{B})$ of g.*

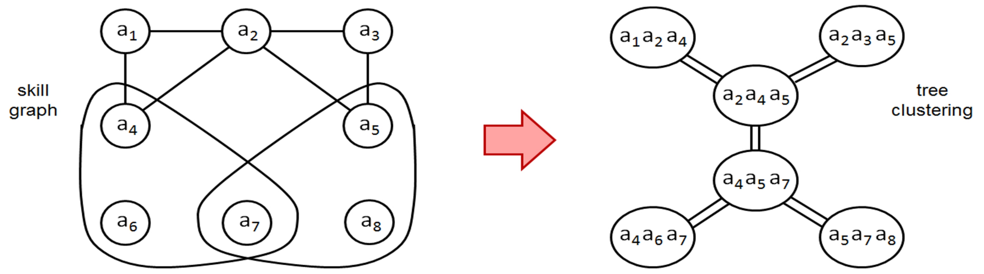


Figure 8.3: A skill graph and its tree decomposition with width 2.

Let $CSG(m, w)$ be the class of all coalitional skill games where the number of tasks is at most m and the treewidth of the corresponding skill graph is at most w. We will now show that, for fixed m and w, the coalition structure generation problem for a game in $CSG(m, w)$ can be solved in time polynomial in the number of agents n and the number of skills k (but exponential in m and w).

To start, observe that a single task can be performed multiple times by a single coalition structure CS. To be more precise, a task that requires a skill which only x agents share can be performed at most x times (this is when each one of those

x agents appears in a different coalition in CS). Let d denote the largest number of agents sharing a single skill; note that $d \leq w+1$. Then a coalition structure can accomplish at most dm tasks. Based on this, we will define a *candidate task solution* as a set $\{\Gamma_i\}_{i=1}^h$ where each Γ_i is a subset of Γ, and $h \leq dm$. For every coalition structure $CS = \{C_i\}_{i=1}^h$, we say that *CS accomplishes* $\{\Gamma_i\}_{i=1}^h$ if C_i accomplishes all tasks in Γ_i, for $i = 1, \ldots, h$. We say that $\{\Gamma_i\}_{i=1}^h$ is *feasible* if there exists at least one coalition structure that accomplishes it. Clearly, the total value obtained by accomplishing these tasks is $\sum_{i=1}^h F(\Gamma_i)$. The problem of finding an optimal coalition structure is thus equivalent to the problem of finding a feasible set of task subsets that maximizes $\sum_{i=1}^h F(\Gamma_i)$. To solve this problem, it is sufficient to iterate over all possible choices of $\{\Gamma_i\}_{i=1}^h$: for each such choice we find the coalition structure that accomplishes it, or determine that it is not feasible. Next, we show how this can be done for a fixed set $\{\Gamma_i\}_{i=1}^h$ in time polynomial in n and k; the bound on the running time follows as the number of candidate task solutions is bounded by $(2^m)^{dm} \leq (2^m)^{(w+1)m}$.

To this end, observe that every coalition structure can be viewed as a *coloring* of the agents, where all agents with the same color form a coalition. Based on this, for each choice of $\{\Gamma_i\}_{i=1}^h$, let us define a *constraint satisfaction problem*[2] whose underlying graph is *the skill graph g*, where:

- the **variables** correspond to the agents;
- the **domain** (i.e., the possible values) of each variable (i.e., agent) consists of the possible colors (i.e., the possible coalitions that the agent can join);
- For each skill s, we have the following **constraint**: *For each $i = 1, \ldots, h$, if some task in Γ_i requires s, then at least one agent in C_i possesses s.*

To solve this *"primal"* constraint satisfaction problem, we first check if the treewidth of g is bounded by w, and if so return a tree decomposition (this can be done in time polynomial in n and k, see [30]). Then, to solve the primal problem, we define a *"dual"* problem. This is another constraint satisfaction problem whose underlying graph is the tree decomposition of g and

- the **variables** correspond to the bags in the tree decomposition;
- the **domain** of every bag consists of the possible colorings of the agents in the bag. The size of this domain is $O(h^{w+1}) = O(((w+1)m)^{w+1})$ since every bag contains at most $w+1$ agents, and every agent has h possible colors;

[2]For more details on constraint satisfaction problems, see [59].

- the **constraints** are of two types. The first prevents an agent from getting different colors in two neighboring bags. This, in turn, ensures that every agent gets the same color in *all* bags (due to the structure of the tree decomposition). The second type of constraints is exactly the same as the one in the primal problem (i.e., *if a skill is required for at least one task in Γ_i, then at least one agent in C_i possesses that skill*).

