

Chapter 8: Computational Coalition Formation

Edith Elkind

(Nanyang Technological University, Singapore)

Talal Rahwan, Nicholas R. Jennings

(University of Southampton, UK)

Lecture Overview

- **Part 1:** Coalitional Game Theory
 - how do **selfish** agents form teams in order to work together
- **Part 2:** Coalition Structure Generation
 - how can a **benevolent** center split agents into teams in the most **efficient** manner

Part 1 (Coalitional Game Theory): Overview

- Introduction
- Definitions
- Solution concepts
- Representations
and computational issues

A Point of Reference: Non-Cooperative Games

- You are probably all familiar with non-cooperative games...
- A non-cooperative game is defined by
 - a set of agents (players) $N = \{1, \dots, n\}$
 - for each agent $i \in N$, a set of actions S_i
 - for each agent $i \in N$, a utility function u_i ,
$$u_i: S_1 \times \dots \times S_n \rightarrow R$$
- Observe that an agent's utility depends not just on her action, but on actions of other agents
 - thus, for i finding the best action involves deliberating about what other will do

Example: Prisoner's Dilemma



- Two agents committed a crime.
- Court does not have enough evidence to convict them of the crime, but can convict them of a minor offence (**1** year in prison each)
- If one suspect confesses (acts as an informer), he walks free, and the other suspect gets **4** years
- If both confess, each gets **3** years
- Agents have no way of **communicating** or making **binding agreements**

Prisoner's Dilemma: Matrix Representation

		P2	
		quiet	confess
P1	quiet	$(-1, -1)$	$(-4, 0)$
	confess	$(0, -4)$	$(-3, -3)$

- Interpretation: the pair (x, y) at the intersection of row i and column j means that the row player gets x and the column player gets y

Prisoners' Dilemma: the Rational Outcome

- **P1**'s reasoning:
 - if **P2** stays quiet, I should confess
 - if **P2** confesses, I should confess, too

	P2	Q	C
P1			
Q	$(-1, -1)$	$(-4, 0)$	
C	$(0, -4)$	$(-3, -3)$	

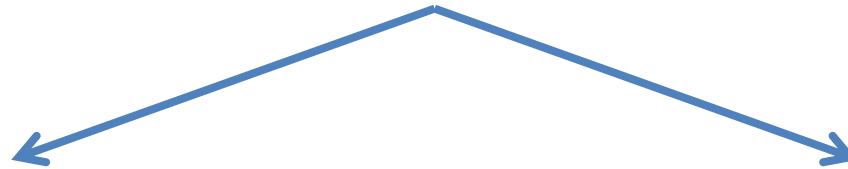
- **P2** reasons in the same way
- Result: both confess and get 3 years in prison.
- However, if they chose to cooperate and stay quiet, they could get away with 1 year each.
- So why do not they cooperate?

Assumptions in Non-Cooperative Games

- Cooperation does not occur in prisoners' dilemma, because players cannot make binding agreements
- But what if binding agreements are possible?
- This is exactly the class of scenarios studied by cooperative game theory

Cooperative Games

- Cooperative games model scenarios, where
 - agents can benefit by cooperating
 - binding agreements are possible
- In cooperative games, actions are taken by groups of agents

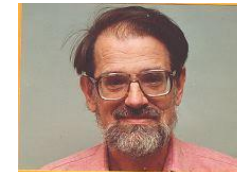
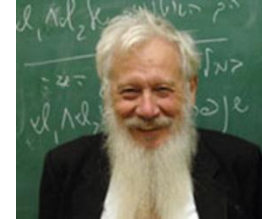


Transferable utility games:
payoffs are given to the group and then divided among its members

Non-transferable utility games: group actions result in payoffs to individual group members

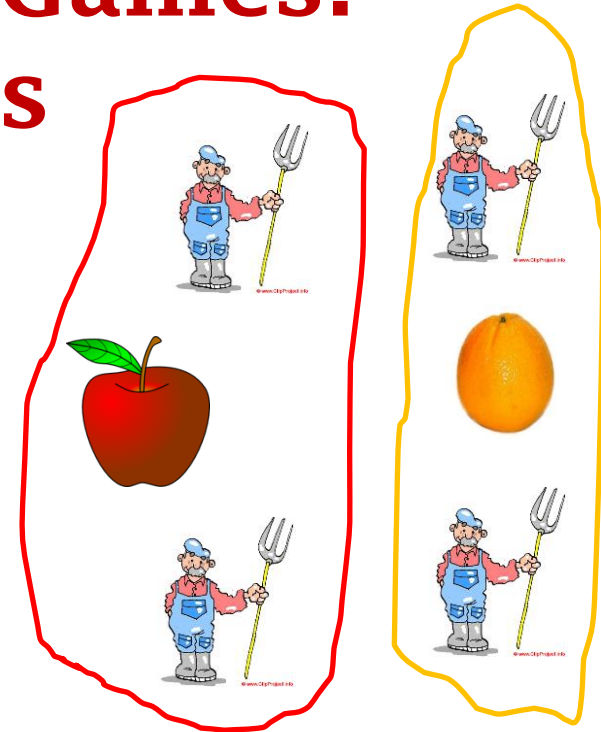
Non-Transferable Utility Games: Writing Papers

- n researchers working at n different universities can form groups to write papers on game theory
- each group of researchers can work together; the composition of a group determines the quality of the paper they produce
- each author receives a payoff from his own university
 - promotion
 - bonus
 - teaching load reduction
- Payoffs are non-transferable



Transferable Utility Games: Happy Farmers

- n farmers can cooperate to grow fruit
- Each group of farmers can grow apples or oranges
- a group of size k can grow $f(k)$ tons of apples and $g(k)$ tons of oranges
 - $f()$, $g()$ are convex functions of k
- Fruit can be sold in the market:
 - if there are x tons of apples and y tons of oranges on the market, the market prices for apples and oranges are $\max\{X - x, 0\}$ and $\max\{Y - y, 0\}$, respectively
 - X, Y are some large enough constants
- The profit of each group depends on the quantity and type of fruit it grows, and the market price



Transferable Utility Games: Buying Ice-cream

- n children, each has some amount of money
 - the i -th child has b_i dollars
- three types of ice-cream tubs are for sale:
 - Type 1 costs \$7, contains 500g
 - Type 2 costs \$9, contains 750g
 - Type 3 costs \$11, contains 1kg
- children have utility for ice-cream, and do not care about money
- The payoff of each group: the maximum quantity of ice-cream the members of the group can buy by pooling their money
- The ice-cream can be shared arbitrarily within the group

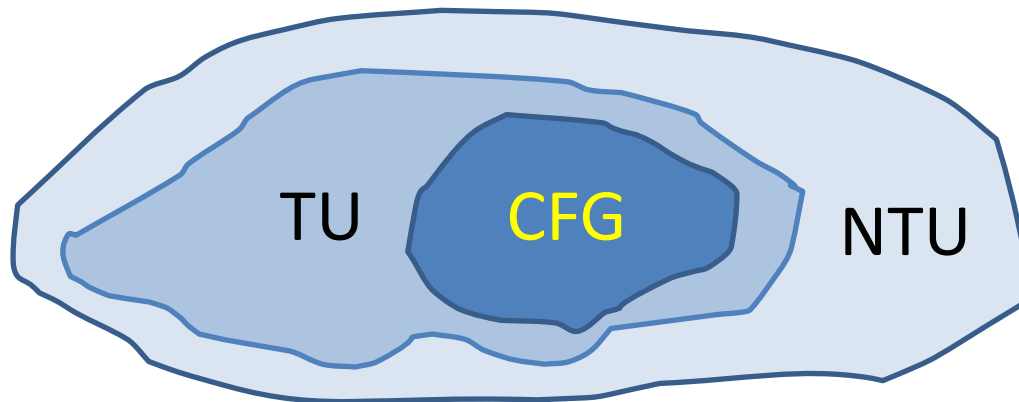


Characteristic Function Games vs. Partition Function Games

- In general TU games, the payoff obtained by a coalition depends on the actions chosen **by other coalitions**
 - these games are also known as **partition function games** (PFG)
- **Characteristic function games** (CFG): the payoff of each coalition only depends on the **action of that coalition**
 - in such games, each coalition can be identified with the profit it obtains by choosing its best action
 - Ice Cream game is a CFG
 - Happy Farmers game is a PFG, but not a CFG

Classes of Cooperative Games: The Big Picture

- Any TU game can be represented as an NTU game with a continuum of actions
 - each payoff division scheme in the TU game can be interpreted as an action in the NTU game



- We will focus on characteristic function games, and use term “TU games” to refer to such games

How Is a Cooperative Game Played?

- Even though agents work together they are still **selfish**
- The partition into coalitions and payoff distribution should be such that no player (or group of players) has an **incentive to deviate**
- We may also want to ensure that the outcome is **fair**: the payoff of each agent is proportional to his **contribution**
- We will now see how to formalize these ideas

Part 1: Overview

- Introduction
- **Definitions**
- Solution concepts
- Representations and computational issues

Transferable Utility Games Formalized

- A transferable utility game is a pair (N, v) , where:
 - $N = \{1, \dots, n\}$ is the set of players
 - $v: 2^N \rightarrow \mathbb{R}$ is the characteristic function
 - for each subset of players C , $v(C)$ is the amount that the members of C can earn by working together
 - usually it is assumed that v is
 - normalized: $v(\emptyset) = 0$
 - non-negative: $v(C) \geq 0$ for any $C \subseteq N$
 - monotone: $v(C) \leq v(D)$ for any C, D such that $C \subseteq D$
- A coalition is any subset of N ;
 N itself is called the grand coalition

Ice-Cream Game: Characteristic Function



C: \$6,



M: \$4,



P: \$3



w = 500

p = \$7



w = 750

p = \$9



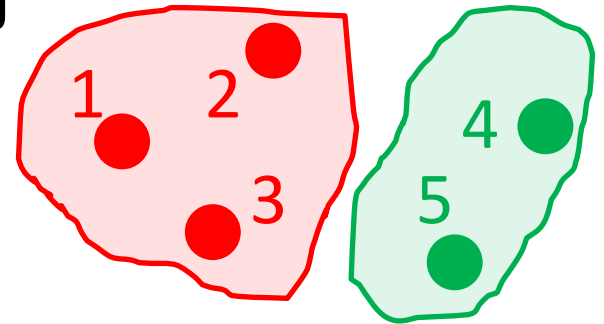
w = 1000

p = \$11

- $v(\emptyset) = v(\{C\}) = v(\{M\}) = v(\{P\}) = 0$
- $v(\{C, M\}) = 750$, $v(\{C, P\}) = 750$, $v(\{M, P\}) = 500$
- $v(\{C, M, P\}) = 1000$

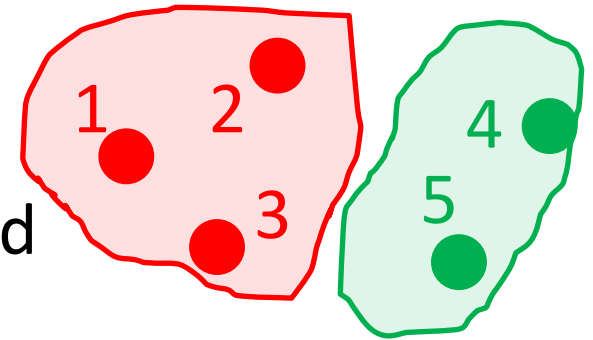
Transferable Utility Games: Outcome

- An **outcome** of a TU game $G = (N, v)$ is a pair (CS, \underline{x}) , where:
 - $CS = (C_1, \dots, C_k)$ is a **coalition structure**, i.e., **partition** of N into coalitions:
 - $\cup_i C_i = N$, $C_i \cap C_j = \emptyset$ for $i \neq j$
 - $\underline{x} = (x_1, \dots, x_n)$ is a **payoff vector**, which distributes the value of each coalition in CS :
 - $x_i \geq 0$ for all $i \in N$
 - $\sum_{i \in C} x_i = v(C)$ for each C in CS



Transferable Utility Games: Outcome

- Example:
 - suppose $v(\{1, 2, 3\}) = 9$, $v(\{4, 5\}) = 4$
 - then $((\{1, 2, 3\}, \{4, 5\}), (3, 3, 3, 3, 1))$ is an outcome
 - $((\{1, 2, 3\}, \{4, 5\}), (2, 3, 2, 3, 3))$ is NOT an outcome: transfers between coalitions are not allowed



- An outcome (CS, \underline{x}) is called an **imputation** if it satisfies **individual rationality**:
 $x_i \geq v(\{i\})$ for all $i \in N$
- Notation: we will denote $\sum_{i \in C} x_i$ by $x(C)$

Superadditive Games

- Definition: a game $G = (N, v)$ is called superadditive if $v(C \cup D) \geq v(C) + v(D)$ for any two disjoint coalitions C and D
- Example: $v(C) = |C|^2$:
 - $v(C \cup D) = (|C| + |D|)^2 \geq |C|^2 + |D|^2 = v(C) + v(D)$
- In superadditive games, two coalitions can always **merge** without losing money; hence, we can assume that players form the **grand coalition**




Superadditive Games

- Convention: in superadditive games, we identify outcomes with payoff vectors for the grand coalition
 - i.e., an outcome is a vector $\underline{x} = (x_1, \dots, x_n)$ with $\sum_{i \in N} x_i = v(N)$
- Caution: some GT/MAS papers define outcomes in this way even if the game is not superadditive
- Any non-superadditive game $G = (N, v)$ can be transformed into a superadditive game $G^{SA} = (N, v^{SA})$ by setting $v^{SA}(C) = \max_{(C_1, \dots, C_k) \in P(C)} \sum_{i=1, \dots, k} v(C_i)$, where $P(C)$ is the space of all partitions of C
- G^{SA} is called the superadditive cover of G

Part 1: Overview

- Introduction
- Definitions
- Solution concepts
 - core
 - least core
 - nucleolus
 - bargaining set
 - kernel
 - Shapley value and Banzhaf index
- Representations and computational issues

What Is a Good Outcome?

-  C: \$4,  M: \$3,  P: \$3
- $v(\emptyset) = v(\{C\}) = v(\{M\}) = v(\{P\}) = 0$
- $v(\{C, M\}) = 500$, $v(\{C, P\}) = 500$, $v(\{M, P\}) = 0$
- $v(\{C, M, P\}) = 750$
- This is a superadditive game
 - outcomes are payoff vectors
- How should the players share the ice-cream?
 - if they share as $(200, 200, 350)$, Charlie and Marcie can **get more** ice-cream by buying a **500g** tub on their own, and **splitting** it equally
 - the outcome $(200, 200, 350)$ is not **stable!**

Transferable Utility Games: Stability

- Definition: the **core** of a game is the set of all **stable** outcomes, i.e., outcomes that no coalition wants to deviate from

$$\text{core}(G) = \{(CS, \underline{x}) \mid \sum_{i \in C} x_i \geq v(C) \text{ for any } C \subseteq N\}$$

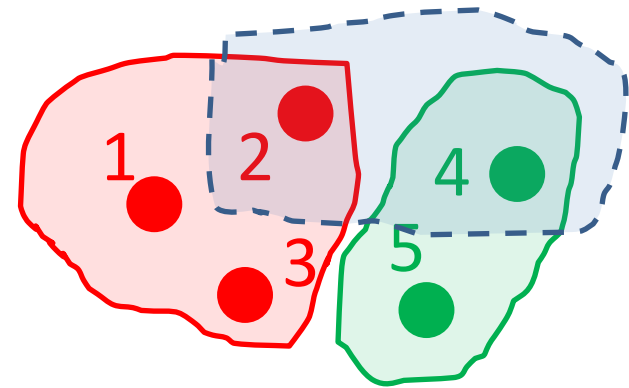
- each coalition earns at least as much as it can make on its own

- Note that G is **not** assumed to be superadditive




- Example

- suppose $v(\{1, 2, 3\}) = 9$,
 $v(\{4, 5\}) = 4$, $v(\{2, 4\}) = 7$

- then $((\{1, 2, 3\}, \{4, 5\}), (3, 3, 3, 3, 1))$ is NOT in the core



Ice-Cream Game: Core

-  C: \$4,  M: \$3,  P: \$3
- $v(\emptyset) = v(\{C\}) = v(\{M\}) = v(\{P\}) = 0$, $v(\{C, M, P\}) = 750$
- $v(\{C, M\}) = 500$, $v(\{C, P\}) = 500$, $v(\{M, P\}) = 0$
- $(200, 200, 350)$ is not in the core:
 - $v(\{C, M\}) > x_C + x_M$
- $(250, 250, 250)$ is in the core:
 - no subgroup of players can deviate so that each member of the subgroup gets more
- $(750, 0, 0)$ is also in the core:
 - Marcie and Pattie cannot get more on their own!

Games with Empty Core

- The core is a very attractive solution concept
- However, some games have empty cores
- $G = (N, v)$
 - $N = \{1, 2, 3\}$, $v(C) = 1$ if $|C| > 1$ and $v(C) = 0$ otherwise
 - consider an outcome (CS, \underline{x})
 - if $CS = (\{1\}, \{2\}, \{3\})$, the grand coalition can deviate
 - if $CS = (\{1, 2\}, \{3\})$, either 1 or 2 gets less than 1, so can deviate with 3
 - same argument for $CS = (\{1, 3\}, \{2\})$ or $CS = (\{2, 3\}, \{1\})$
 - suppose $CS = \{1, 2, 3\}$:
 - $x_i > 0$ for some i , so $x(N \setminus \{i\}) < 1$, yet $v(N \setminus \{i\}) = 1$

Core and Superadditivity

- Suppose the game is **not superadditive**, but the outcomes are defined as payoff vectors for the **grand coalition**
- Then the core may be **empty**, even if according to the **standard** definition it is not
- $G = (N, v)$
 - $N = \{1, 2, 3, 4\}$, $v(C) = 1$ if $|C| > 1$ and $v(C) = 0$ otherwise
 - not superadditive: $v(\{1, 2\}) + v(\{3, 4\}) = 2 > v(\{1, 2, 3, 4\})$
 - no payoff vector for the grand coalition is in the core:
 - either $\{1, 2\}$ or $\{3, 4\}$ get less than **1**, so can deviate
 - $((\{1, 2\}, \{3, 4\}), (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}))$ is in the core

ε -Core

- If the core is empty, we may want to find **approximately stable** outcomes
- Need to **relax** the notion of the core:
core: $(CS, \underline{x}): x(C) \geq v(C)$ for all $C \subseteq N$
 ε -core: $(CS, \underline{x}): x(C) \geq v(C) - \varepsilon$ for all $C \subseteq N$
- Is usually defined for superadditive games only
- Example: $G = (N, v)$, $N = \{1, 2, 3\}$,
 $v(C) = 1$ if $|C| > 1$, $v(C) = 0$ otherwise
 - $1/3$ -core is non-empty: $(1/3, 1/3, 1/3) \in 1/3$ -core
 - ε -core is empty for any $\varepsilon < 1/3$:
 $x_i \geq 1/3$ for some $i = 1, 2, 3$, so $x(N \setminus \{i\}) \leq 2/3$, $v(N \setminus \{i\}) = 1$

Least Core

- If an outcome (CS, \underline{x}) is in ε -core, the deficit $v(C) - x(C)$ of any coalition is at most ε
- We are interested in outcomes that minimize the worst-case deficit
- Let $\varepsilon^*(G) = \inf \{ \varepsilon \mid \varepsilon\text{-core of } G \text{ is not empty} \}$
 - it can be shown that $\varepsilon^*(G)$ -core is not empty
- Definition: $\varepsilon^*(G)$ -core is called the least core of G
 - $\varepsilon^*(G)$ is called the value of the least core
- $G = (N, v)$, $N = \{1, 2, 3\}$,
 $v(C) = 1$ if $|C| > 1$, $v(C) = 0$ otherwise
 - $1/3$ -core is non-empty, but ε -core is empty for any $\varepsilon < 1/3$, so least core = $1/3$ -core

Further Solution Concepts

- We will now define 5 more solution concepts:
 - the nucleolus
 - the bargaining set
 - the kernel
 - the Shapley value
 - the Banzhaf index
- } more complicated stability considerations
- } fairness considerations
- For simplicity, we will define all these solution concepts for superadditive games only
 - however, all definitions generalize to non-superadditive games

Nucleolus

- Least core is always non-empty
- However, it may contain more than one point
- Can we identify the **most stable** point in the least core?
- Given an outcome \underline{x} of a game $G(N, v)$, let $d(C) = v(C) - x(C)$
 - $d(C)$ is called the deficit of C wrt \underline{x}
- The least core minimizes the **max** deficit
- The nucleolus of G is an imputation that
 - minimizes the **max** deficit
 - given this, minimizes the **2nd-largest** deficit, etc.

Nucleolus: Formal Definition and Properties

- Definition: the deficit vector for an outcome \underline{x} is the list of deficits of all 2^n coalitions, ordered from the largest to the smallest
- Definition: the nucleolus is an imputation that corresponds to the lexicographically smallest deficit vector
- If we optimize over all outcomes (and not just imputations), we obtain pre-nucleolus
- Nucleolus is unique
 - the “most stable” outcome
- Appears in Talmud as an estate division scheme

Objections and Counterobjections

- An outcome is not in the core if some coalition objects to it; but is the objection itself **plausible**?
- Fix an imputation \underline{x} for a game $G=(N, v)$
- A pair (\underline{y}, S) , where \underline{y} is an imputation and $S \subseteq N$, is an **objection** of player i against player j to \underline{x} if
 - $i \in S, j \notin S, y(S) = v(S)$
 - $y_k > x_k$ for all $k \in S$
- A pair (\underline{z}, T) , where \underline{z} is an imputation and $T \subseteq N$, is a **counterobjection** to the objection (\underline{y}, S) if
 - $j \in T, i \notin T, z(S) = v(S), T \cap S \neq \emptyset$
 - $z_k \geq x_k$ for all $k \in T \setminus S$
 - $z_k \geq y_k$ for all $k \in T \cap S$

Bargaining Set

- An objection is said to be **justified** if it does not admit a counterobjection
- Definition: the **bargaining set** of a game **G** consist of all imputations that do not admit a **justified objection**
- The core is the set of all imputations that do not admit an **objection**. Hence
$$\text{core} \subseteq \text{bargaining set}$$

Kernel (1/2)

- Let $I(i, k) = \{ C \subseteq N \mid i \in C, k \notin C \}$
- Definition: the surplus $sur(i, k)$ of an agent i over agent k wrt an imputation \underline{x} is
$$sur(i, k) = \max_{C \in I(i, k)} d(C), \text{ where } d(C) = v(C) - x(C)$$
- Definition: an agent i outweighs agent k under \underline{x} if $sur(i, k) > sur(k, i)$
- If i outweighs k under \underline{x} ,
 i should be able to claim some of k 's payoff x_k
- However, the amount he can claim is limited by individual rationality: we should have $x_k \geq v(\{k\})$

Kernel (2/2)

- Definition: two agents i and k are in **equilibrium** with respect to the imputation \underline{x} if one of the following holds:
 - $\text{sur}(i, k) = \text{sur}(k, i)$
 - $\text{sur}(i, k) > \text{sur}(k, i)$ and $x_k = v(\{k\})$
 - $\text{sur}(i, k) < \text{sur}(k, i)$ and $x_i = v(\{i\})$
- Definition: an imputation \underline{x} is in the **kernel** if any two agents i and k are in equilibrium wrt \underline{x}
nucleolus \subseteq kernel \subseteq bargaining set

Stability vs. Fairness

- Outcomes in the core may be **unfair**
- $G = (N, v)$
 - $N = \{1, 2\}$, $v(\emptyset) = 0$, $v(\{1\}) = v(\{2\}) = 5$, $v(\{1, 2\}) = 20$
- $(15, 5)$ is in the core:
 - player **2** cannot benefit by deviating
- However, this is unfair since **1** and **2** are **symmetric**
- How do we divide payoffs in a fair way?

Marginal Contribution

- A fair payment scheme would reward each agent according to his **contribution**
- First attempt: given a game $G = (N, v)$, set $x_i = v(\{1, \dots, i-1, i\}) - v(\{1, \dots, i-1\})$
 - payoff to each player = his **marginal contribution to the coalition of his predecessors**
- We have $x_1 + \dots + x_n = v(N)$
 - \underline{x} is a payoff vector
- However, payoff to each player depends on the order
- $G = (N, v)$
 - $N = \{1, 2\}$, $v(\emptyset) = 0$, $v(\{1\}) = v(\{2\}) = 5$, $v(\{1, 2\}) = 20$
 - $x_1 = v(1) - v(\emptyset) = 5$, $x_2 = v(\{1, 2\}) - v(\{1\}) = 15$

Average Marginal Contribution

- Idea: to remove the dependence on ordering, can **average** over all possible orderings
- $G = (N, v)$
 - $N = \{1, 2\}$, $v(\emptyset) = 0$, $v(\{1\}) = v(\{2\}) = 5$, $v(\{1, 2\}) = 20$
 - **1, 2**: $x_1 = v(1) - v(\emptyset) = 5$, $x_2 = v(\{1, 2\}) - v(\{1\}) = 15$
 - **2, 1**: $y_2 = v(2) - v(\emptyset) = 5$, $y_1 = v(\{1, 2\}) - v(\{2\}) = 15$
 - $z_1 = (x_1 + y_1)/2 = 10$, $z_2 = (x_2 + y_2)/2 = 10$
 - the resulting outcome is fair!
- Can we generalize this idea?

Shapley Value

- Reminder: a **permutation** of $\{1, \dots, n\}$ is a one-to-one mapping from $\{1, \dots, n\}$ to itself
 - let $P(N)$ denote the set of all permutations of N
- Let $S_\pi(i)$ denote the set of predecessors of i in $\pi \in P(N)$



- For $C \subseteq N$, let $\delta_i(C) = v(C \cup \{i\}) - v(C)$
- Definition: the **Shapley value** of player i in a game $G = (N, v)$ with $|N| = n$ is

$$\phi_i(G) = 1/n! \sum_{\pi: \pi \in P(N)} \delta_i(S_\pi(i))$$

- In the previous slide we have $\phi_1 = \phi_2 = 10$

Shapley Value: Probabilistic Interpretation

- ϕ_i is i 's average marginal contribution to the coalition of its predecessors, over all permutations
- Suppose that we choose a permutation of players uniformly at random, among all possible permutations of N
 - then ϕ_i is the expected marginal contribution of player i to the coalition of his predecessors

Shapley Value: Properties (1)-(2)

- Proposition: in any game G ,
$$\phi_1 + \dots + \phi_n = v(N)$$
 - (ϕ_1, \dots, ϕ_n) is a payoff vector
- Definition: a player i is a **dummy** in a game $G = (N, v)$ if $v(C) = v(C \cup \{i\})$ for any $C \subseteq N$
- Proposition: if a player i is a dummy in a game $G = (N, v)$ then $\phi_i = 0$

Shapley Value: Properties (3)-(4)

- Definition: given a game $G = (N, v)$, two players i and j are said to be **symmetric** if $v(C \cup \{i\}) = v(C \cup \{j\})$ for any $C \subseteq N \setminus \{i, j\}$
- Proposition: if i and j are symmetric then $\phi_i = \phi_j$
- Definition: Let $G_1 = (N, u)$ and $G_2 = (N, v)$ be two games with the same set of players. Then $G = G_1 + G_2$ is the game with the set of players N and characteristic function w given by $w(C) = u(C) + v(C)$ for all $C \subseteq N$
- Proposition: $\phi_i(G_1 + G_2) = \phi_i(G_1) + \phi_i(G_2)$

Axiomatic Characterization

- Properties of Shapley value:
 1. Efficiency: $\phi_1 + \dots + \phi_n = v(N)$
 2. Dummy: if i is a dummy, $\phi_i = 0$
 3. Symmetry: if i and j are symmetric, $\phi_i = \phi_j$
 4. Additivity: $\phi_i(G_1 + G_2) = \phi_i(G_1) + \phi_i(G_2)$
- Theorem: Shapley value is the **only** payoff distribution scheme that has properties (1) - (4)

Banzhaf Index

- Instead of averaging over all permutations of players, we can average over all coalitions
- Definition: the Banzhaf index of player i in a game $G = (N, v)$ with $|N| = n$ is
$$\beta_i(G) = 1/2^{n-1} \sum_{C \subseteq N \setminus \{i\}} \delta_i(C)$$
- Satisfies dummy axiom, symmetry and additivity
- However, may fail efficiency:
it may happen that $\sum_{i \in N} \beta_i \neq v(N)$

Shapley and Banzhaf: Examples

- Example 1 (unanimity game):
 - $G = (N, v)$, $|N| = n$, $v(C) = 1$ if $C = N$, $v(C) = 0$ otherwise
 - $\delta_i(C) = 1$ iff $C = N \setminus \{i\}$
 - $\phi_i(G) = (n-1)!/n! = 1/n$ for $i = 1, \dots, n$
 - $\beta_i(G) = 1/2^{n-1}$ for $i = 1, \dots, n$
- Example 2 (majority game):
 - $G = (N, v)$, $|N| = 2k$, $v(C) = 1$ if $|C| > k$, $v(C) = 0$ otherwise
 - $\delta_i(C) = 1$ iff $|C| = k$
 - $\phi_i(G) = (n-1)!/n! = 1/n$ for $i = 1, \dots, n$
 - $\beta_i(G) = 1/2^{n-1} \times (2k)!/(k!)^2 \approx 2/\sqrt{\pi k}$ for $i = 1, \dots, n$

Part 1: Overview

- Introduction
- Definitions
- Solution concepts
- Representations and computational issues

Computational Issues in Coalitional Games

- We have defined many solution concepts - but can we compute them **efficiently**?
- Problem: the **naive** representation of a coalitional game is **exponential** in the number of players **n**
 - need to **list values** of all coalitions
- We are usually interested in algorithms whose running time is **polynomial** in **n**
- So what can we do?

How to Deal with Representation Issues?

- Strategy 1: oracle representation
 - assume that we have a **black-box poly-time** algorithm that, given a coalition $C \subseteq N$, outputs its value $v(C)$
 - for some **special classes** of games, this allows us compute some solution concepts using **polynomially** many queries
- Strategy 2: restricted classes
 - consider games on **combinatorial structures**
 - **problem**: not all games can be represented in this way
- Strategy 3: give up on worst-case succinctness
 - devise **complete** representation languages that allow for **compact** representation of **interesting** games

Part 1: Overview

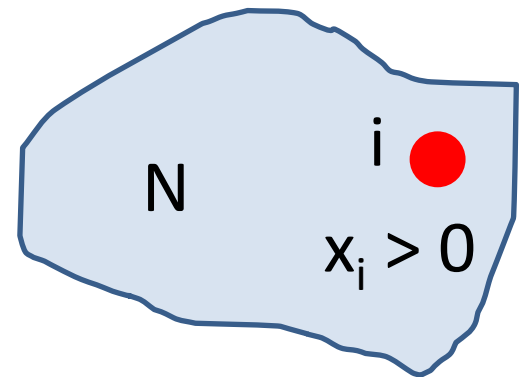
- Introduction
- Definitions
- Solution concepts
- Representations and computational issues
 - oracle representation
 - combinatorial optimization games
 - weighted voting games
 - complete representation languages

Simple Games

- Definition: a game $G = (N, v)$ is **simple** if
 - $v(C) \in \{0, 1\}$ for any $C \subseteq N$
 - v is **monotone**: if $v(C) = 1$ and $C \subseteq D$, then $v(D) = 1$
- A coalition C in a simple game is said to be **winning** if $v(C) = 1$ and **losing** if $v(C) = 0$
- Definition: in a simple game, a player i is a **veto** player if $v(C) = 0$ for any $C \subseteq N \setminus \{i\}$
 - equivalently, by monotonicity, $v(N \setminus \{i\}) = 0$
- Traditionally, in simple games an outcome is identified with a payoff vector for N
- Theorem: a simple game has a non-empty core iff it has a veto player.

Simple Games: Characterization of the Core

- Proof (\Leftarrow):
 - suppose i is a veto player
 - consider a payoff vector \underline{x} with $x_i = 1$, $x_k = 0$ for $k \neq i$
 - no coalition C can deviate from \underline{x} :
 - if $i \in C$, we have $\sum_{k \in C} x_k = 1 \geq v(C)$
 - if $i \notin C$, we have $v(C) = 0$
- Proof (\Rightarrow):
 - consider an arbitrary payoff vector \underline{x} :
 - we have $\sum_{k \in N} x_k = v(N) = 1$; thus $x_i > 0$ for some $i \in N$
 - but then $N \setminus \{i\}$ can deviate:
 - since i is not a veto, $v(N \setminus \{i\}) = 1$, yet $x(N \setminus \{i\}) = 1 - x_i < 1$

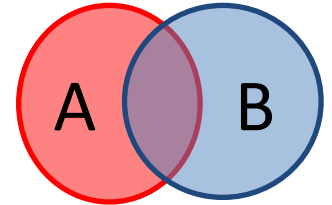


Simple Games: Checking Non-Emptiness of the Core

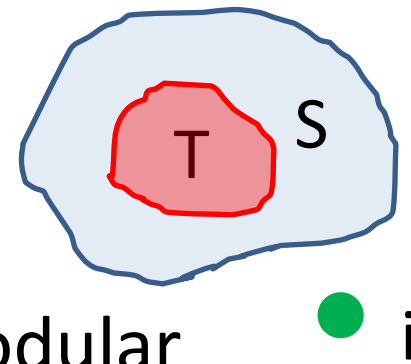
- Corollary: in a simple game G , a payoff vector \underline{x} is in the core iff $x_i = 0$ for any non-veto player i
 - proved similarly
- Checking if a player i is a veto player is easy
 - a single oracle access to compute $v(N \setminus \{i\})$
- Thus, in simple games
 - checking non-emptiness of the core or
 - checking if a given outcome is in the coreis easy given oracle access to the characteristic function
 - this is no longer the case if we allow coalition structures

Convex Games

- Definition: a function $f:2^N \rightarrow R$ is called **supermodular** if $f(\emptyset) = 0$ and $f(A \cup B) + f(A \cap B) \geq f(A) + f(B)$ for any $A, B \subseteq N$ (not necessarily disjoint)
 - any supermodular function is superadditive, but the converse is not true



- Proposition: if f is supermodular, $T \subset S$, and $i \notin S$, then $f(T \cup \{i\}) - f(T) \leq f(S \cup \{i\}) - f(S)$
 - a player is more useful when he joins a bigger coalition



- Definition: a game $G = (N, v)$ is **convex** if its characteristic function is supermodular

Convex Games: Non-Emptiness of The Core

- Proposition: any convex game has a **non-empty core**
- Proof:
 - set $x_1 = v(\{1\})$,
 $x_2 = v(\{1, 2\}) - v(\{1\})$,
...
 $x_n = v(N) - v(N \setminus \{n\})$
 - i.e., pay each player his marginal contribution to the coalition formed by his predecessors
 - \underline{x} is a payoff vector: $x_1 + x_2 + \dots + x_n =$
 $= v(\{1\}) + v(\{1, 2\}) - v(\{1\}) + \dots + v(N) - v(N \setminus \{n\}) = v(N)$
 - remains to show that (x_1, x_2, \dots, x_n) is in the core

Convex Games Have Non-Empty Core

- Proof (continued):
 - $x_1 = v(\{1\})$, $x_2 = v(\{1, 2\}) - v(\{1\})$, ..., $x_n = v(N) - v(N \setminus \{n\})$
 - pick any coalition $C = \{i, j, \dots, s\}$, where $i < j < \dots < s$
 - we will prove $v(C) \leq x_i + x_j + \dots + x_s$, i.e., C cannot deviate
 - $v(C) = v(\{i\}) + v(\{i, j\}) - v(\{i\}) + \dots + v(C) - v(C \setminus \{s\})$
 - $v(\{i\}) = v(\{i\}) - v(\emptyset) \leq v(\{1, \dots, i-1, i\}) - v(\{1, \dots, i-1\}) = x_i$
 - $v(\{i, j\}) - v(\{i\}) \leq v(\{1, \dots, j-1, j\}) - v(\{1, \dots, j-1\}) = x_j$
 -
 - $v(C) - v(C \setminus \{s\}) \leq v(\{1, \dots, s-1, s\}) - v(\{1, \dots, s-1\}) = x_s$
 - thus, $v(C) \leq x_i + x_j + \dots + x_s$



Convex Games: Remarks

- This proof suggests a simple algorithm for **constructing** an outcome in the core
 - order the players as $1, \dots, n$
 - query the oracle for $v(\{1\}), v(\{1, 2\}), \dots, v(N)$
 - set $x_i = v(\{1, \dots, i-1, i\}) - v(\{1, \dots, i-1\})$
- This argument also shows that for convex games the **Shapley value** is in the **core**
 - the core is a **convex** set
 - Shapley value is a **convex combination** of outcomes in the core

Checking Non-emptiness of the Core: Superadditive Games

- An outcome in the core of a superadditive game satisfies the following constraints:
 - $x_i \geq 0$ for all $i \in N$
 - $\sum_{i \in N} x_i = v(N)$
 - $\sum_{i \in C} x_i \geq v(C)$ for any $C \subseteq N$
- A linear feasibility program, with one constraint for each coalition: $2^n + n + 1$ constraints
 - sometimes can be solved in polynomial time solvers using [separation oracles](#)

Superadditive Games: Computing the Least Core

- LFP for the core  LP for the least core

$\min \varepsilon$

$$x_i \geq 0 \text{ for all } i \in N$$

$$\sum_{i \in N} x_i = v(N)$$

$$\sum_{i \in C} x_i \geq v(C) - \varepsilon \text{ for any } C \subseteq N$$

- A **minimization** program, rather than a **feasibility** program
 - sometimes can be solved in polynomial time using a separation oracle

Core and Related Concepts: Non-Superadditive Games

- What if the game is not superadditive?
- Can solve a similar LFP for each coalition structure $CS = (C^1, \dots, C^k)$:
 - $x_i \geq 0$ for all $i \in N$
 - $\sum_{i \in C^1} x_i = v(C^1)$
 - ...
 - $\sum_{i \in C^k} x_i = v(C^k)$
 - $\sum_{i \in C} x_i \geq v(C)$ for any $C \subseteq N$
- Running time: # of partitions of N x time to solve an exp -sized LFP - infeasible in general.