Note that a solution to the dual problem is in fact a valid solution to the primal problem. Since the underlying graph of the dual problem is a tree, it can be solved in time polynomial in n and k [4, 59].

5.5.4 Agent-Type Representation

Aziz and de Keijzer [3] and Ueda et al. [71] studied the coalition structure generation problem under the agent-type representation (see Section 4.3.4). Recall that under this representation the game is given by a partition of the set of agents A into T types $A^1,\dots,A^T$ and a type-based characteristic function $v^t : \Psi \to \mathbb{R}$, where $\Psi = \{\langle n^1,\dots,n^T\rangle \mid 0 \le n^i \le |A^i|\}$. Thus, a coalition structure can be viewed as a partition of $\langle |A^1|,\dots,|A^T|\rangle$. Formally, we have the following definition.

Definition 8.14 *A* type-partition *of a coalition-type $\psi = \langle n^1,\dots,n^T\rangle$ is a set of coalition-types $\lambda = \{\langle n_i^1,\dots,n_i^T\rangle\}_{i=1}^{\ell}$ such that $\langle \sum_{i=1}^{\ell} n_i^1,\dots,\sum_{i=1}^{\ell} n_i^T\rangle = \psi$. The* value *of λ is computed as $V^t(\lambda) = \sum_{i=1}^{\ell} v^t(\langle n_i^1,\dots,n_i^T\rangle)$.*

For example, $\{\langle 0,1,2\rangle,\langle 4,3,2\rangle\}$ is one of the possible type-partitions of $\langle 4,4,4\rangle$, and $V^t(\{\langle 0,1,2\rangle,\langle 4,3,2\rangle\}) = v^t(\langle 0,1,2\rangle) + v^t(\langle 4,3,2\rangle)$.

Thus, while we typically deal with "coalitions" and "coalition structures," in an agent-type representation we deal with "coalition-types" and "type-partitions." The problem of finding an optimal coalition structure is then equivalent to that of finding an optimal type-partition of $\langle |A^1|,\dots,|A^T|\rangle$. For example, if we have four types and five agents of each type, we need to find an optimal type-partition of $\langle 5,5,5,5\rangle$. Two dynamic programming algorithms were proposed for this problem; both run in $O(n^{2T})$ time [3, 71]. We will present the one given in [3], since it is easier to describe.

For any coalition-type $\psi \in \Psi$, let us denote by $f^t(\psi)$ the value of the optimal type-partition of ψ. Then, we can compute $f^t(\psi)$ recursively as follows [3]:

$$f^t(\psi) = \begin{cases} 0 & \text{if } n^i = 0 \text{ for } i = 1,\dots,T \\ \max\{f^t(\langle n^1 - x^1,\dots,n^T - x^T\rangle) + v^t(\langle x^1,\dots,x^T\rangle) & \\ \qquad \mid x^i \le n^i \text{ for } i = 1,\dots,T\} & \text{otherwise.} \end{cases} \tag{8.13}$$

Based on this recursive formula, we can compute the optimal type-partition by dynamic programming. Specifically, the algorithm works by filling two tables, namely R and Q, each with an entry for every coalition-type. Entry $R[\langle n^1,\dots,n^T\rangle]$ of table R stores an optimal type-partition of $\langle n^1,\dots,n^T\rangle$, whereas entry $Q[\langle n^1,\dots,n^T\rangle]$ of table Q stores the value of this type-partition. The algorithm fills out these tables using (8.13), where "lower" entries are filled in first, i.e., if $m^i \leq n^i$ for all $i = 1,\dots,T$, then $\langle m^1,\dots,m^T\rangle$ is dealt with before $\langle n^1,\dots,n^T\rangle$. For each $\langle n^1,\dots,n^T\rangle$, the algorithm finds a coalition type $\langle x^1,\dots,x^T\rangle$ that maximizes the max-expression of (8.13), and then sets

$$\begin{aligned} Q[\langle n^1,\dots,n^T\rangle] &= Q[\langle n^1-x^1,\dots,n^T-x^T\rangle]+v^t(\langle x^1,\dots,x^T\rangle),\\ R[\langle n^1,\dots,n^T\rangle] &= R[\langle n^1-x^1,\dots,n^T-x^T\rangle],\langle x^1,\dots,x^T\rangle. \end{aligned}$$

By the end of this process, we compute $Q[\langle|A^1|,\dots,|A^T|\rangle]$ and $R[\langle|A^1|,\dots,|A^T|\rangle]$, which provide the solution to the coalition structure generation problem. Filling out each cell of R and Q requires $O(n^T)$ operations, and the size of each table is $|\Psi| < n^T$. Hence, the algorithm runs in time $O(n^{2T})$.