Part 1: Overview

- Introduction
- Definitions
- Solution concepts
- Representations and computational issues
 - oracle representation
 - combinatorial optimization games
 - weighted voting games
 - complete representation languages

Weighted Voting Games

- n parties in the parliament
- Party i has w_i representatives
- A coalition of parties can form a government only if its total size is at least q
 - usually $q \geq \lfloor \sum_{i=1, \dots, n} w_i / 2 \rfloor + 1$: strict majority
- Notation: $w(C) = \sum_{i \in C} w_i$
- This setting can be described by a game $G = (N, v)$, where
 - $N = \{1, \dots, n\}$
 - $v(C) = 1$ if $w(C) \geq q$ and $v(C) = 0$ otherwise
- Observe that weighted voting games are simple games
- Notation: $G = [q; w_1, \dots, w_n]$
 - q is called the **quota**

Weighted Voting Games: UK

- United Kingdom, 2005:
 - 650 seats, $q = 326$
 - Conservatives (C): 196
 - Labour (L): 354
 - Liberal Democrats (LD): 62
 - 8 other parties (O), with a total of 38 seats
- $N = \{C, L, LD, O\}$
- for any $X \subseteq N$, $v(X) = 1$ if and only if $L \in X$
- L is a veto player, C, LD, and O are dummies
- $\phi_L = 1$, $\phi_C = \phi_{LD} = \phi_O = 0$



Weighted Voting Games: UK

- United Kingdom, 2010:
 - 650 seats, $q = 326$
 - Conservatives (C): 307
 - Labour (L): 258
 - Liberal Democrats (LD): 57
 - 8 other parties (O), with a total of 28 seats



- $N = \{C, L, LD, O\}$
- $v(\{C, L\}) = v(\{C, LD\}) = v(\{C, O\}) = 1$
- $v(\{L, LD\}) = v(\{L, O\}) = v(\{LD, O\}) = 0, v(\{L, LD, O\}) = 1$
- L, LD and O are symmetric
- $\phi_C = 1/2, \phi_L = \phi_{LD} = \phi_O = 1/6$

Weighted Voting Games as Resource Allocation Games

- Each agent i has a certain amount of a resource w_i
 - time or money or battery power
- One or more tasks with a resource requirement q and a value V
- If a coalition has enough resources to complete the task (q or more units), it earns its value V , else it earns 0
 - By normalization, can assume $V = 1$
- If $q < \sum_i w_i/2$, grand coalition need not form
 - weighted voting games with coalition structures

Shapley Value in Weighted Voting Games

- In a simple game $G = (N, v)$, a player i is said to be **pivotal**
 - for a coalition $C \subseteq N$ if $v(C) = 0$, $v(C \cup \{i\}) = 1$
 - for a permutation $\pi \in P(N)$ if he is pivotal for $S_\pi(i)$
- In simple games player i 's Shapley value = $\Pr[i \text{ is pivotal for a random permutation}]$
 - measure of **voting power**
- Shapley value is widely used to measure power in various voting bodies
- UK elections'10 illustrate that **power \neq weight**

Weighted Voting Games: Computational Aspects

- Deciding if a player is a dummy: **coNP**-complete
- Computing Shapley value and Banzhaf index:
 - **#P**-complete [Deng & Papadimitriou'94]
 - **hard** to approximate
- Computing the core/checking if an outcome is in the core:
 - **poly-time** (since WVG are simple games)
 - if we allow coalition structures, these problems become computationally **hard** [Elkind et al.'08b]

Weighted Voting Games: Small Weights

- Suppose all weights are at most polynomial in n
 - realistic in many applications
- Then
 - Shapley value and Banzhaf index can be computed in **poly-time** by dynamic programming [Matsui & Matsui'00]
 - value of the least core is **poly-time** computable [Elkind et al.'09a]
 - nucleolus is **poly-time** computable [Elkind and Pasechnik'09]

WVG and Simple Games

- WVGs are simple games
- Can every simple game be represented as a WVG?
- $G = (N, v)$:
 - $N = \{1, 2, 3, 4\}$
 - $v(C) = 1$ iff $C \cap \{1, 3\} \neq \emptyset$ and $C \cap \{2, 4\} \neq \emptyset$

- Suppose $G = [q; w_1, w_2, w_3, w_4]$

$$w_1 + w_2 \geq q, \quad w_3 + w_4 \geq q$$

$$w_1 + w_2 + w_3 + w_4 \geq 2q$$

$$w_1 + w_3 < q, \quad w_2 + w_4 < q$$

$$w_1 + w_2 + w_3 + w_4 < 2q$$

} a contradiction!

A Generalization: Vector Weighted Voting Games

- The game in the previous slide can be thought of as a combination of two WVGs:
 - $G^{\text{odd}} = [1; 1, 0, 1, 0]$ and $G^{\text{even}} = [1; 0, 1, 0, 1]$
 - to win, a coalition needs to win in both games
- Definition: a k -weighted voting game is a tuple $[N; \mathbf{q}; \underline{\mathbf{w}}_1, \dots, \underline{\mathbf{w}}_n]$, where $|N| = n$ and
 - $\mathbf{q} = (q^1, \dots, q^k)$ is a vector of k real quotas
 - for each $i \in N$, $\underline{\mathbf{w}}_i = (w_i^1, \dots, w_i^k)$ is a vector of k real weights
- $v(C) = 1$ if $\sum_{i \in C} w_i^j \geq q^j$ for each $j = 1, \dots, k$ and $v(C) = 0$ otherwise

Vector Weighted Voting Games

- Given a k -VWVG $G = [N; \mathbf{q}; \underline{\mathbf{w}}_1, \dots, \underline{\mathbf{w}}_n]$, we can define $G^j = [q^j; w^j_1, \dots, w^j_n]$
- G^j is a weighted voting game
 - we will refer to G^j as the j -th component of G
- To win in G , a coalition needs to win in each of the component games
 - we can write $G = G^1 \wedge \dots \wedge G^k$
 - thus, G is a conjunction of its component games
- a k -VWG models a resource allocation games with k types of resources
 - each task needs q^j units of resource j

VWVG in the Wild: EU Voting

- Voting in the European Union is a 3-WVG

$G = G^1 \wedge G^2 \wedge G^3$, where

- G^1 corresponds to commissioners
- G^2 corresponds to countries
- G^3 corresponds to population



- The players are the 27 member states:
Germany, UK, France, Italy, Spain, Poland,
Romania, The Netherlands, Greece, Czech
Republic, Belgium, Hungary, Portugal, Sweden,
Bulgaria, Austria, Slovak Republic, Denmark,
Finland, Ireland, Lithuania, Latvia, Slovenia,
Estonia, Cyprus, Luxembourg, Malta.

EU Voting Game

- $G^1 = [255; 29, 29, 29, 29, 27, 27, 14, 13, 12, 12, 12, 12, 12, 10, 10, 10, 7, 7, 7, 7, 7, 4, 4, 4, 4, 4, 3]$
- $G^2 = [14; 1, 1]$
- $G^3 = [620; 170, 123, 122, 120, 82, 80, 47, 33, 22, 21, 21, 21, 21, 18, 17, 17, 11, 11, 11, 8, 8, 5, 4, 3, 2, 1, 1]$
 – UK, Greece, Estonia
- For a proposal to pass, it needs to be supported by
 - 74% of the commissioners
 - 50% of the member states
 - 62% of the EU population

VWVGs and Simple Games

- VWVGs are **strictly more expressive** than WVGs
- Theorem: **any** simple game can be represented as a **vector** weighted voting game
- Proof: consider a simple game $G=(N, v)$
 - for each losing coalition $C \subseteq N$, we construct a game $G^C = [q^C; w^C_1, \dots, w^C_n]$ as follows:
 - $q^C = 1, w^C_i = 1$ if $i \notin C$ and $w^C_i = 0$ if $i \in C$
 - D loses in G^C iff $D \subseteq C$
 - Let $G^* = \bigwedge_{v(C)=0} G^C$
 - if $v(D) = 0$, D loses in G^D and hence in G^*
 - if $v(D) = 1$, by monotonicity D wins in each component game and hence in G^*

Dimensionality

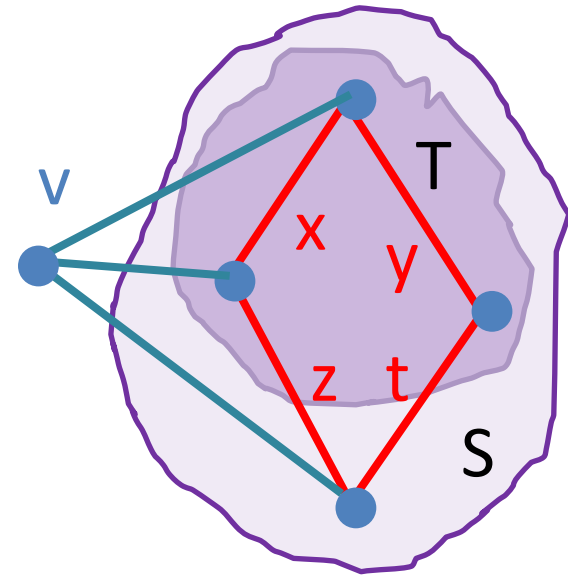
- Vector weighted voting games form a **complete** representation language for simple games
- However, the construction in the previous slide may use **exponentially many** component games
- Definition: the **dimension** $\dim(G)$ of a simple game G is the minimum number of components in its VWVG representation
 - every simple game has dimension $O(2^n)$
 - there exist simple games of dimension $\Omega(2^{n/2-1})$

Part 1: Overview

- Introduction
- Definitions
- Solution concepts
- Representations and computational issues
 - oracle representation
 - combinatorial optimization games
 - weighted voting games
 - complete representation languages

Induced Subgraph Games

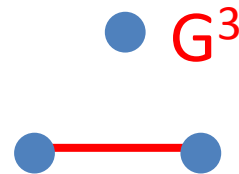
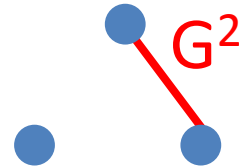
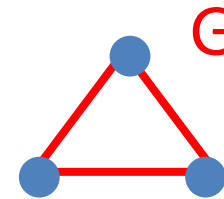
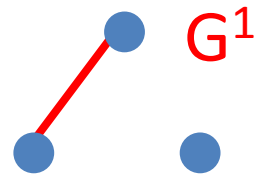
- Players are vertices of a weighted graph
- Value of a coalition = total weight of internal edges
 - $v(T) = x+y$, $v(S) = x+y+z+t$
- Models social networks
 - Facebook, LinkedIn
 - cell phone companies with free in-network calls
- If all edge weights are non-negative, this game is convex:
 - $\delta_v(S) \geq \delta_v(T)$



Induced Subgraph Games: Complexity

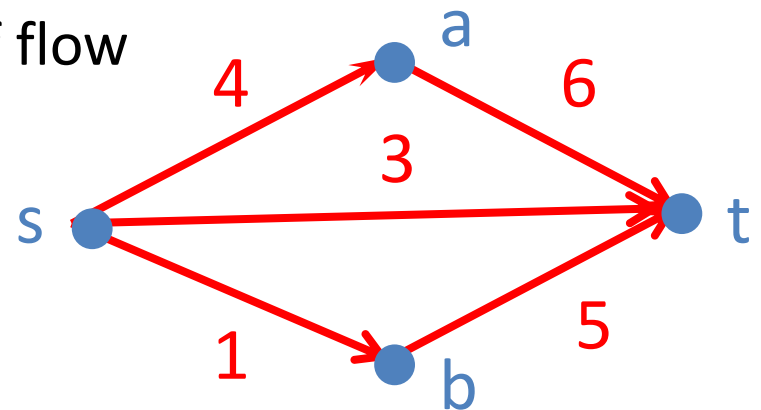
[Deng, Papadimitriou'94]

- If all edge weights are **non-negative**, the core is non-empty
 - also, we can check in **poly-time** if a given outcome is in the core
- In general, determining emptiness of the core is **NP-complete**
- Shapley value is **easy** to compute:
 - let $E = \{e^1, \dots, e^k\}$ be the list of edges of the graph
 - let G^j be the induced subgraph game on the graph that contains edge e^j only
 - we have $G = G^1 + \dots + G^k$
 - $\phi_i(G^j) = w(e^j)/2$ if e^j is adjacent to i and 0 otherwise
 - $\phi_i(G) = (\text{weight of edges adjacent to } i)/2$



Network Flow Games

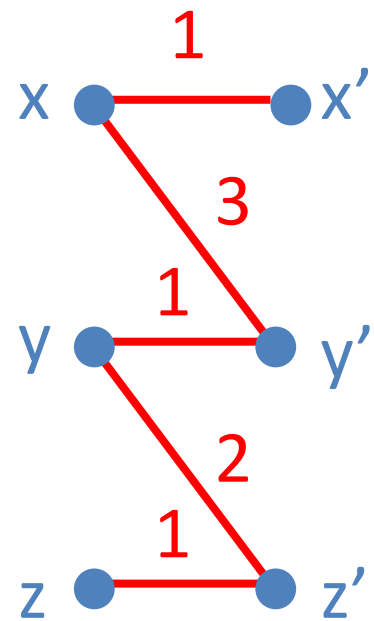
- Agents are edges in a network with source s and sink t
 - edge e_i has capacity c_i
- Value of a coalition = amount of $s-t$ flow it can carry
 - $v(\{sa, at\}) = 4$, $v(\{sa, at, st\}) = 7$
- Thresholded network flow games (TNFG):
there exists a threshold T such that
 - $v(C) = 1$ if C can carry $\geq T$ units of flow
 - $v(C) = 0$ otherwise
- TNFG with $T = 6$
 - $v(\{sa, at\}) = 0$, $v(\{sa, at, st\}) = 1$



Assignment Games

[Shapley & Shubik'72]

- Players are vertices of a bipartite graph (V, W, E)
- Value of a coalition = weight of the max-weight induced matching
 - $v(\{x, y, z\}) = 0$, $v(\{x, x', y'\}) = 3$
- Generalization: matching games
 - same definition, but the graph need not be bipartite



Part 1: Overview

- Introduction
- Definitions
- Solution concepts
- Representations and computational issues
 - oracle representation
 - combinatorial optimization games
 - weighted voting games
 - complete representation languages

Coalitional Skill Games

[Bachrach & Rosenschein'08]

- Set of skills $S = \{s_1, \dots, s_k\}$
- Set of agents N : agent i has a subset of skills $S_i \subseteq S$
- Set of tasks $T = \{t_1, \dots, t_m\}$
 - each task t_j requires a subset of skills $S(t_j) \subseteq S$
- A skill set of a coalition C : $s(C) = \bigcup_{i \in C} S_i$
- Tasks that C can perform: $T(C) = \{t_j \mid S(t_j) \subseteq S(C)\}$
- Utility function $u : 2^T \rightarrow \mathbb{R}$
 - e.g., sum or max of values of individual tasks
- Characteristic function: $v(C) = u(T(C))$

Coalitional Skill Games: Expressiveness and Complexity

- Any **monotone** game can be expressed as a CSG:
 - given a game $G = (N, v)$,
we create a task t^C and set $u(t^C) = v(C)$ for any $C \subseteq N$
 - each agent i has a unique skill s_i
 - t^C requires the skills of all agents in C
 - set $u(T') = \max \{ u(t) \mid t \in T' \}$
 - $u(T(C)) = \max \{ u(t^D) \mid D \subseteq C \} = \max \{ v(D) \mid D \subseteq C \} = v(C)$
- However, the representation is only **succinct** when the game is naturally defined via a **small set of tasks**
- [Bachrach&Rosenschein'08] discuss complexity of many solution concepts under this formalism

Synergy Coalition Games

[Conitzer & Sandholm'06]

- Superadditive game: $v(C \cup D) \geq v(C) + v(D)$ for any two disjoint coalitions C and D
- Idea: if a game is superadditive, and $v(C) = v(C_1) + \dots + v(C_k)$ for any partition (C_1, \dots, C_k) of C (no synergy), no need to store $v(C)$
- Representation: list $v(\{1\}), \dots, v(\{n\})$ and all synergies
- Succinct when there are few synergies
- This representation allows for efficient checking if an outcome is in the core.
- However, it is still hard to check if the core is non-empty.

Marginal Contribution Nets

[Jeong & Shoham'05]

- Idea: represent the game by a set of rules of the form **pattern** \rightarrow **value**
 - **pattern** is a Boolean formula over **N**
 - **value** is a number
- A rule **applies** to a coalition if its fits the pattern
- $v(C)$ = **sum** of values of all rules that apply to **C**
- Example:
 - $R_1: (1 \wedge 2) \vee 5 \rightarrow 3$
 - $R_2: 2 \wedge 3 \rightarrow -2$
 - $v(\{1, 2\}) = 3, v(\{2, 3\}) = -2, v(\{1, 2, 3\}) = 1$

Marginal Contribution Nets

- Computing the Shapley value:
 - let $G(R_1, \dots, R_k)$ be the game given by the set of rules R_1, \dots, R_k
 - we have $G(R_1, \dots, R_k) = G(R_1) + \dots + G(R_k)$
 - thus, by **additivity** it suffices to compute players' Shapley values in games with a **single** rule R
 - if $R = \psi \rightarrow x$, where ψ is a **conjunction** of k variables, then $\phi_i = x/k$ if i appears in ψ and 0 otherwise
 - a more complicated (but still poly-time) algorithm for **read-once formulas** [Elkind et al.'09b]
 - **NP-hard** for if ψ is an **arbitrary** Boolean formula
- Core-related questions are computationally hard [leong&Shoham'05]

Coalition Structure Generation

How do we **partition the set of agents** into coalitions to maximize the overall profit?