5.6 Constrained Coalition Formation

So far, we assumed that agents can split into teams in any way they like. However, in practice some coalition structures may be inadmissible. To deal with this issue, Rahwan et al. [54] proposed the *constrained coalition formation (CCF)* framework, which allows one to impose constraints on the coalition structures that can be formed. Formally, a CCF game is a tuple $\langle A,\mathcal{CS},v\rangle$, where A is the set of agents, $\mathcal{CS}$ is the set of coalition structures that are *feasible* (i.e., allowed to form), and v is the characteristic function that assigns a real value to every coalition that appears in some feasible coalition structure. Note that, in the general case, the notion of feasibility is defined for coalition structures rather than coalitions. For instance, if $A = \{a_1,a_2,a_3,a_4\}$ and we define $\mathcal{CS}$ as the set of all coalition structures in which all coalitions have the same size, then the coalition structure $\{\{a_1\},\{a_2\},\{a_3,a_4\}\}$ is not feasible, even though each of its component coalitions may be a part of a feasible coalition structure.

There are, however, many settings of interest where the constraints implied by $\mathcal{CS}$ can be reduced to constraints on individual coalitions. More formally, a CCF game $G = \langle A,\mathcal{CS},v\rangle$ is *locally constrained* if there exists a set of coalitions $\mathcal{C} \subseteq 2^A$ such that $\mathcal{CS} = \{CS \in \mathcal{P}^A \mid CS \subseteq \mathcal{C}\}$. We will refer to the coalitions in $\mathcal{C}$ as *feasible coalitions*.

To represent the constraints succinctly, the authors propose the use of propositional logic. More formally, let $B_A = \{b_i \mid a_i \in A\}$ be a set of Boolean variables, and let φ be a propositional formula over B_A, constructed using the usual classical connectives ($\wedge, \vee, \neg, \rightarrow, \ldots$). A coalition C *satisfies* φ if φ is satisfied under the truth assignment that sets all b_i with $a_i \in C$ to **true** and all b_i with $a_i \notin C$ to **false**. For example, any coalition containing a_1 and a_2 satisfies $\varphi = b_1 \wedge b_2$. It has been shown that this language can represent any locally constrained CCF game, and that it can be extended so as to represent any CCF game [54].

Rahwan et al. then define a natural subclass of locally constrained CCF games, which they call *basic* CCF games. Intuitively, the constraints in a basic CCF game are expressed in the form of (1) *sizes* of coalitions that are allowed to form, and (2) *subsets* of agents whose presence in any coalition is viewed as desirable/prohibited. The constraints of the former type are called *size constraints*, denoted as $\mathcal{S} \subseteq \{1, \ldots, n\}$. As for the latter type of constraints, the desirable subsets of agents are called *positive constraints*, denoted as $\mathcal{P} \subseteq 2^A$, while the prohibited subsets are called *negative constraints*, denoted as $\mathcal{N} \subseteq 2^A$. Thus, a coalition C is feasible if (1) its size is permitted, i.e., $|C| \in \mathcal{S}$, and (2) it contains at least one of the desirable subsets and none of the prohibited ones, i.e., $\exists P \in \mathcal{P} : P \subseteq C$ and $\forall N \in \mathcal{N}, N \nsubseteq C$. We will denote the set of all such feasible coalitions as $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$.

The set of constraints in a basic CCF game can be transformed into another, isomorphic set so as to facilitate both the process of identifying feasible coalitions and the process of searching for an optimal feasible coalition structure [54]. This transformation is based on the observation that, for any agent $a_i \in A$, the coalitions in $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ can be divided into:

- coalitions that contain a_i. For those, any constraint $P \in \mathcal{P} : a_i \in P$ has the same effect as $P \backslash \{a_i\}$. Similarly, any constraint $N \in \mathcal{N} : a_i \in N$ has the same effect as $N \backslash \{a_i\}$. Thus, every such P or N can be replaced with $P \backslash \{a_i\}$ or $N \backslash \{a_i\}$, respectively;

- coalitions that do not contain a_i. For those, every positive or negative constraint that contains a_i has no effect, and so can be removed.