The Coalition Structure Generation Problem

Example: given 3 agents, the possible coalitions are:

$\{a_1\}$ $\{a_2\}$ $\{a_3\}$ $\{a_1, a_2\}$ $\{a_1, a_3\}$ $\{a_2, a_3\}$ $\{a_1, a_2, a_3\}$

The possible coalition structures are:

$\{\{a_1\}, \{a_2\}, \{a_3\}\}$ $\{\{a_1, a_2\}, \{a_3\}\}$ $\{\{a_2\}, \{a_1, a_3\}\}$ $\{\{a_1\}, \{a_2, a_3\}\}$ $\{\{a_1, a_2, a_3\}\}$

The input is the characteristic function

$$v(\{a_1\}) = 20$$

$$v(\{a_2\}) = 40$$

$$v(\{a_3\}) = 30$$

$$v(\{a_1, a_2\}) = 70$$

$$v(\{a_1, a_3\}) = 40$$

$$v(\{a_2, a_3\}) = 65$$

$$v(\{a_1, a_2, a_3\}) = 95$$

What we want as output is a coalition structure in which the sum of values is maximized

$$V(\{\{a_1\}, \{a_2\}, \{a_3\}\}) = 20 + 40 + 30 = 90$$

$$V(\{\{a_1, a_2\}, \{a_3\}\}) = 70 + 30 = 100$$

$$V(\{\{a_2\}, \{a_1, a_3\}\}) = 40 + 40 = 80$$

$$V(\{\{a_1\}, \{a_2, a_3\}\}) = 20 + 65 = 85$$

$$V(\{\{a_1, a_2, a_3\}\}) = 95$$

optimal coalition structure

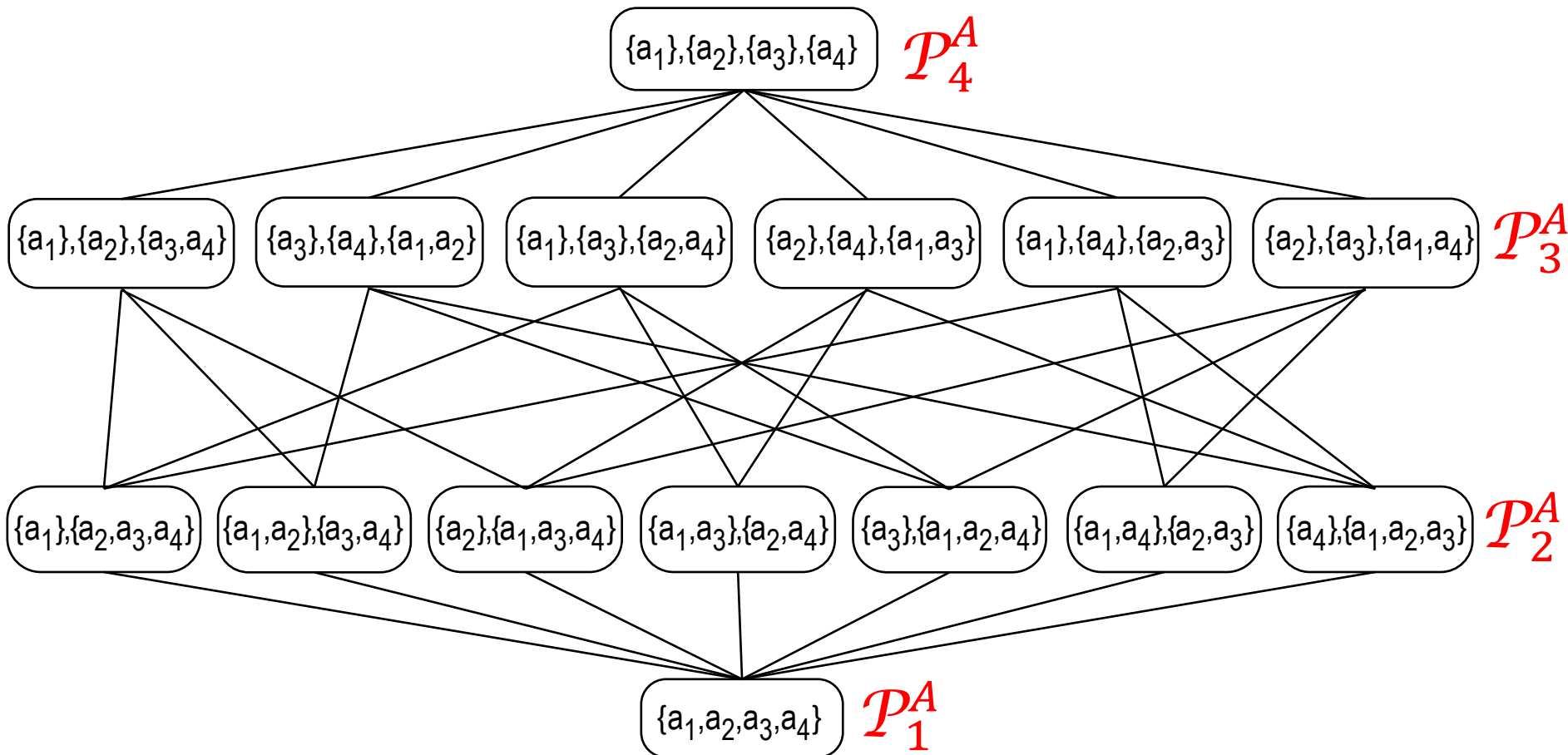
Coalition Structure Generation

How should we solve this problem?

We will present multiple algorithms, but first we need to present the main **representations** of the search space (the set of coalition structures, denoted as \mathcal{P}^A)

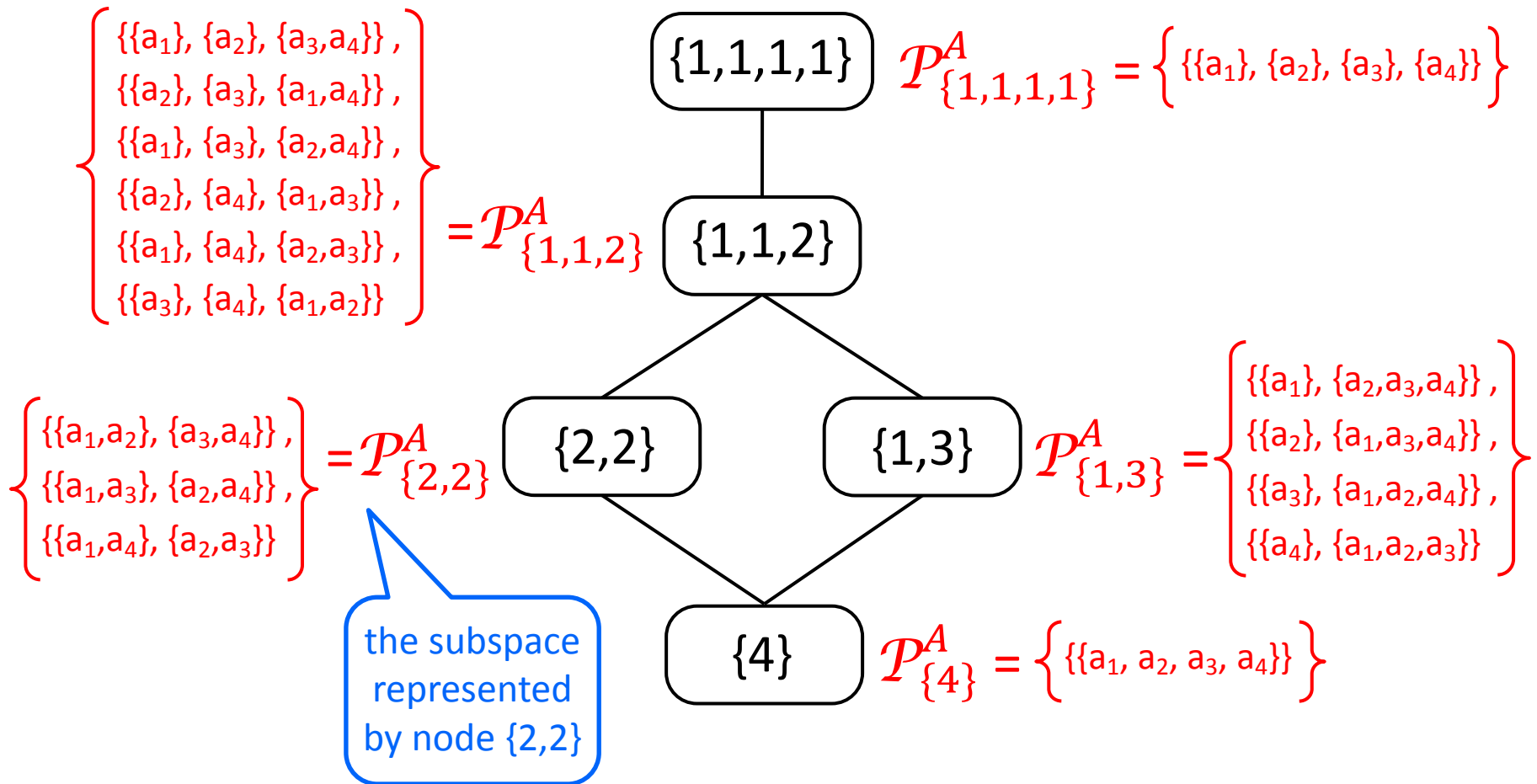
The Coalition Structure Graph (example of 4 agents)

$\mathcal{P}_i^A \subseteq \mathcal{P}^A$ contains all coalition structures that consist of exactly i coalitions



The Integer Partition Graph (example of 4 agents)

Every node represents a subspace (coalition sizes match the integers in that node)



Coalition Structure Generation

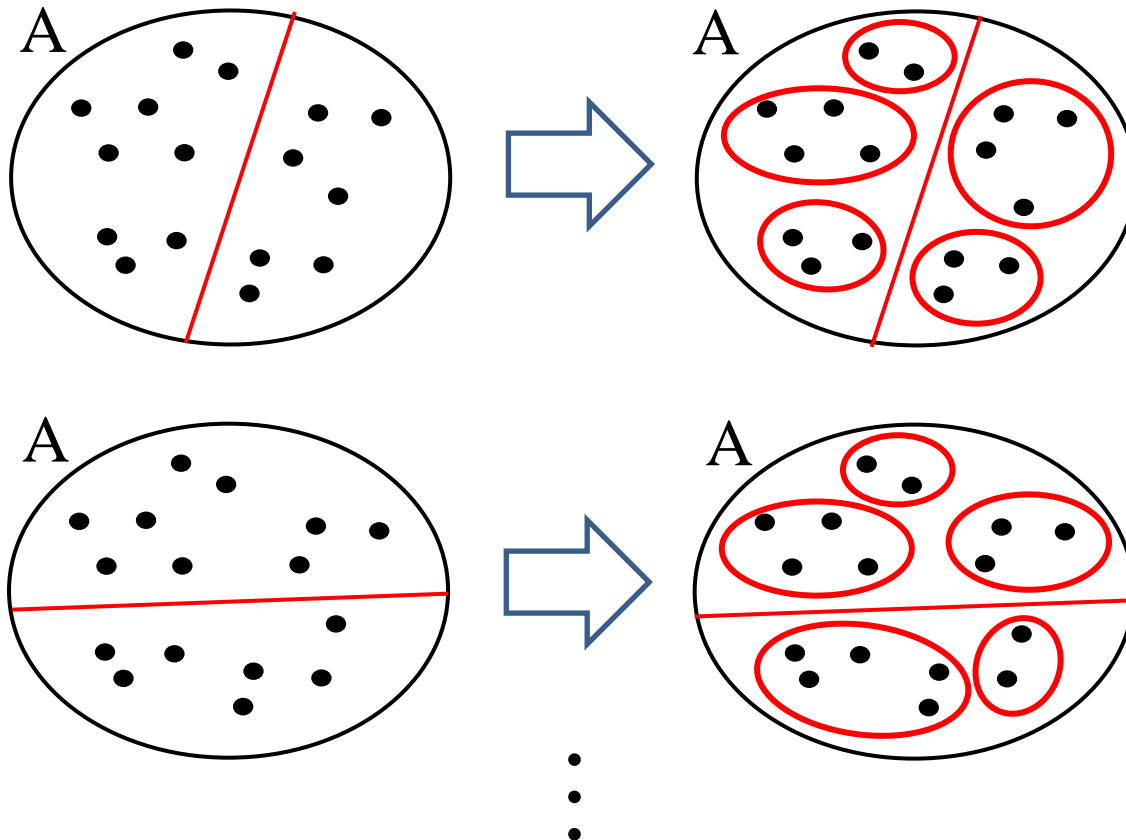
Solving the problem using **Dynamic Programming**

The Dynamic Programming (DP) Algorithm

Main observation:

To examine all coalition structure $CS : |CS| \geq 2$, it is sufficient to:

- try the possible ways to split the **set of agents into two sets**, and
- for every half, find **the optimal partition** of that half.



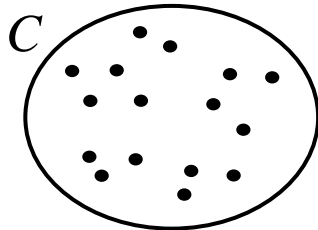
The Dynamic Programming (DP) Algorithm

Main theorem:

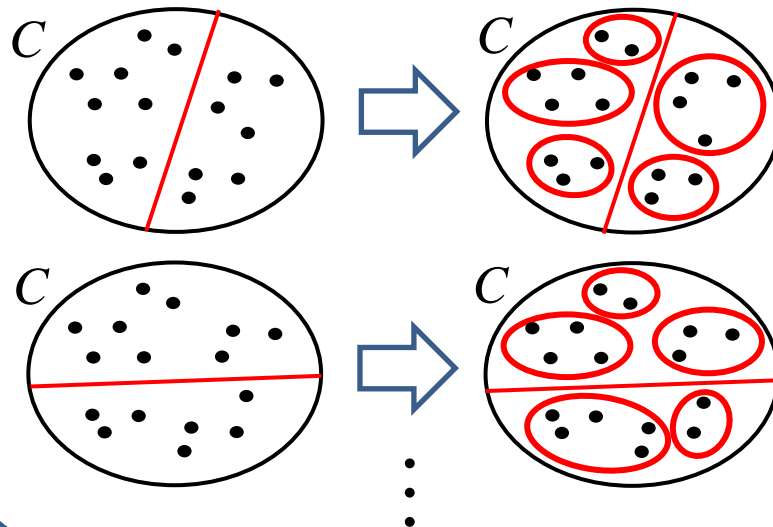
Given a coalition $C \in A$, let \mathcal{P}^C be the set of partitions of C , and let $f(C)$ be the value of an optimal partition of C , i.e., $f(C) = \max_{P \in \mathcal{P}^C} v(P)$. Then,

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

the value of the coalition
itself (without partitioning)



the maximum value for all such partitions



The Dynamic Programming (DP) Algorithm

Algorithm:

- Iterate over all coalitions $C:|C|=1$, then over all $C:|C|=2$, then all $C:|C|=3$, etc.
- For every coalition, C , compute $f(C)$ using the above equation
- While computing $f(C)$:
 - the algorithm stores in $t(C)$ the best way to split C in two
 - unless it is more beneficial to keep C as it is (i.e., without splitting)
- By the end of this process, $f(A)$ will be computed, which is by definition the value of the optimal coalition structure
- It remains to compute the optimal coalition structure itself, by using $t(A)$

Consider the following example of 4 agents

Example:

input:

$$v(\{1\}) = 30$$

$$v(\{2\}) = 40$$

$$v(\{3\}) = 25$$

$$v(\{4\}) = 45$$

$$v(\{1,2\}) = 50$$

$$v(\{1,3\}) = 60$$

$$v(\{1,4\}) = 80$$

$$v(\{2,3\}) = 55$$

$$v(\{2,4\}) = 70$$

$$v(\{3,4\}) = 80$$

$$v(\{1,2,3\}) = 90$$

$$v(\{1,2,4\}) = 120$$

$$v(\{1,3,4\}) = 100$$

$$v(\{2,3,4\}) = 115$$

$$v(\{1,2,3,4\}) = 140$$

coalition	evaluations performed before setting f		t	f
step 1 {1} {2} {3} {4}	$v(\{1\})=30$	$f(\{1\})=30$	{1}	30
	$v(\{2\})=40$	$f(\{2\})=40$	{2}	40
	$v(\{3\})=25$	$f(\{3\})=25$	{3}	25
	$v(\{4\})=45$	$f(\{4\})=45$	{4}	45
step 2 {1,2} {1,3} {1,4} {2,3} {2,4} {3,4}	$v(\{1,2\})=50$	$f(\{1\})+f(\{2\})=70$	{1} {2}	70
	$v(\{1,3\})=60$	$f(\{1\})+f(\{3\})=55$	{1,3}	60
	$v(\{1,4\})=80$	$f(\{1\})+f(\{4\})=75$	{1,4}	80
	$v(\{2,3\})=55$	$f(\{2\})+f(\{3\})=65$	{2} {3}	65
	$v(\{2,4\})=70$	$f(\{2\})+f(\{4\})=85$	{2} {4}	85
	$v(\{3,4\})=80$	$f(\{3\})+f(\{4\})=70$	{3,4}	80
	step 3 {1,2,3} {1,2,4} {1,3,4} {2,3,4}	$v(\{1,2,3\})=90$	$f(\{1\})+f(\{2,3\})=95$	{2} {1,3}
$f(\{2\})+f(\{1,3\})=100$		$f(\{3\})+f(\{1,2\})=95$		
$v(\{1,2,4\})=120$		$f(\{1\})+f(\{2,4\})=115$	{1,2,4}	120
$f(\{2\})+f(\{1,4\})=110$		$f(\{4\})+f(\{1,2\})=115$		
$v(\{1,3,4\})=100$		$f(\{1\})+f(\{3,4\})=110$	{1} {3,4}	110
$f(\{3\})+f(\{1,4\})=105$	$f(\{4\})+f(\{1,3\})=105$			
$v(\{2,3,4\})=115$	$f(\{2\})+f(\{3,4\})=120$	{2} {3,4}	120	
$f(\{3\})+f(\{2,4\})=110$	$f(\{4\})+f(\{2,3\})=110$			
step 4 {1,2,3,4}	$v(\{1,2,3,4\})=140$	$f(\{1\})+f(\{2,3,4\})=150$	{1,2}	step 5 150
	$f(\{2\})+f(\{1,3,4\})=150$	$f(\{3\})+f(\{1,2,4\})=145$	{3,4}	
	$f(\{4\})+f(\{1,2,3\})=145$	$f(\{1,2\})+f(\{3,4\})=150$		
	$f(\{1,3\})+f(\{2,4\})=145$	$f(\{1,4\})+f(\{2,3\})=145$		

The Dynamic Programming (DP) Algorithm

Note:

- While DP is guaranteed to find an optimal coalition structure, many of its operations were shown to be redundant
- An improved dynamic programming algorithm (called IDP) was developed that avoids all redundant operations

Advantage:

- IDP is the fastest algorithm that finds an optimal coalition structure in $O(3^n)$

Disadvantage:

- IDP provides no interim solutions before completion, meaning that it is not possible to trade computation time for solution quality.

Based on this, we present in the following slides algorithms that allow such a trade off.

Coalition Structure Generation

“Anytime” algorithms

Anytime Algorithms

Introduction

Definition:

An “*anytime*” algorithm is one whose solution quality improves gradually as computation time increases.

This way, an interim solution is always available in case the algorithm run to completion

Advantages:

- agents might not have time to run the algorithm to completion
- Being anytime makes the algorithm more robust against failure.