Thus, the problem of dealing with $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ is replaced with two simpler problems; we can then apply the same procedure recursively. This process can be visualized as a tree, where the root is $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$, and each node has two outgoing edges: one leads to a subtree containing some agent a_j and the other leads to a subtree that does not contain a_j. As we move down the tree, the problems become simpler and simpler, until one of the following two cases is reached: (1) a case where one can easily generate the feasible coalitions, which is called a *base case*, or (2) a case where one can easily verify that there are no feasible coalitions (i.e., the constraints cannot be satisfied), which we call an *impossible case* (see [54] for more details). This is illustrated in Figure 8.4 (A), where the edge labels a_i and $\overline{a_i}$ indicate whether the branch contains, or does not contain, a_i, respectively. By

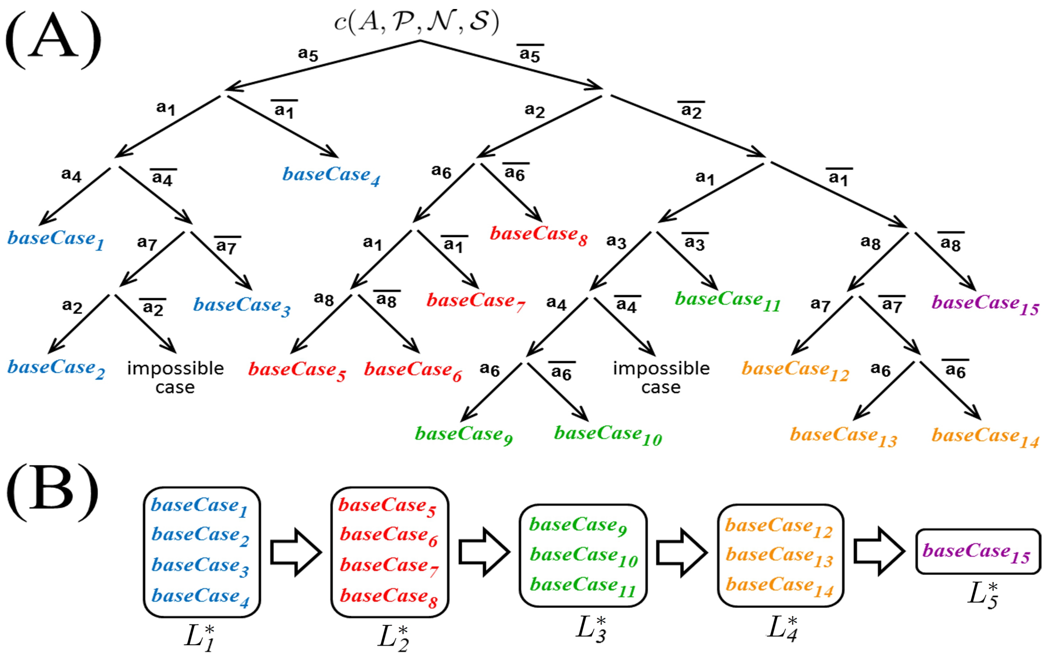


Figure 8.4: Feasible coalitions and coalition structures: given a basic CCF, (A) shows how to generate feasible coalitions, while (B) shows how to generate feasible coalition structures.

generating the feasible coalitions in all base cases, one ends up with the feasible coalitions in $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$.

The tree structure described above also facilitates the search for an optimal feasible coalition structure. Indeed, observe that every such tree contains exactly one path that (1) starts with the root, (2) ends with a leaf, and (3) consists of edges that are each labeled with $\overline{a_i}$ for some $a_i \in A$. In Figure 8.4, for example, this path is the one connecting $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ to *baseCase*$_{15}$. Now, let us denote by A^* the sequence of agents that appear in the labels of this path. For instance, in Figure 8.4, we have $A^* = \langle a_5, a_2, a_1, a_8 \rangle$. Finally, let us denote by a_i^* the i^{th} agent in A^*.

With these definitions in place, we can now present the coalition structure generation algorithm in [54]; we will call this algorithm DC as it uses a *divide-and-conquer* technique. The basic idea is to create lists, $L_1^*, \cdots, L_{|A^*|+1}^*$, where L_1^* consists of the base cases that contain a_1^*, each L_i^*, $i = 1, \ldots, |A^*|$, consists of the base cases that contain a_i^* but not $a_1^*, \ldots, a_{i-1}^*$, and $L_{|A^*|+1}^*$ consists of the base cases that do not contain $a_1^*, \ldots, a_{|A^*|}^*$. This is illustrated in Figure 8.4 (B). Importantly, by constructing the lists in this way, we ensure that every feasible coalition structure contains *exactly* one coalition from L_1^*, and *at most* one coalition from each L_i^*, $i > 1$. Thus, the algorithm picks a coalition, say C_1, from some base case in L_1^*, and checks whether $\{C_1\}$ is a feasible coalition structure. If not, then the agents in C_1 are added to the negative constraints of all base cases in L_2^*. This places further constraints on the coalitions in those base cases, so as to ensure that they do not overlap with C_1. Next, the algorithm picks a coalition, say C_2, from some base case in L_2^*, and checks whether $\{C_1, C_2\}$ is a feasible coalition structure, and so on. Eventually, all feasible coalition structures are examined. To speed up the search, the algorithm applies a branch-and-bound technique (see [54] for more details). This algorithm was compared to the integer programming formulation in Section 5.3.3, where z contains a column for every *feasible* coalition, instead of a column for every *possible* coalition. This comparison showed that DC outperforms the integer programming approach by orders of magnitude.

6 Conclusions

We gave a brief overview of basic notions of cooperative game theory, followed by a discussion of a number of representation formalisms for coalitional games that have been proposed in the literature. We then presented several algorithms for finding an optimal coalition structure, both under the standard representation, and under the more succinct encodings discussed earlier in the chapter. There are several other approaches to the optimal coalition structure generation problem, which we were unable to cover due to space constraints; this problem continues to attract a lot of attention from the multiagent research community due to its challenging nature and numerous applications.

We would like to conclude this chapter by giving a few pointers to the literature. Most standard game theory textbooks provide some coverage of cooperative game theory; the well-known text of Osborne and Rubinstein [47] is a good example. There are also several books that focus exclusively on cooperative games [11, 15, 48]. A very recent book by Chalkiadakis et al. [13] treats the topics covered in the first part of this chapter in considerably more detail than we do, and also discusses coalition formation under uncertainty. However, its coverage of the coalition structure generation problem is much less comprehensive than ours.

7 Exercises

1. $\boxed{\text{Level 1}}$ Compute the Shapley values of all players in the two variants of the ice cream game described in Example 8.2. Do these games have non-empty cores?

2. $\boxed{\text{Level 1}}$ Argue that any n-player induced subgraph game can be represented as a basic MC-net with $O(n^2)$ rules.

3. $\boxed{\text{Level 1}}$ Given the characteristic function shown in Table 8.1, where the value of the grand coalition is 165, identify the optimal coalition structure using the same steps as those of the integer partition-based (IP) algorithm.

4. $\boxed{\text{Level 1}}$ Write the pseudo-code of the dynamic programming (DP) algorithm for coalition structure generation.

5. $\boxed{\text{Level 2}}$ Prove Propositions 8.2–8.5.

6. $\boxed{\text{Level 2}}$ Construct a non-monotone game in which some player's Shapley value is 0, even though this player is not a dummy.

C:$\lvert C\rvert$=1	$v(C)$	C:$\lvert C\rvert$=2	$v(C)$	C:$\lvert C\rvert$=3	$v(C)$	C:$\lvert C\rvert$=4	$v(C)$
$\{a_1\}$	20	$\{a_1,a_2\}$	40	$\{a_1,a_2,a_3\}$	70	$\{a_1,a_2,a_3,a_4\}$	110
$\{a_2\}$	10	$\{a_1,a_3\}$	30	$\{a_1,a_2,a_4\}$	70	$\{a_1,a_2,a_3,a_5\}$	140
$\{a_3\}$	30	$\{a_1,a_4\}$	30	$\{a_1,a_2,a_5\}$	60	$\{a_1,a_2,a_4,a_5\}$	100
$\{a_4\}$	30	$\{a_1,a_5\}$	40	$\{a_1,a_3,a_4\}$	60	$\{a_1,a_3,a_4,a_5\}$	150
$\{a_5\}$	10	$\{a_2,a_3\}$	40	$\{a_1,a_3,a_5\}$	40	$\{a_2,a_3,a_4,a_5\}$	100
		$\{a_2,a_4\}$	20	$\{a_1,a_4,a_5\}$	80		
		$\{a_2,a_5\}$	30	$\{a_2,a_3,a_4\}$	70		
		$\{a_3,a_4\}$	20	$\{a_2,a_3,a_5\}$	50		
		$\{a_3,a_5\}$	65	$\{a_2,a_4,a_5\}$	75		
		$\{a_4,a_5\}$	35	$\{a_3,a_4,a_5\}$	75		

Table 8.1: Sample characteristic function given five agents.

7. $\boxed{\text{Level 2}}$ Consider two simple games $G^1 = (A, v^1)$ and $G^2 = (A, v^2)$ with the same set of players A. Suppose that a player $i \in A$ is not a dummy in both games. Can we conclude that i is not a dummy in the game $G^\cap = (A, v^\cap)$, with the characteristic function $v^\cap$ given by $v^\cap(C) = \min\{v^1(C), v^2(C)\}$? What about the game $G^\cup = (A, v^\cup)$, where $v^\cup$ is given by $v^\cup(C) = \max\{v^1(C), v^2(C)\}$?