Notation:

We will denote the optimal coalition structure as CS^*

Anytime Algorithms

1. Algorithms based on Identifying Subspaces with Worst-Case Guarantees

Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

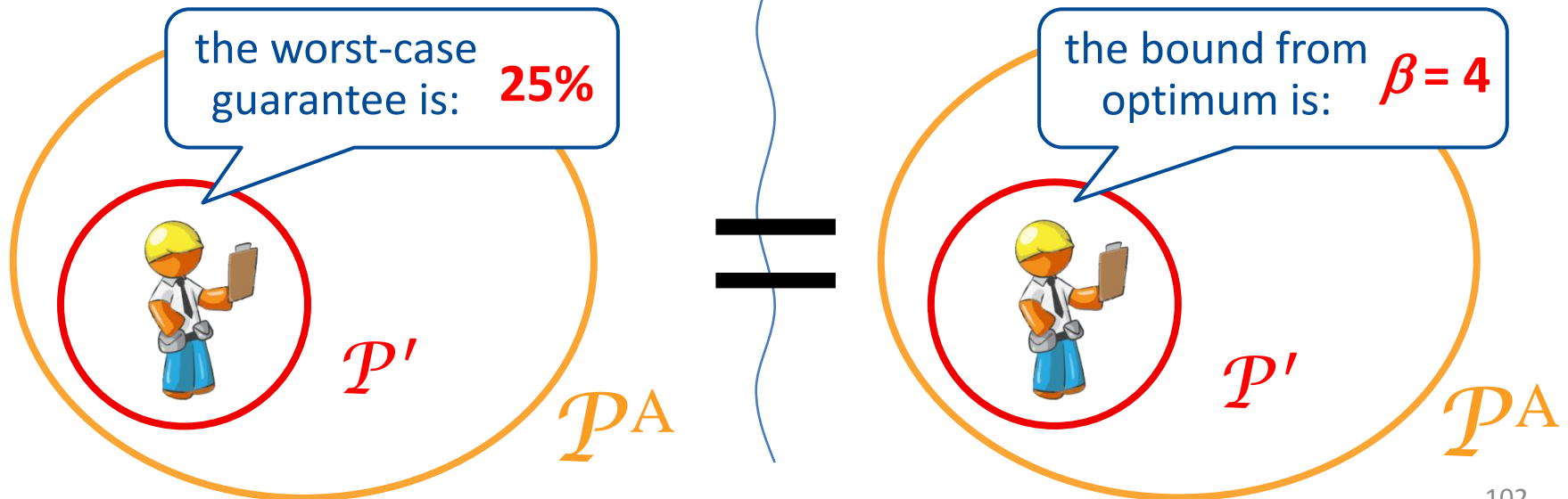
The optimal solution is:

$$CS^* = \operatorname{argmax}_{CS \in \mathcal{P}^A} V(CS)$$

is there a subset $\mathcal{P}' \subseteq \mathcal{P}^A$ with worst-case guarantees on solution quality?

i.e., a subset $\mathcal{P}' \subseteq \mathcal{P}^A$ guaranteed to contain a solution that is within a bound β from optimum? That is,

$$\frac{\operatorname{argmax}_{CS \in \mathcal{P}^A} V(CS)}{\operatorname{argmax}_{CS \in \mathcal{P}^A} V(CS)} \leq \beta$$

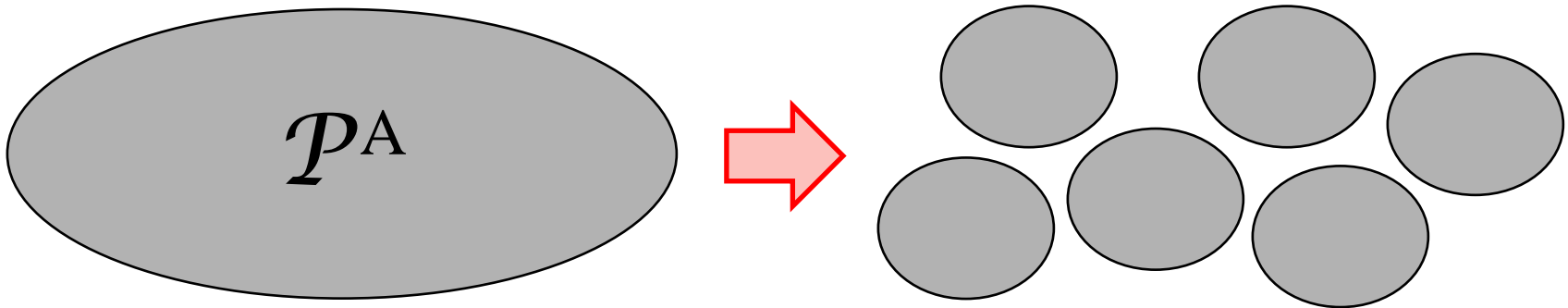


Anytime Algorithms

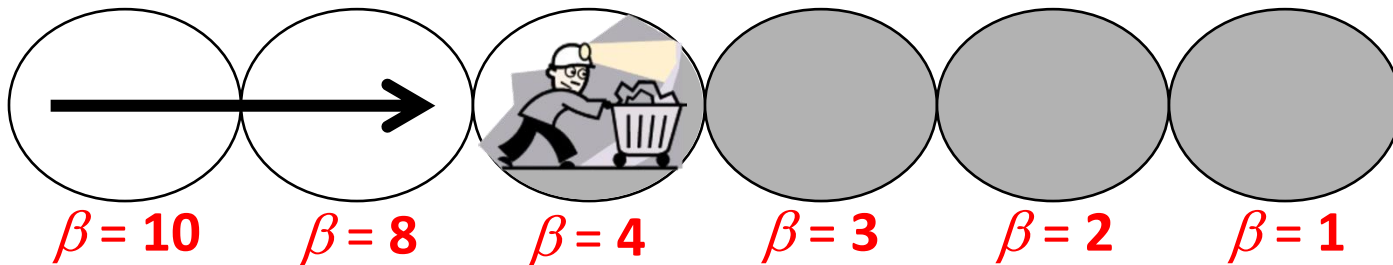
Identifying Subspaces with Worst-Case Guarantees

Based on this idea, we can design an anytime algorithm:

1. **Divide the search space** into subsets according to some criterion



2. **Identify a sequence** in which these subsets must be searched such that the bound β is improved after each subset



Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

Theorem:

- To establish a worst-case bound, β , it is sufficient to search the lowest 2 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} .
- No algorithm can establish any bound by search a different set of at most 2^{n-1} coalition structures

Proof:

- For a partial search to establish a bound, every $C \subseteq A$ must appear in at least one of the searched coalition structures.
 - The grand coalition appears in \mathcal{P}_1^A , and
 - every other coalition $C \subset A$ appears in $\{C, A \setminus C\} \in \mathcal{P}_2^A$, thus,
 - the value of the best coalition structure in $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ is at least $\max_{C \subseteq A} v(C)$
- Since CS^* include at most n coalitions, then: $v(CS^*) \leq n \times \max_{C \subseteq A} v(C)$. Thus,

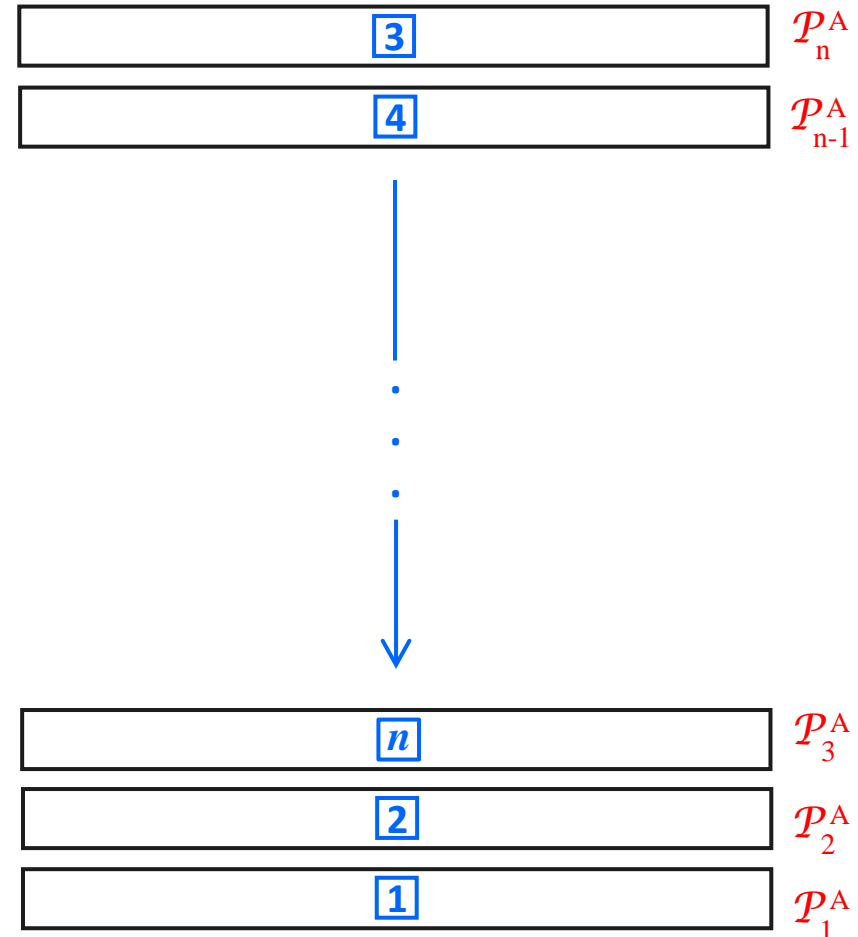
$$\frac{V(CS^*)}{\max_{CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A} V(CS)} \leq n$$

Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

- an *anytime* algorithm was proposed that searches the coalition structure graph one level at a time (see sequence in figure).
- The authors showed how the bound improves once the algorithm finishes searching every level $\mathcal{P}_i^A : i = n, \dots, 3$ (see [Sandholm et al, AIJ-1999] for details).
- An important result is that, after searching $\mathcal{P}_1^A \cup \mathcal{P}_2^A$ (which gives a bound $\beta = n$), it is possible to drop the bound to $\beta = \lceil n/2 \rceil$ by searching \mathcal{P}_n^A (which only contains one coalition structure!)

i Search steps of Sandholm et al. [1998]



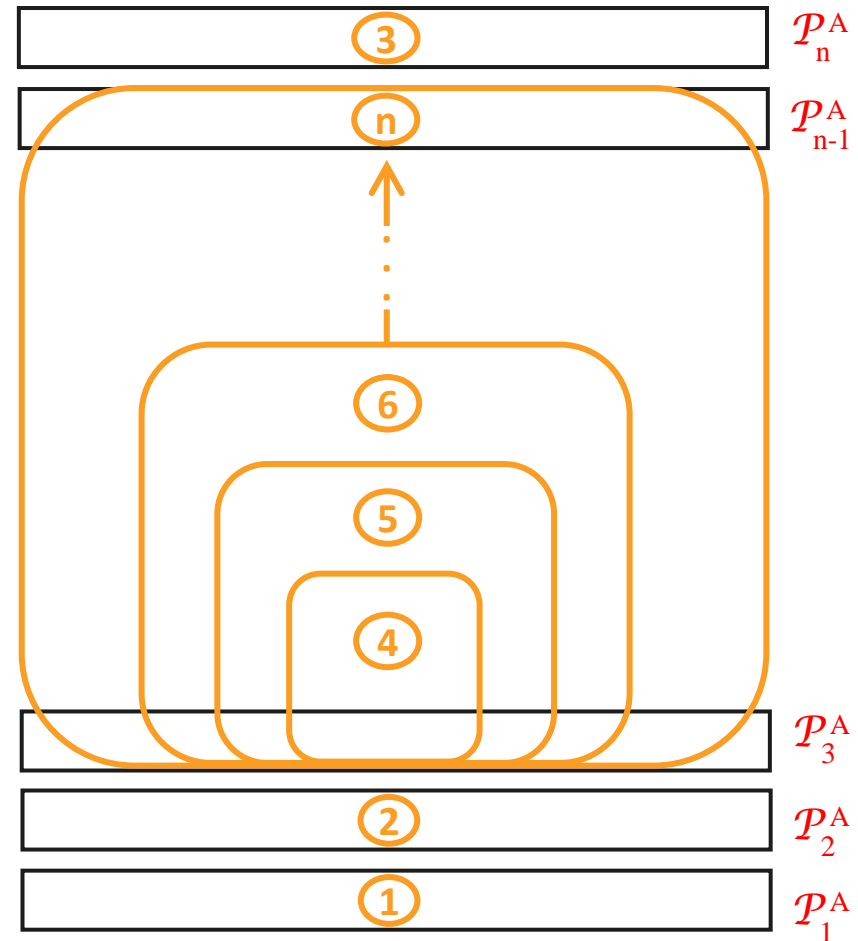
Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

A different algorithm was proposed.

- It starts by searching $\mathcal{P}_1^A \cup \mathcal{P}_2^A \cup \mathcal{P}_n^A$ as before.
- Then, it searches certain subsets of all remaining levels. Specifically:
 - It searches all 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).
- It has been shown that, for any given value of d , the algorithm establishes a bound $\beta = 2d - 1$

j Search steps of Dang & Jennings [2004]



Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

So far,

- we only know how to specify the bound after searching the aforementioned subsets of coalition structures.

but how can we specify a tight bound for any subset ?

- We know the *minimum* subset to be searched to establish $\beta = n$ (which is $\mathcal{P}_1^A \cup \mathcal{P}_2^A$) and $\beta = n-1$ (which is $\mathcal{P}_1^A \cup \mathcal{P}_2^A \cup \mathcal{P}_n^A$)

But what is the minimum search to establish any bound $\beta = b$?

Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

Remember:

We proved that, by searching $\mathcal{P}_1^A \cup \mathcal{P}_2^A$, we get a solution that is within a bound $\beta=n$ from the optimal solution (i.e., CS^*)

The intuition was:

- CS^* contains **at most n coalition**, and
- every one of those coalitions appears in some $CS \in \mathcal{P}_1^A \cup \mathcal{P}_2^A$, so
- CS^* **cannot be more than n times better** than the best solution in $\mathcal{P}_1^A \cup \mathcal{P}_2^A$

Main idea:

We can generalize this to “*groups of coalitions*”, i.e.,

- CS^* contains **at most x groups of coalition**, and
- every one of those groups appears in some $CS \in \mathcal{P}' \subseteq \mathcal{P}^A$, so
- CS^* **cannot be more than x times better** than the best solution in \mathcal{P}'

Anytime Algorithms

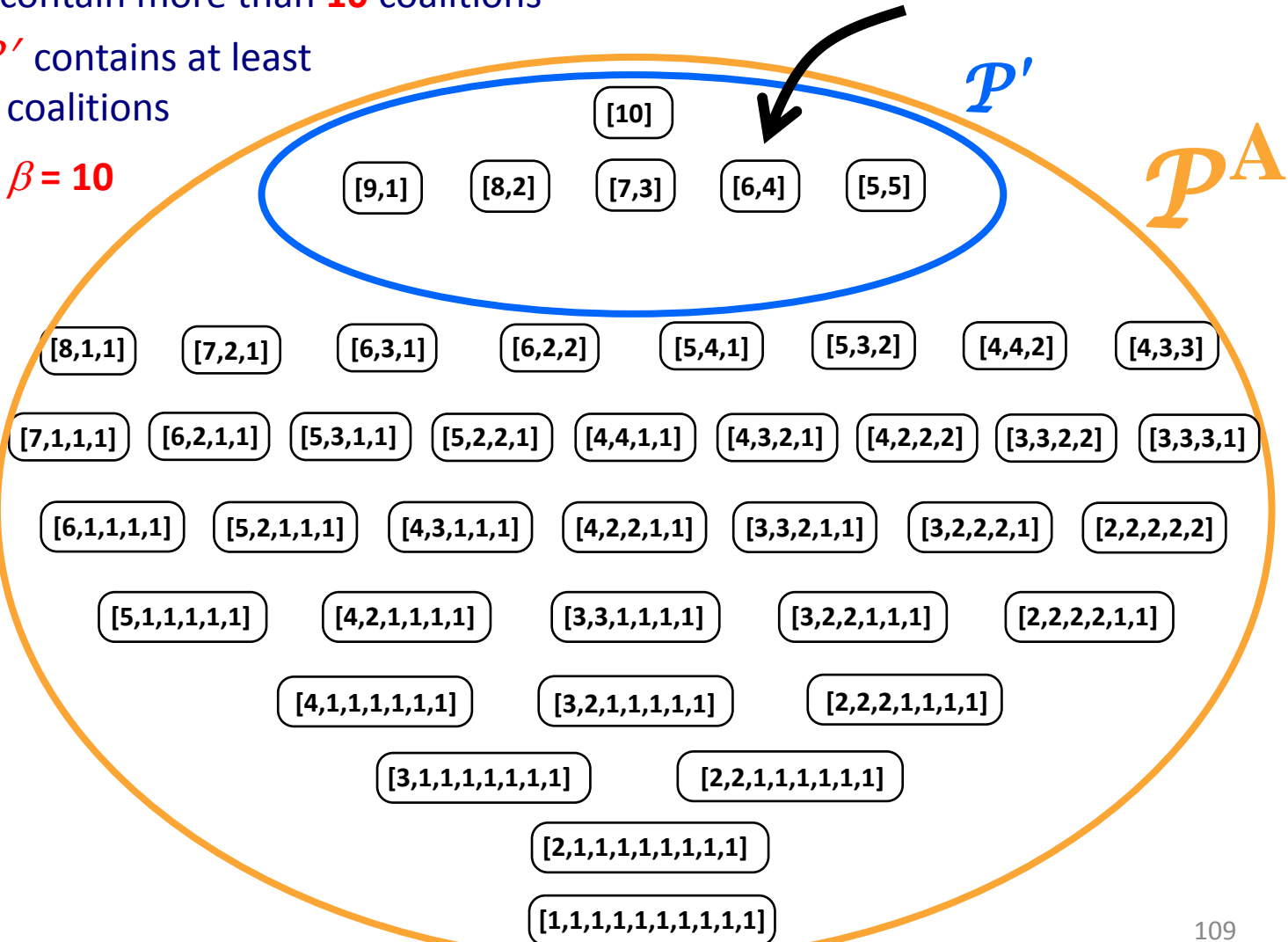
Example of 10 agents:

Space is divided based on integer partitions

- CS^* cannot contain more than **10** coalitions
- every $CS \in \mathcal{P}'$ contains at least one of those coalitions

So, the bound is $\beta = 10$

this is the set that we search



Anytime Algorithms

- So far, we searched:

$$\mathcal{P}_{[10]}^A \cup \mathcal{P}_{[9,1]}^A \cup \mathcal{P}_{[8,2]}^A \cup \mathcal{P}_{[7,3]}^A \cup \mathcal{P}_{[6,4]}^A \cup \mathcal{P}_{[5,5]}^A$$

- Now, what happens if we search $\mathcal{P}_{[1,1,1,1,1,1,1,1,1,1]}^A$, and consider a “group” of coalitions to be one in which the sizes of the coalitions match any of the following?

[1]	[1,1]
[2]	[1,1,1]
[3]	[1,1,1,1]
[4]	[1,1,1,1,1]
[5]	[1,1,1,1,1,1]
[6]	[1,1,1,1,1,1,1]
[7]	[1,1,1,1,1,1,1,1]
[8]	[1,1,1,1,1,1,1,1,1]
[9]	[1,1,1,1,1,1,1,1,1,1]
[10]	

Anytime Algorithms

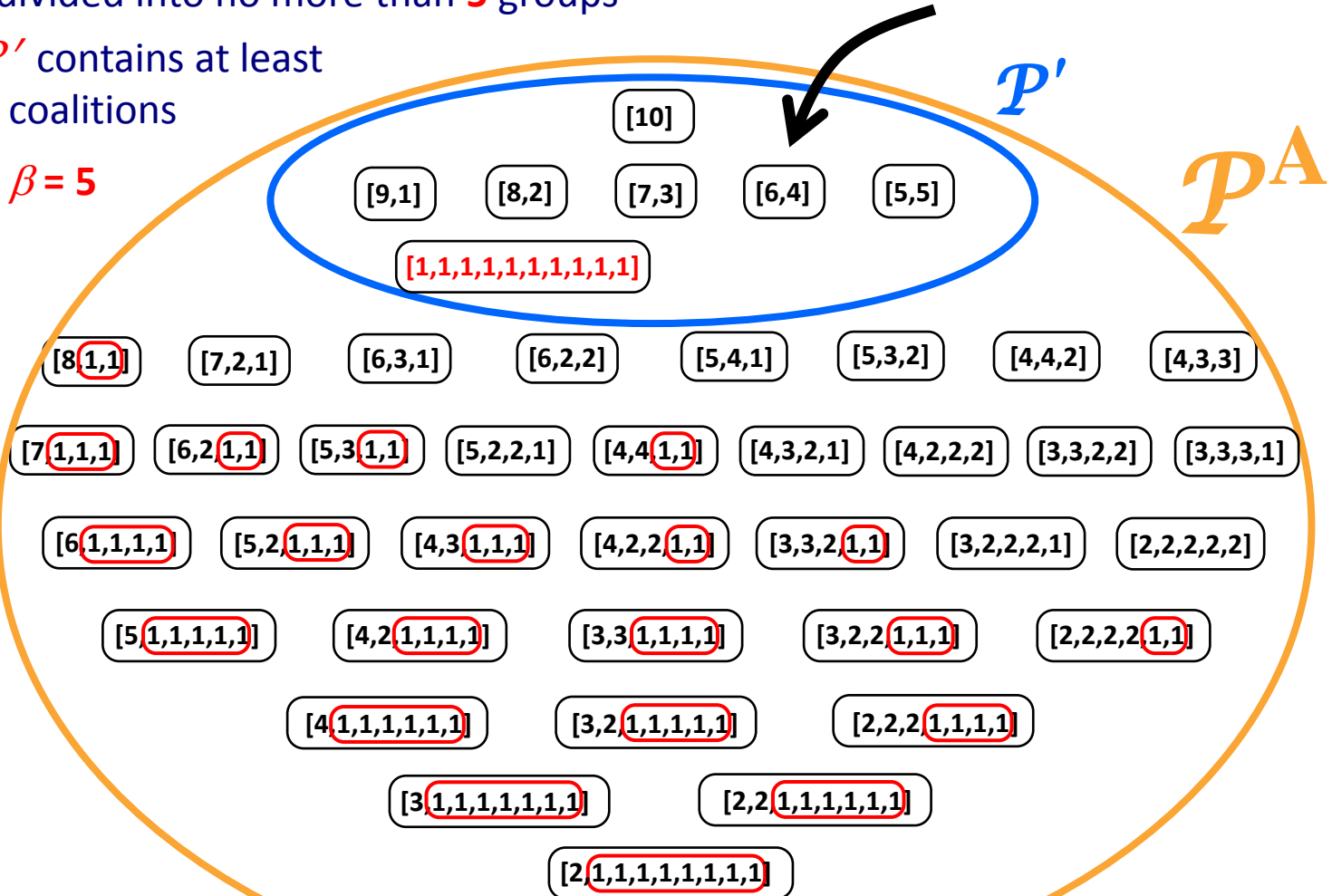
Example of 10 agents:

Space is divided based on integer partitions

- CS^* can be divided into no more than **5** groups
- every $CS \in \mathcal{P}'$ contains at least one of those coalitions

So, the bound is $\beta = 5$

this is the set that we search



Anytime Algorithms

- So far, we searched:

$$\mathcal{P}_{[1,1,1,1,1,1,1,1,1,1]}^A \cup \mathcal{P}_{[10]}^A \cup \mathcal{P}_{[9,1]}^A \cup \mathcal{P}_{[8,2]}^A \cup \mathcal{P}_{[7,3]}^A \cup \mathcal{P}_{[6,4]}^A \cup \mathcal{P}_{[5,5]}^A$$

- Now, what happens if we search $\mathcal{P}_{[2,2,1,1,1,1,1,1]}^A$, and consider a “group” of coalitions to be one in which the sizes of the coalitions match any of the following?

[1]	[1,1]		
[2]	[1,1,1]	[2,1]	
[3]	[1,1,1,1]	[2,1,1]	[2,2]
[4]	[1,1,1,1,1]	[2,1,1,1]	[2,2,1]
[5]	[1,1,1,1,1,1]	[2,1,1,1,1]	[2,2,1,1]
[6]	[1,1,1,1,1,1,1]	[2,1,1,1,1,1]	[2,2,1,1,1]
[7]	[1,1,1,1,1,1,1,1]	[2,1,1,1,1,1,1]	[2,2,1,1,1,1]
[8]	[1,1,1,1,1,1,1,1,1]	[2,1,1,1,1,1,1,1]	[2,2,1,1,1,1,1]
[9]	[1,1,1,1,1,1,1,1,1,1]		
[10]			

Anytime Algorithms

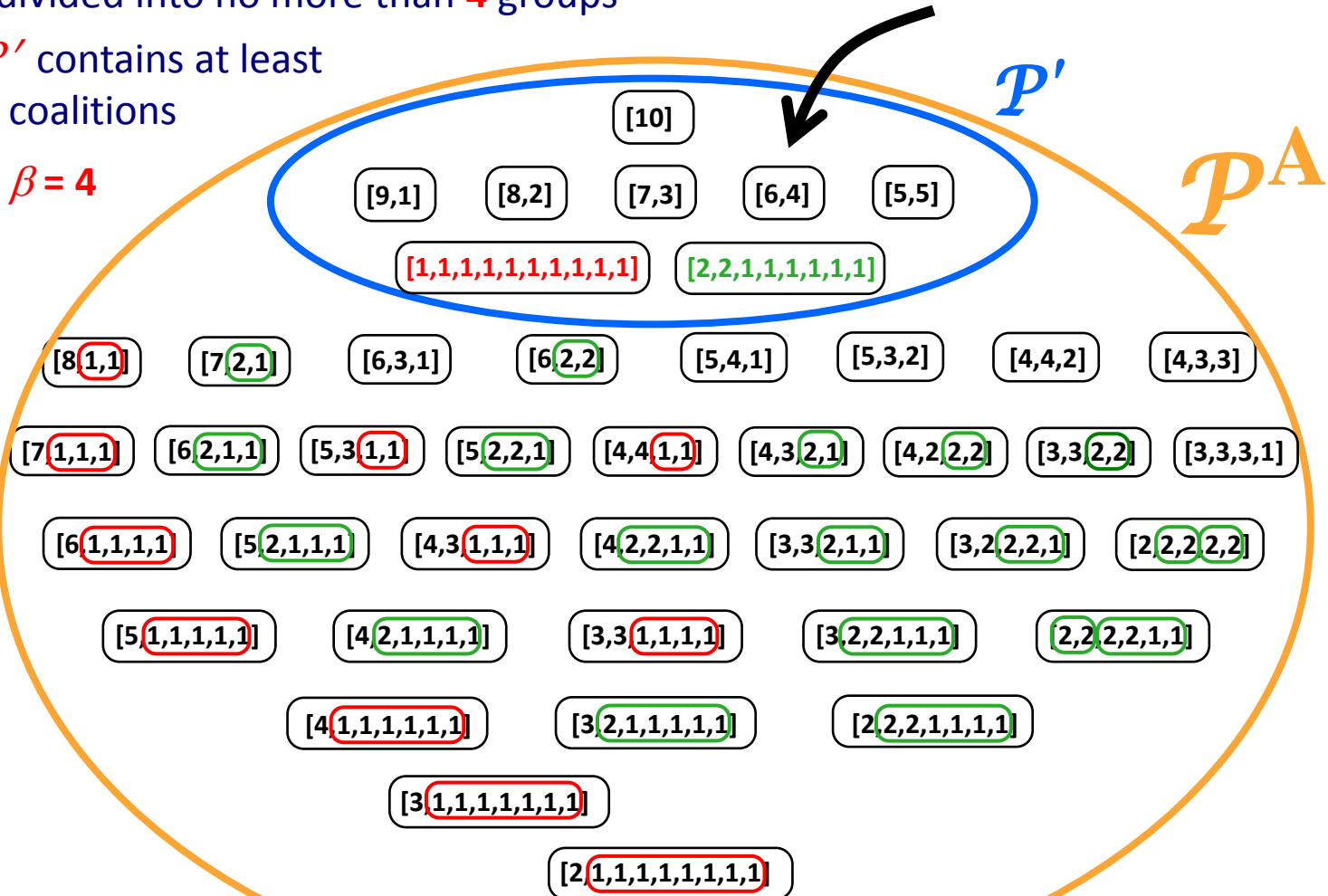
Example of 10 agents:

Space is divided based on integer partitions

- CS^* can be divided into no more than 4 groups
- every $CS \in \mathcal{P}'$ contains at least one of those coalitions

So, the bound is $\beta = 4$

this is the set that we search



Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

So, the idea is:

- To establish a bound $\beta = b$, all we need is find a set of integer partitions, $J' \subseteq J^n$, such that, if we take every possible subset of every $I \in J'$, then with these subsets we can partition every $I \in J^n$ into at most b parts.
- One can optimize this by looking for the set $J' \subseteq J^n$ of integer partitions that corresponds to the smallest number of coalition structures, i.e., the one that minimizes $|\bigcup_{I \in J'} \mathcal{P}_I^A|$
 - This has been shown to be the smallest possible set that must be searched to establish a bound $\beta = b$

Anytime Algorithms

Identifying Subspaces with Worst-Case Guarantees

So far:

- to establish any bound, we showed that the required set can be described in terms of subspaces that are represented by integer partitions
- It would be useful to have an algorithm that can efficiently search those subspaces.
- In the following slides we present an algorithm that does exactly that.

Anytime Algorithms

2. Algorithms based on the integer-partition based representation

Anytime Algorithms

Integer Partition-based Search

- An anytime algorithm was developed based on the **Integer Partition** representation (the algorithm is called **IP**)
- IP uses the observation that, for any $\mathcal{P}_I^A \subseteq \mathcal{P}^A$, we can compute **upper and lower bounds** on the value of the best coalition structure in \mathcal{P}_I^A .
- let Max_s^A and Avg_s^A be the **maximum** and **average** values of all coalitions of size s .

Anytime Algorithms

Integer Partition-based Search

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

$$\frac{\sum_{CS \in \mathcal{P}_I^A} V(CS)}{|\mathcal{P}_I^A|} = \sum_{i \in I} I(i) Avg_i^A$$

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 , i.e., this number is equal for any two coalitions that are of the same size. Denote this number as $\mathcal{N}_I^{|C|}$. Then:

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

To prove this, it suffices to prove:
$$\frac{\sum_{i \in I} \mathcal{N}_I^{|C|} \binom{n}{i} Avg_i^A}{|\mathcal{P}_I^A|} = \sum_{i \in I} I(i) Avg_i^A$$

We will prove this by showing that:
$$\forall i \in I, \mathcal{N}_I^{|C|} \binom{n}{i} = I(i) \cdot |\mathcal{P}_I^A|$$

Anytime Algorithms

Integer Partition-based Search

How to prove that: $\forall i \in I, \mathcal{N}_I^{|\mathcal{C}|} \binom{n}{i} = I(i) \cdot |\mathcal{P}_I^A|$? Observe that every $CS \in \mathcal{P}_I^A$ contains exactly $I(i)$ coalitions of size i . So:

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

We have shown that
$$\sum_{\mathcal{C}:|\mathcal{C}|=i} \mathcal{N}_I^{|\mathcal{C}|} = I(i) \cdot |\mathcal{P}_I^A| \quad (1)$$

On the other hand, since $\mathcal{N}_I^{|\mathcal{C}|}$ is equal for all coalitions of size $|\mathcal{C}|$. then:

$$\sum_{\mathcal{C}:|\mathcal{C}|=i} \mathcal{N}_I^{|\mathcal{C}|} = \mathcal{N}_I^{|\mathcal{C}|} \binom{n}{i} \quad (2)$$

From (1) and (2), we find that: $\mathcal{N}_I^{|\mathcal{C}|} \binom{n}{i} = I(i) \cdot |\mathcal{P}_I^A|$

Anytime Algorithms

Integer Partition-based Search

IP computes bounds on the value of the best coalition structure in \mathcal{P}_I^A :

$$UB_I = \sum_{s \in I} I(s) \cdot Max_s^A \quad LB_I = \sum_{s \in I} I(s) \cdot Avg_s^A$$

and computes bounds on the optimal coalition structure's value, $V(CS^*)$:

$$UB^* = \max_{I \in \mathcal{I}^n} UB_I \quad LB^* = \max_{I \in \mathcal{I}^n} LB_I$$

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:

$$\beta = UB^* / V(CS^*)$$

Computing UB^* allows for identifying subspaces that have no potential of containing an optimal coalition structure, which are:

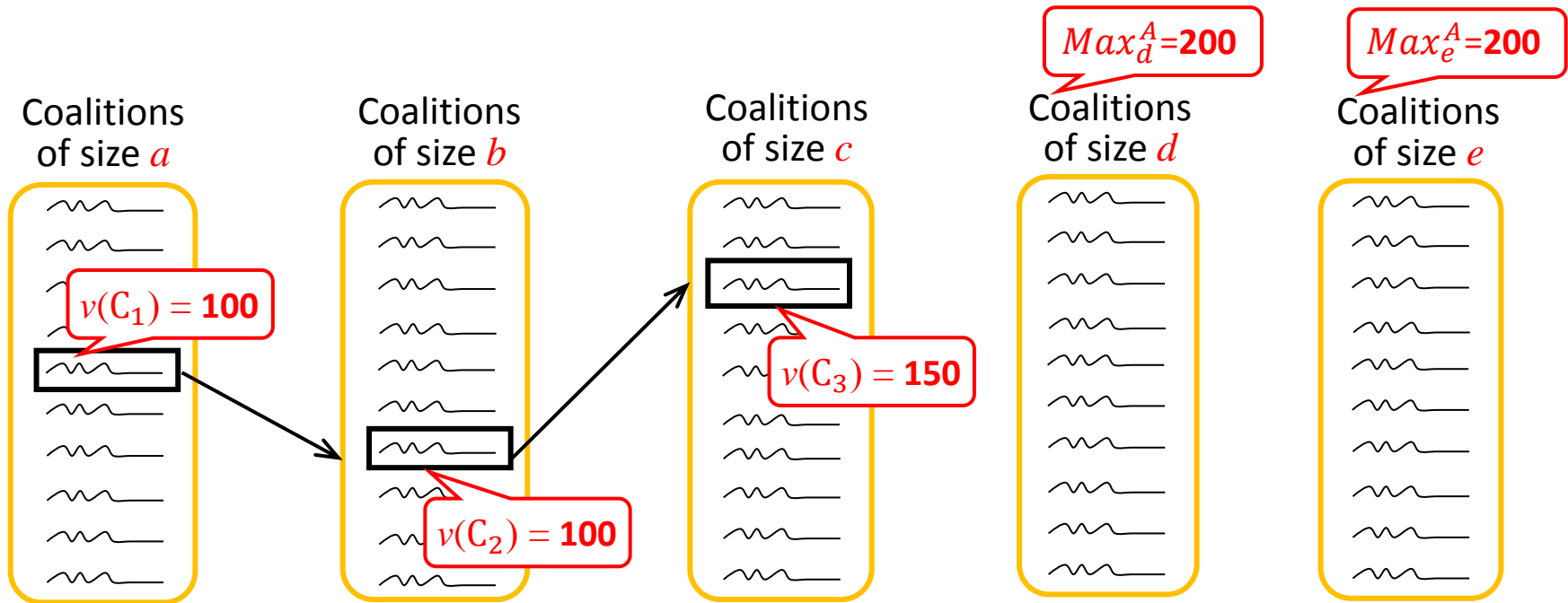
$$\mathcal{P}_I^A: UB_I < LB^*$$

The remaining subspaces are searched using depth-first search combined with branch-and-bound techniques

Anytime Algorithms

Integer Partition-based Search

Example: while searching some subspace, $\mathcal{P}_{[a,b,c,d,e]}^A$, suppose the value of the best coalition structure found so far was **800**. Then, if:



then no need to go deeper in the search tree, because in the very best case we will find a coalition structure of which the value is: **$100+100+150+200+200$** , which is less than **800**

Anytime Algorithms

Integer Partition-based Search

IP runs in $O(n^n)$ time, while IDP runs in $O(3^n)$. However:

- IP is an *anytime* algorithm, unlike IDP
- IP is significantly faster than IDP for coalition-value distributions
- the bound that IP generates, i.e., $\beta = \text{UB}^* / v(\text{CS}^*)$ is significantly better than those obtained by searching particular subsets as before.

An algorithm, called **IDP-IP**, combines IDP with IP. It has the **best of both**:

- it takes from IDP the worst-case complexity, $O(3^n)$
- it takes from IP the ability to run very fast in practice, and the ability to return solutions anytime

Anytime Algorithms

3. Integer programming

Anytime Algorithms

Integer Programming

$$\mathbf{Z} = \begin{matrix} & \begin{matrix} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} & C_{13} & C_{14} & C_{15} \end{matrix} \\ \left(\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right) & \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{matrix} \end{matrix}$$

$\{a_1, a_2, a_4\}$
 $\{a_3\}$

$$\max \sum_{j=1, \dots, 2^n} v(C_j) \cdot x_j$$

$$\text{subject to } \sum_{j=1, \dots, 2^n} z_{i,j} \cdot x_j = 1 \quad \text{for } i = 1, 2, \dots, n$$

$$\text{and } x_j \in \{1, 0\} \quad \text{for } j = 1, 2, \dots, 2^n$$

However, IP has been shown to significantly outperform this approach

Metaheuristic Algorithms

Good scalability, “*good*” solution, no guarantees!

Metaheuristic Algorithms

As the number of agents increases, the problem becomes too hard, and the only practical option is to use metaheuristic algorithms.

Advantage:

- Can usually be applied for very large problems.

Disadvantage:

- No guarantees that an optimal solution is ever found
- No guarantees on the quality of their solutions.

Examples:

- Genetic Algorithms [Sen & Dutta, 2000]
- Simulated Annealing [Keinanen, 2009]
- Decentralized greedy algorithm [Shehory & Kraus, 1998]
- Greedy algorithm based on GRASP [Di Mauro *et al*, 2010]

Compact Representations

Do we really need to explicitly specify the value of every possible coalition?