8. $\boxed{\text{Level 2}}$ Prove that any outcome in the core maximizes the social welfare, i.e., for any coalitional game G it holds that if $(CS, \mathbf{x})$ is in the core of G, then $CS \in \text{argmax}_{CS \in \mathcal{P}^A} V(CS)$.

9. $\boxed{\text{Level 2}}$ Argue that the problem of finding an optimal coalition structure in a weighted voting game is NP-hard.

10. $\boxed{\text{Level 2}}$ Prove that the running time of the dynamic programming algorithm described in Section 5.2 is $O(3^n)$.

11. $\boxed{\text{Level 2}}$ Provide a formal proof of Theorem 8.6.

12. $\boxed{\text{Level 3}}$ For every pair $(\mathcal{L}_1, \mathcal{L}_2)$ of complete representation languages considered in Section 4.3, find a family of games that can be compactly represented in $\mathcal{L}_1$, but not in $\mathcal{L}_2$, or prove that any game that admits a succinct encoding in $\mathcal{L}_1$ also admits a succinct encoding in $\mathcal{L}_2$.

13. $\boxed{\text{Level 3}}$ Write an implementation of the different metaheuristic algorithms outlined in Section 5.4, and run experiments to compare those algorithms and identify their relative strengths and weaknesses. Are there other metaheuristic algorithms that can be used for coalition structure generation?

14. $\boxed{\text{Level 3}}$ All the algorithms in Section 5 were developed for settings where (1) overlapping coalitions are prohibited, and (2) every coalition's value is not influenced by the coalition structure to which it belongs (unlike in a partition function game, where a coalition's value in one coalition can be different than that in another). Extend one of those algorithms so as to deal with settings where the aforementioned assumptions do not hold.

15. $\boxed{\text{Level 4}}$ Elkind et al. [24] show that the problem of computing the least core of a weighted voting game admits a pseudopolynomial algorithm as well as an FPTAS. The pseudopolynomial algorithm extends to the nucleolus [26]; however, it is not known if the problem of computing the nucleolus admits an FPTAS. Develop an FPTAS for this problem, or prove that this is not possible (under a suitable complexity assumption).

References

[1] George E. Andrews and Kimmo Eriksson. *Integer Partitions*. Cambridge University Press, Cambridge, UK, 2004.

[2] Haris Aziz, Felix Brandt, and Paul Harrenstein. Monotone cooperative games and their threshold versions. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1107–1114, 2010.

[3] Haris Aziz and Bart de Keijzer. Complexity of coalition structure generation. In *AAMAS'11: Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 191–198, 2011.

[4] Yoram Bachrach, Reshef Meir, Kyomin Jung, and Pushmeet Kohli. Coalitional structure generation in skill games. In *AAAI'10: Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 703–708, 2010.

[5] Yoram Bachrach and Jeffrey S. Rosenschein. Coalitional skill games. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1023–1030, 2008.

[6] Yoram Bachrach and Jeffrey S. Rosenschein. Power in threshold network flow games. *Autonomous Agents and Multi-Agent Systems*, 18(1):106–132, 2009.

[7] John F. Banzhaf. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19:317–343, 1965.

[8] Eric T. Bell. Exponential numbers. *American Mathematical Monthly*, 41:411–419, 1934.

[9] J. M. Bilbao, J. R. Fernández, N. Jiminéz, and J. J. López. Voting power in the European Union enlargement. *European Journal of Operational Research*, 143:181–196, 2002.

[10] Peter Borm, Herbert Hamers, and Ruud Hendrickx. Operations research games: A survey. *TOP*, 9:139–199, 2001.

[11] Rodica Brânzei, Dinko Dimitrov, and Stef Tijs. *Models in Cooperative Game Theory*. Springer, 2005.

[12] Georgios Chalkiadakis, Edith Elkind, Evangelos Markakis, Maria Polukarov, and Nicholas R. Jennings. Cooperative games with overlapping coalitions. *Journal of Artificial Intelligence Research (JAIR)*, 39:179–216, 2010.

[13] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan and Claypool, 2011.

[14] Vincent Conitzer and Tuomas Sandhom. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence*, 170(6–7):607–619, 2006.

[15] Imma Curiel. *Cooperative Game Theory and Applications*. Kluwer, 1997.

[16] Viet D. Dang and Nicholas R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS'04: Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 564–571, 2004.