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

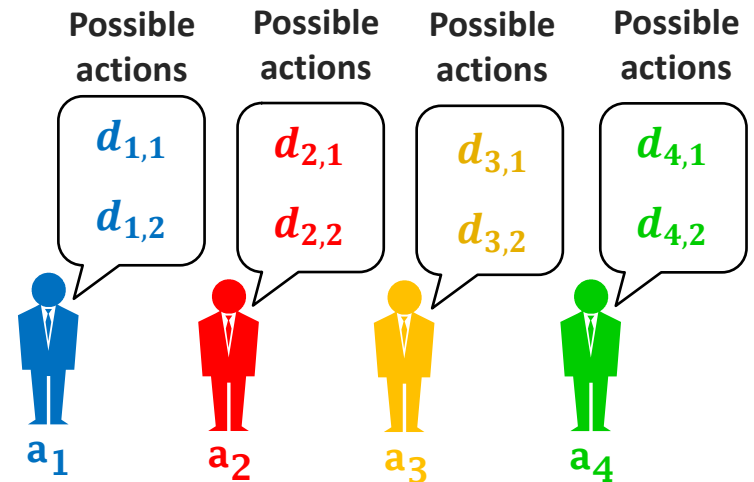
Compact Representations

1. DCOP - Distributed Constraint Optimization

DCOP - Distributed Constraint Optimization

In this framework:

- each agent a_i has a choice of actions:
 $d_{i,1}, d_{i,2}, d_{i,3} \dots$
- a (possibly negative) reward is assigned to every combination of actions
- every agent must choose an action to maximize the sum of rewards.



Ueda *et al.* [2010] studied coalition structure generation where:

- the multi-agent system is represented as one big DCOP
- every coalition's value is computed as the optimal solution of the DCOP among the agents of that coalition.

DCOP - Distributed Constraint Optimization (continued...)

- Solving a DCOP is NP-hard. So, computing the values of all coalitions requires **solving 2^n NP-hard problems!**

- **So, how do we find an optimal coalition structure?**

But who said we must follow the conventional method? which is:

1. compute values of all coalitions, and then
2. find the best combination of disjoint and exhaustive coalitions

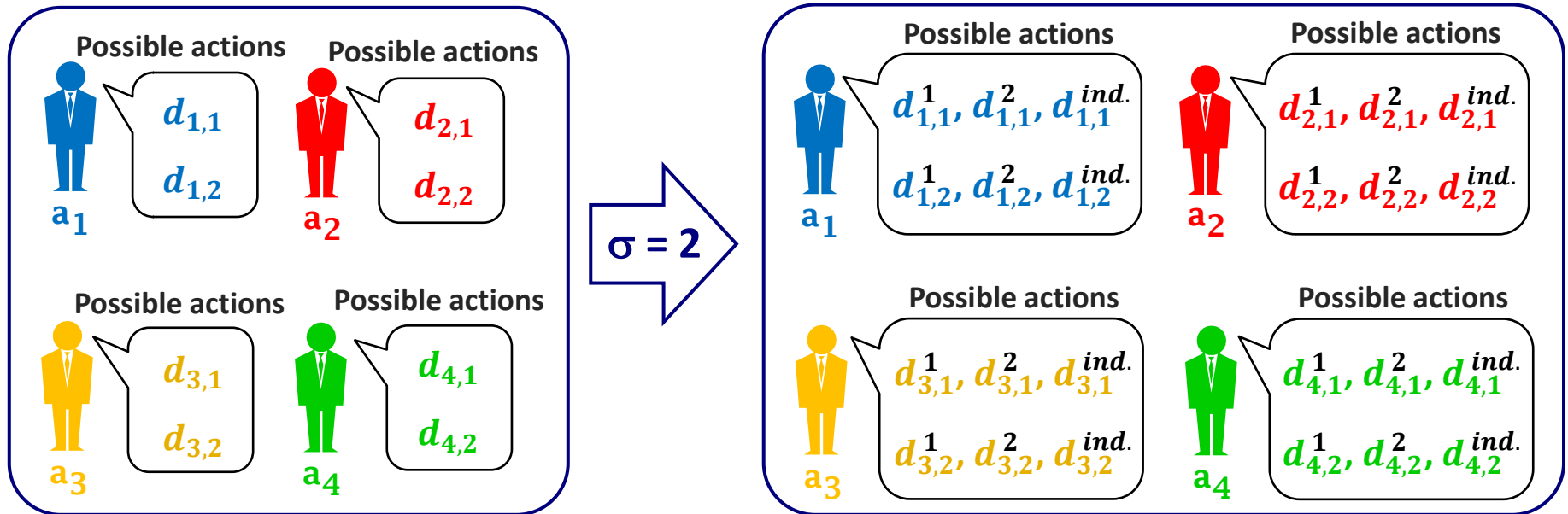
- We can represent the problem differently, so that an optimal coalition structure can be found by solving a single DCOP !
- The modification is controlled by a single parameter, σ , which specifies the max number of permitted “*multi-agent*” coalitions

DCOP - Distributed Constraint Optimization (continued...)

Every action, $d_{i,j}$, is replaced with: $d_{i,j}^1, d_{i,j}^2, \dots, d_{i,j}^\sigma, d_{i,j}^{ind.}$

agent a_i performs action $d_{i,j}$ while joining the 1st multi-agent coalition

agent a_i performs action $d_{i,j}$ independently, i.e., in coalition $\{a_i\}$



To find the optimal coalition structure, solve the new DCOP using any DCOP algorithm, e.g., ADOPT [Modi, 2003] or DPOP [Petcu&Faltings, 2005]

DCOP - Distributed Constraint Optimization (continued...)

Theorem: Assume that $v(C) \geq 0$ for all C . Let CS^* be the optimal coalition structure, and let CS_k^* be the optimal when the number of “multi-agent” coalitions is at most k . The following holds: $\frac{V(CS^*)}{V(CS_k^*)} \leq \frac{\lfloor n/2 \rfloor}{k}$

Proof: Assume that CS^* consists of $l > k$ multi-agent coalitions, C_1, \dots, C_l and assume $v(C_1) \geq \dots \geq v(C_l)$. Then, $v(C_{k+1}) + \dots + v(C_l) \leq \frac{l-k}{l} V(CS^*)$

Now, consider a coalition structure, CS' , which is identical to CS^* , except that every $C_i : i = k+1, \dots, l$ is split into single-agent coalitions. Then, the **maximum difference in value between CS' and CS^* is: $\frac{l-k}{l} V(CS^*)$** . i.e.,

$$\frac{V(CS^*)}{V(CS')} \leq \frac{l}{k}$$

Finally, observe that $l \leq \lfloor n/2 \rfloor$ (because CS^* cannot possibly contain more than $\lfloor n/2 \rfloor$ multi-agent coalitions).

Compact Representations

2. Marginal Contribution Nets

Marginal Contribution Nets (MC-nets)

Definition:

With MC-nets, a game is represented by a set of rules \mathcal{R} :

- Each rule $r \in \mathcal{R}$ is of the form: $B_r \rightarrow \vartheta_r$
 - ϑ_r is a real value.
 - B_r is a Boolean formula over a set of variables $\{b_1, \dots, b_n\}$
- A rule $r \in \mathcal{R}$ is *applicable* to coalition C if B_r is satisfied by the following assignment:
 - $b_i = \text{true}$ if $a_i \in C$, and $b_i = \text{false}$ if $a_i \notin C$.
- The value of a coalition C is the sum of values of all rules applicable to C

Example:

The rules $\mathcal{R} = \{b_1 \wedge b_2 \rightarrow 5, b_2 \rightarrow 2\}$ represent the game $G=(A,v)$, where:

- $A=\{a_1, a_2\}$,
- $v(\{a_1\})=0$
- $v(\{a_2\})=2$
- $v(\{a_1, a_2\})=7$

Marginal Contribution Nets (continued...)

An MC-net is said to be basic if the left-hand side of any rule is a conjunction of variables or their negations.

In this case, we write a rule $r \in \mathcal{R}$ as $(P_r, N_r) \rightarrow \mathfrak{g}_r$ where:

- P_r the set of positive literals
- N_r the set of negative literals.

Any game $G=(A,v)$ can be represented by a basic MC-net with *at most* 2^n-1 rules:

- for each coalition C , create the rule:

$$(\bigwedge_{i:ai \in C} b_i) \wedge (\bigwedge_{i:ai \notin C} \neg b_i) \rightarrow v(C)$$

However, many interesting games require much fewer rules

Marginal Contribution Nets (continued...)

Coalition structure generation was studied with a restricted class of basic MC-nets, where $P_r \neq \emptyset$ and $\vartheta_r > 0$ for every r .

Definition:

a set of rules $\mathcal{R}' \subseteq \mathcal{R}$ is *feasible* if all rules in \mathcal{R}' are applicable *at the same time* to some coalition structure CS (i.e., every $r \in \mathcal{R}'$ is applicable to some $C \in CS$).

Based on this: in MC-nets:

coalition structure generation = find a feasible set \mathcal{R}' that maximizes $\sum_{r \in \mathcal{R}'} \vartheta_r$

Solution:

Use mixed integer programming (MIP)

In the following slides, we explain the MIP formulation

Marginal Contribution Nets (continued...)

The MIP formulation is based on the observation that, for any two rules r, r' the possible relations between r and r' can be classified into 4 classes:

1. **Incompatible (IC)**: This is when $P_r \cap P_{r'} \neq \emptyset$ and $(P_r \cap N_{r'} \neq \emptyset$ or $P_{r'} \cap N_r \neq \emptyset)$.

Example: $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_2, a_3\}, \{a_1\}) \rightarrow \vartheta_2$ are not applicable at the same time, because:

- the 1st rule requires a_1 and a_2 to appear together in a coalition,
- the 2nd rule requires a_2 and a_3 to appear together in a coalition that does not contain a_1

2. **Compatible on same coalition (CS)**: when $P_r \cap P_{r'} \neq \emptyset$ and $P_r \cap N_{r'} = P_{r'} \cap N_r = \emptyset$

Example: $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_2, a_3\}, \{a_4\}) \rightarrow \vartheta_2$ are applicable *at the same time* in any coalition structure CS that contains a coalition C , where that $\{a_1, a_2, a_3\} \subseteq C$ and $a_4 \notin C$. **Note**: *both rules apply to the same coalition.*

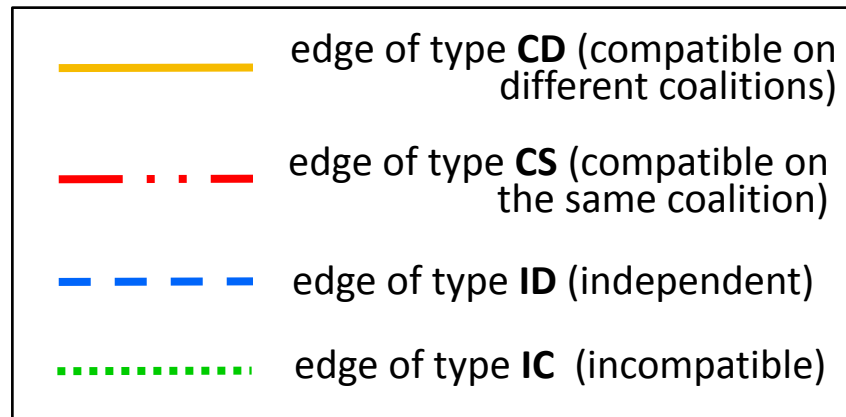
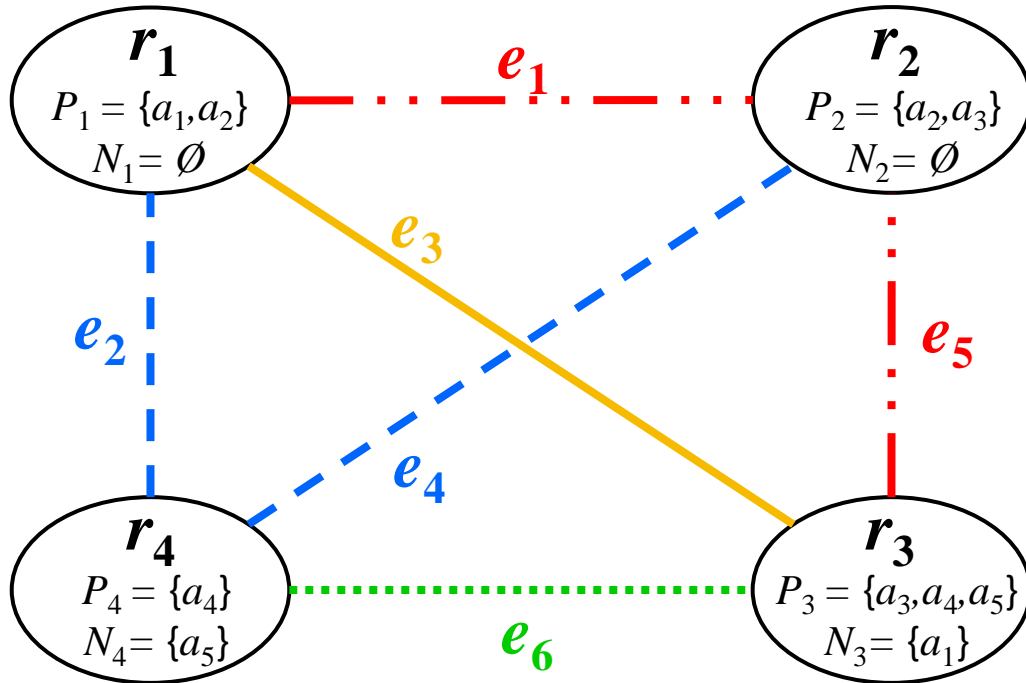
3. **Compatible on different coalitions (CD)**: $P_r \cap P_{r'} = \emptyset$ and $(P_r \cap N_{r'} \neq \emptyset$ or $P_{r'} \cap N_r \neq \emptyset)$

Example: $(\{a_1, a_2\}, \emptyset) \rightarrow \vartheta_1$ and $(\{a_3, a_4\}, \{a_1\}) \rightarrow \vartheta_2$ are applicable at the same time in some CS as long as a_1, a_2 appear in a coalition, and a_3, a_4 appear in a different one.

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

Marginal Contribution Nets (continued...)

Graphical representation (4 types of edges, representing the 4 types of relations)



Marginal Contribution Nets (continued...)

Theorem:

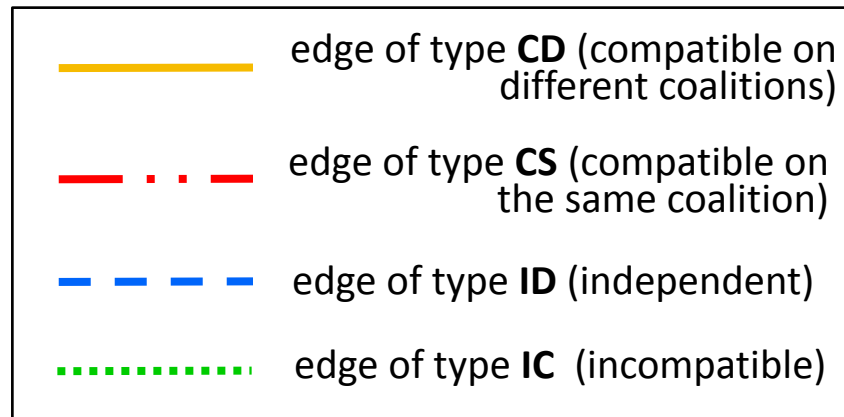
A set of rules \mathcal{R}' is feasible if and only if:

1. no pair $r_1, r_2 \in \mathcal{R}'$ are connected by an edge of type **IC**,
2. for any $r_1, r_2 \in \mathcal{R}'$, if r_1 can be reached from r_2 via series of edges of type **CS**, then the edge (r_1, r_2) must NOT be of type **CD**

The 2nd condition can be **generalize** as follows: Any set of rules:



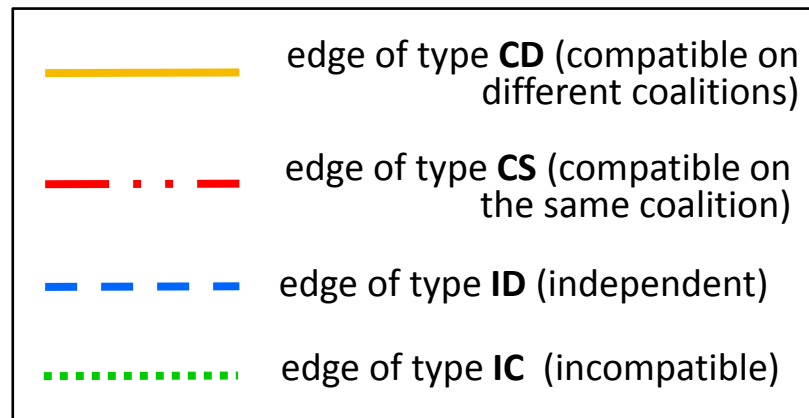
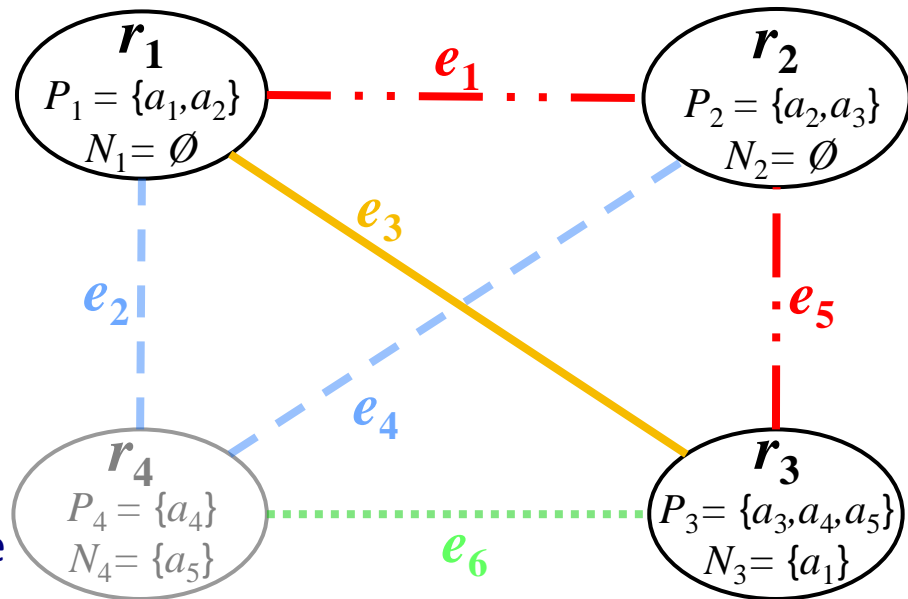
must all be applicable to a single coalition (which contains $P_1 \cup P_2 \cup \dots \cup P_m$)



Marginal Contribution Nets (continued...)

Intuition: consider an example (see figure):

- An edge of type **CS** connects r_1 to r_2 . Thus, they must be applicable to a single coalition in **CS**, say C' , such that $P_1 \cup P_2 \subseteq C'$.
- An edge of type **CS** connects r_2 to r_3 . Thus, they must be applicable to a single coalition in **CS**, say C'' , such that $P_2 \cup P_3 \subseteq C''$.
- 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 . Thus, any rule set containing r_1, r_2, r_3 is not feasible.