[17] Morton Davis and Michael Maschler. The kernel of a cooperative game. *Naval Research Logistics Quarterly*, 12(3):223–259, 1965.

[18] Nicolaas G. de Bruijn. *Asymptotic Methods in Analysis*. Dover, 1981.

[19] Xiaotie Deng, Qizhi Fang, and Xiaoxun Sun. Finding nucleolus of flow game. In *SODA'06: 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 124–131, 2006.

[20] Xiaotie Deng, Toshihide Ibaraki, and Hiroshi Nagamochi. Algorithmic aspects of the core of combinatorial optimization games. *Mathematics of Operations Research*, 24(3):751–766, 1999.

[21] Xiaotie Deng and Christos Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[22] Ezra Einy, Ron Holzman, and Dov Monderer. On the least core and the Mas-Colell bargaining set. *Games and Economic Behavior*, 28:181–188, 1999.

[23] Edith Elkind, Georgios Chalkiadakis, and Nicholas R. Jennings. Coalition structures in weighted voting games. In *ECAI'08: Eighteenth European Conference on Artificial Intelligence*, pages 393–397, 2008.

[24] Edith Elkind, Leslie Ann Goldberg, Paul Goldberg, and Michael Wooldridge. On the computational complexity of weighted voting games. *Annals of Mathematics and Artificial Intelligence*, 56(2):109–131, 2009.

[25] Edith Elkind, Leslie Ann Goldberg, Paul Goldberg, and Michael Wooldridge. A tractable and expressive class of marginal contribution nets and its applications. *Mathematical Logic Quarterly*, 55(4):362–376, 2009.

[26] Edith Elkind and Dmitrii Pasechnik. Computing the nucleolus of weighted voting games. In *SODA'09: 20th ACM-SIAM Symposium on Discrete Algorithms*, 2009.

[27] Piotr Faliszewski, Edith Elkind, and Michael Wooldridge. Boolean combinations of weighted voting games. In *AAMAS'09: 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 185–192, 2009.

[28] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[29] Donald B. Gillies. Solutions to general non-zero-sum games. In H. W. Kuhn, A. W. Tucker, and L. D. Luce, editors, *Contributions to the Theory of Games, volume IV*, pages 47–85. Princeton University Press, 1959.

[30] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In *MFCS'01: 26th International Symposium on Mathematical Foundations of Computer Science*, pages 37–57, 2001.

[31] Daniel Granot and Frieda Granot. On some network flow games. *Mathematics of Operations Research*, 17(4):792–841, 1992.

[32] Samuel Ieong and Yoav Shoham. Marginal contribution nets: a compact representation scheme for coalitional games. In *ACM EC'05: 6th ACM Conference on Electronic Commerce*, pages 193–202, 2005.

[33] Ehud Kalai and Eitan Zemel. Generalized network problems yielding totally balanced games. *Operations Research*, 30(5):998–1008, 1982.

[34] Ehud Kalai and Eitan Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, 7(3):476–478, 1982.

[35] Helena Keinänen. Simulated annealing for multi-agent coalition formation. In *KES-AMSTA'09: Third KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pages 30–39, 2009.

[36] Walter Kern and Daniël Paulusma. Matching games: the least core and the nucleolus. *Mathematics of Operations Research*, 28(2):294–308, 2003.

[37] William Lucas and Robert Thrall. n-person games in partition function form. *Naval Research Logistic Quarterly*, pages 281–298, 1963.

[38] Andreu Mas-Colell. An equivalence theorem for a bargaining set. *Journal of Mathematical Economics*, 18:129–139, 1989.

[39] Michael Maschler, Bezalel Peleg, and Lloyd S. Shapley. Geometric properties of the kernel, nucleolus, and related solution concepts. *Mathematics of Operations Research*, 4:303–338, 1979.

[40] Tomomi Matsui and Yasuko Matsui. A survey of algorithms for calculating power indices of weighted majority games. *Journal of the Operations Research Society of Japan*, 43(1):71–86, 2000.

[41] Yasuko Matsui and Tomomi Matsui. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1-2):305–310, 2001.

[42] Nicola Di Mauro, Teresa M. A. Basile, Stefano Ferilli, and Floriana Esposito. Coalition structure generation with GRASP. In *AIMSA'10: Fourteenth International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 111–120, 2010.

[43] Tomasz Michalak, Jacek Sroka, Talal Rahwan, Michael Wooldridge, Peter McBurney, and Nicholas R. Jennings. A distributed algorithm for anytime coalition structure generation. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1007–1014, 2010.

[44] Pragnesh Jay Modi. *Distributed Constraint Optimization for Multiagent Systems*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 2003.

[45] Naoki Ohta, Vincent Conitzer, Ryo Ichimura, Yuko Sakurai, Atsushi Iwasaki, and Makoto Yokoo. Coalition structure generation utilizing compact characteristic function representations. In *CP'09: Fifteenth International Conference on Principles and Practice of Constraint Programming*, pages 623–638, 2009.

[46] Naoki Ohta, Atsushi Iwasaki, Makoto Yokoo, Kohki Maruono, Vincent Conitzer, and Tuomas Sandholm. A compact representation scheme for coalitional games in open anonymous environments. In *AAAI'06: Twenty-First National Conference on Artificial Intelligence*, pages 697–702, 2006.

[47] Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[48] David Peleg and Peter Sudhölter. *Introduction to the Theory of Cooperative Games*. Springer, 2007.

[49] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI'05: Nineteenth International Joint Conference on Artificial Intelligence*, pages 266–271, 2005.

[50] Kislaya Prasad and Jerry S. Kelly. NP-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, 1990.

[51] Talal Rahwan and Nicholas R. Jennings. An algorithm for distributing coalitional value calculations among cooperative agents. *Artificial Intelligence*, 171(8–9):535–567, 2007.

[52] Talal Rahwan and Nicholas R. Jennings. Coalition structure generation: Dynamic programming meets anytime optimisation. In *AAAI'08: Twenty-Third AAAI Conference on Artificial Intelligence*, pages 156–161, 2008.

[53] Talal Rahwan and Nicholas R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1417–1420, 2008.

[54] Talal Rahwan, Tomasz P. Michalak, Edith Elkind, Piotr Faliszewski, Jacek Sroka, Michael Wooldridge, and Nicholas R. Jennings. Constrained coalition formation. In *AAAI'11: Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 719–725, 2011.

[55] Talal Rahwan, Tomasz P. Michalak, and Nicholas R. Jennings. Minimum search to establish worst-case guarantees in coalition structure generation. In *IJCAI'11: Twenty-Second International Joint Conference on Artificial Intelligence*, pages 338–343, 2011.

[56] Talal Rahwan, Sarvapali D. Ramchurn, Viet D. Dang, and Nicholas R. Jennings. Near-optimal anytime coalition structure generation. In *IJCAI'07: Twentieth International Joint Conference on Artificial Intelligence*, pages 2365–2371, 2007.

[57] Talal Rahwan, Sarvapali D. Ramchurn, Andrea Giovannucci, Viet D. Dang, and Nicholas R. Jennings. Anytime optimal coalition structure generation. In *AAAI'07: Twenty-Second Conference on Artificial Intelligence*, pages 1184–1190, 2007.

[58] Talal Rahwan, Sarvapali D. Ramchurn, Andrea Giovannucci, and Nicholas R. Jennings. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research (JAIR)*, 34:521–567, 2009.

[59] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 2003.

[60] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst-case guarantees. *Artificial Intelligence*, 111(1–2):209–238, 1999.

[61] David Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 17:1163–1170, 1969.

[62] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[63] Sandip Sen and Partha Dutta. Searching for optimal coalition structures. In *ICMAS'00: Sixth International Conference on Multi-Agent Systems*, pages 286–292, 2000.

[64] Lloyd S. Shapley. A value for *n*-person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games, volume II*, pages 307–317. Princeton University Press, 1953.

[65] Lloyd S. Shapley and Martin Shubik. The assignment game I: The core. *International Journal of Game Theory*, 1:111–130, 1972.

[66] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1–2):165–200, 1998.

[67] Tammar Shrot, Yonatan Aumann, and Sarit Kraus. On agent types in coalition formation problems. In *AAMAS'10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 757–764, 2010.

[68] Tamas Solymosi and T. E. S. Raghavan. An algorithm for finding the nucleolus of assignment games. *International Journal of Game Theory*, 23:119–143, 1994.

[69] Alan D. Taylor and William S. Zwicker. *Simple Games*. Princeton University Press, 1999.

[70] Suguru Ueda, Atsushi Iwasaki, Makoto Yokoo, Marius Calin Silaghi, Katsutoshi Hirayama, and Toshihiro Matsui. Coalition structure generation based on distributed constraint optimization. In *AAAI'10: Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 197–203, 2010.

[71] Suguru Ueda, Makoto Kitaki, Atsushi Iwasaki, and Makoto Yokoo. Concise characteristic function representations in coalitional games based on agent types. In *IJCAI'11: Twenty-Second International Joint Conference on Artificial Intelligence*, pages 393–399, 2011.

[72] D. Yun Yeh. A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, 26(4):467–474, 1986.

[73] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.