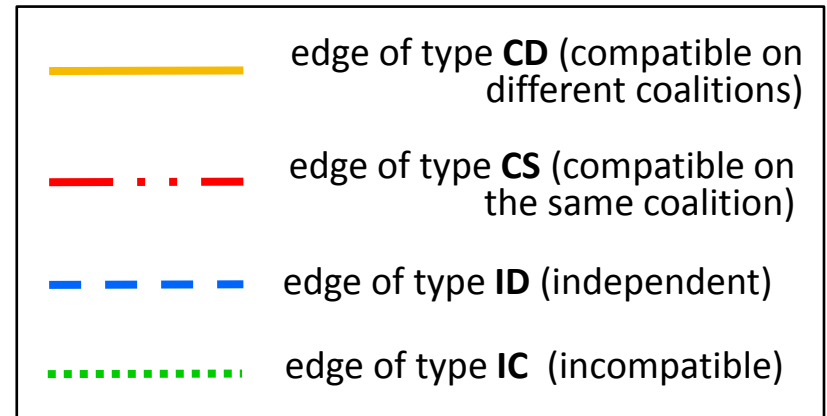
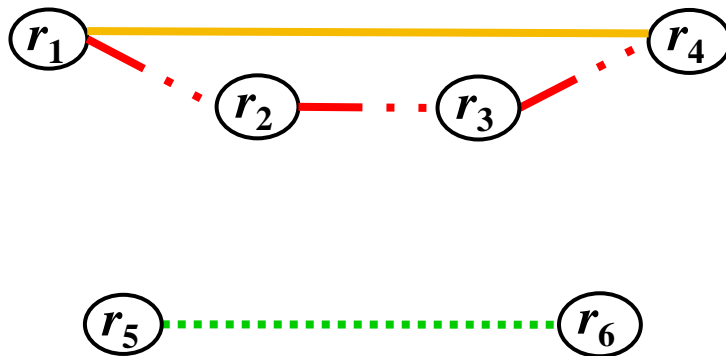
Marginal Contribution Nets (continued...)

As we said earlier:

With MC-nets, finding an optimal coalition structure generation is equivalent to finding a feasible set \mathcal{R}' that maximizes $\sum_{r \in \mathcal{R}'} \vartheta_r$

The mixed Integer program:

The main objective is to find a set of rules that maximize $\sum_{r \in \mathcal{R}'} \vartheta_r$ while avoiding the following two cases:



Marginal Contribution Nets (continued...)

Details of mixed integer program:

- For every rule r , define a binary variable x_r
- For every edge e of type **CD**, and for every rule r such that:

- r is an endpoint of e , or
- r is an endpoint of some other edge of type **CS**

define a variable y_r^e

Example from the figure:

Since e_3 is of type **CD**, we define variables:

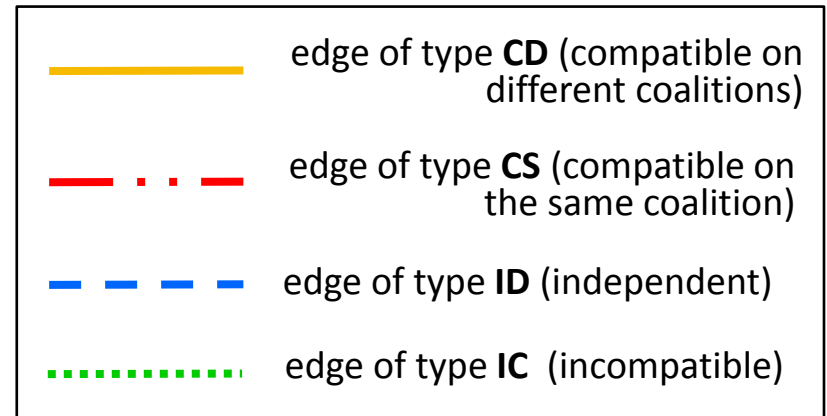
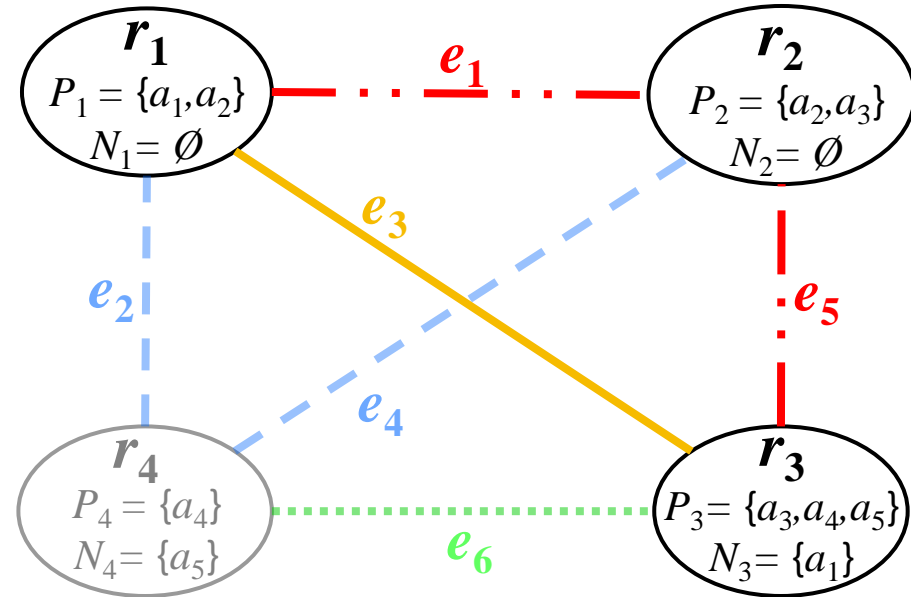
$y_{r_1}^{e_3}, y_{r_2}^{e_3}, y_{r_3}^{e_3}$ (because every rule in $\{r_1, r_2, r_3\}$ is either an end point of e_3 , or an end point of some edge of type **CS**)

In this example, the constraints in the mixed integer program correspond to:

$$y_{r_1}^{e_3} \neq y_{r_3}^{e_3}$$

$$y_{r_1}^{e_3} = y_{r_2}^{e_3} \text{ (only if } x_1=1 \text{ and } x_2=1)$$

$$y_{r_2}^{e_3} = y_{r_3}^{e_3} \text{ (only if } x_2=1 \text{ and } x_3=1)$$



Compact Representations

3. Coalitional Skill Games

Coalitional Skill Games

In many settings, the value of a coalition can be defined in terms of the skills that are possessed by the agents.

How do we represent this?

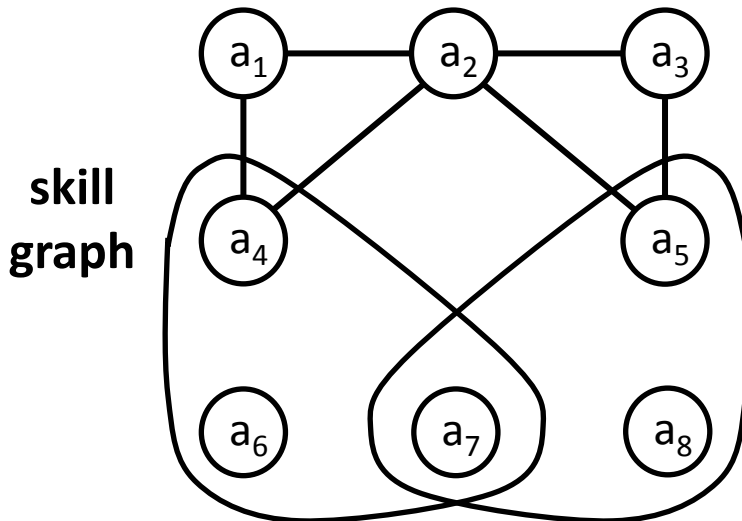
- Set of skills, S
- a set of tasks, Γ
- every task $\tau \in \Gamma$ has a skill requirement, $S^\tau \subseteq S$, and a payoff
- each agent $a_i \in A$ has a set of skills $S^{a_i} \subseteq S$
- A coalition $C \subseteq A$ *achieves a* task τ if it has all skills required for τ
- there is a *task value function*, $F : 2^\Gamma \rightarrow \mathbb{R}$. For every subset, $\Gamma' \subseteq \Gamma$, it specifies the payoff from achieving all tasks in Γ'
- The value of a coalition C is: $v(C) = F(\{ \tau : S^\tau \subseteq \bigcup_{a_i \in C} S^{a_i} \})$

Coalitional Skill Games (continued...)

How do we find an optimal coalition structure?

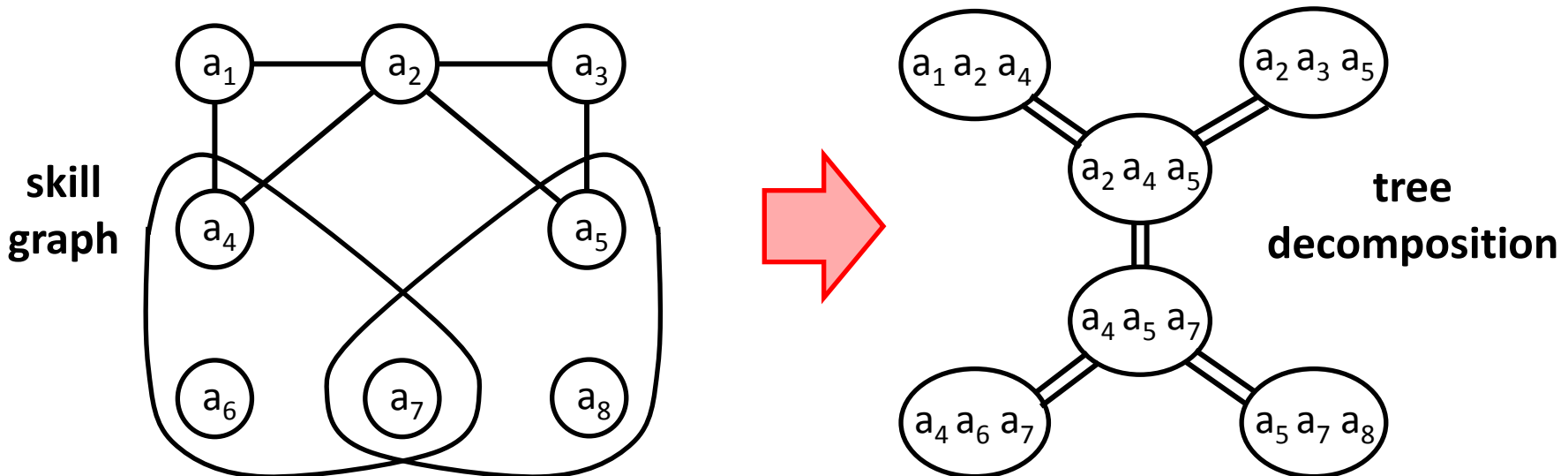
To do this, we need more definitions:

- Given a skill set, \mathcal{S} , its *skill graph* is a hypergraph, $g = \langle V, E \rangle$:
 - every agent is represented as a vertex,
 - every skill, $s_j \in \mathcal{S}$ is represented as a hyperedge $e_{s_i} \in E$ connecting agents that possess this skill.



Coalitional Skill Games (continued...)

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



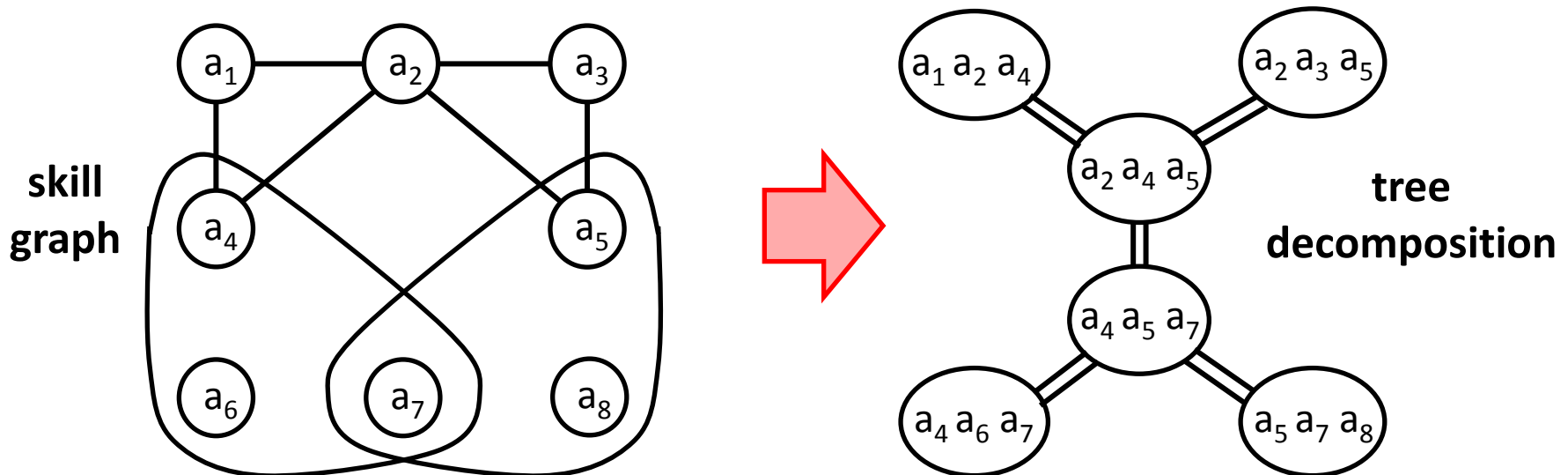
Coalitional Skill Games (continued...)

Main idea:

- In general, **constraint satisfaction problems** can be solved in polynomial time if the graph is a tree.

Based on this:

- we will show how to represent the coalition structure generation problem as solving **multiple constraint satisfaction problems on the skill graph**
- For each such problem, we will show how to convert it to a **constraint satisfaction problem on the tree decomposition**



Coalitional Skill Games (continued...)

Note: 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).

Based on this, if m is the largest number of tasks, and d is the largest number of agents sharing a single skill, then a coalition structure can accomplish at most dm tasks.

We can then define a candidate task solution as a set $\{\Gamma_i\}_{i=1}^h$, where each Γ_i is a subset of Γ , and $h \leq dm$.

Based on this: in coalitional skill games:

coalition structure generation = find a candidate task solution that maximizes $\sum_{i=1}^h F(\Gamma_i)$ and can be accomplished by some coalition structure.

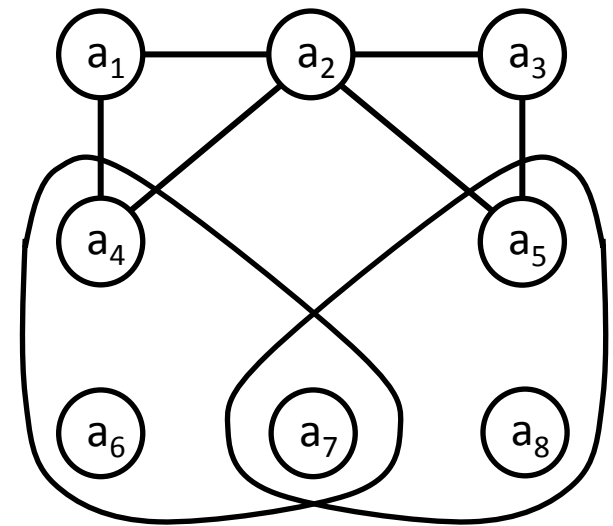
Algorithm: iterate over all possible choices of $\{\Gamma_i\}_{i=1}^h$. For each such choice: *find the coalition structure that accomplishes it, or determine that it is not feasible*. Next, we show how this is done.

Coalitional Skill Games (continued...)

every coalition structure can be viewed as a **coloring** of agents (agents with the same color belong to the same coalition).

Based on this, define a **constraint satisfaction problem 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), which are h in total;
- For each skill s , we have the following **constraint**: *For $i = 1, \dots, h$, if some task in Γ_i requires s , then at least one agent in C_i has s .*

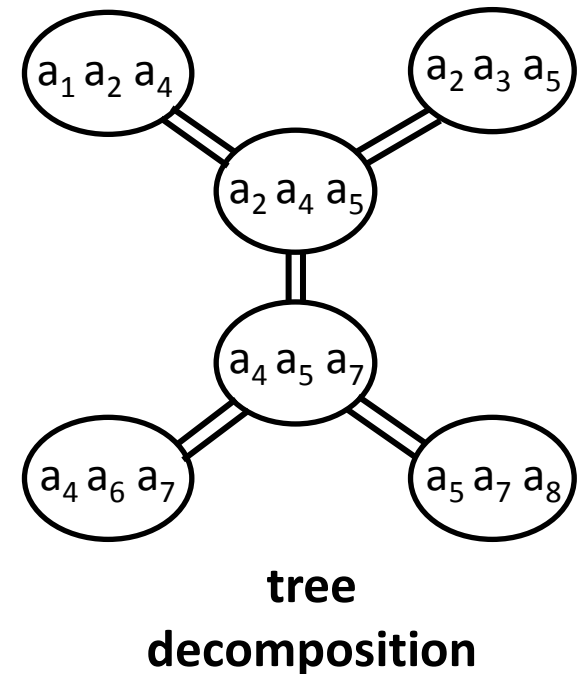


skill
graph

Coalitional Skill Games (continued...)

To solve this problem, we define another constraint satisfaction problem whose underlying graph is the tree decomposition of g , where:

- 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 **constraints** are of two types.
 - The 1st 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.
 - The 2nd type is the same as before:
For $i = 1, \dots, h$, if some task in Γ_i requires s , then at least one agent in C_i has s .

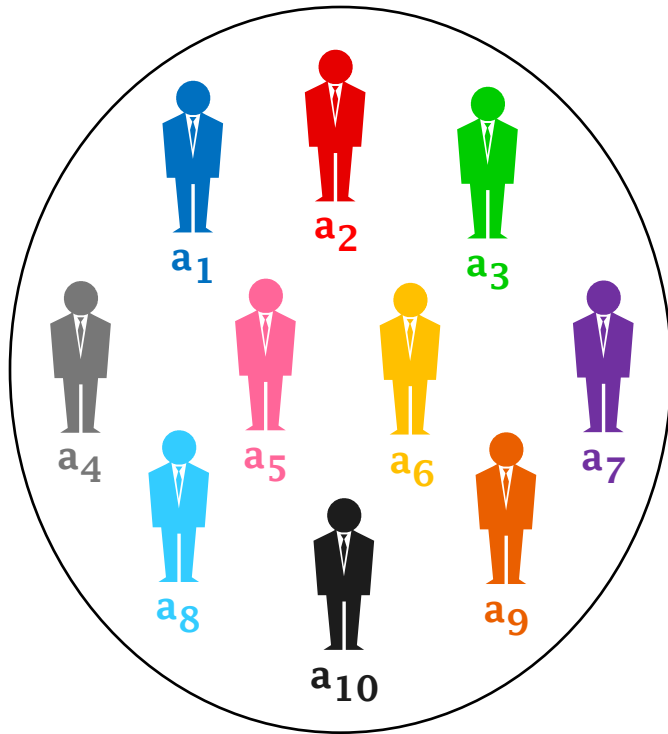


Compact Representations

4. Agent-type Representation

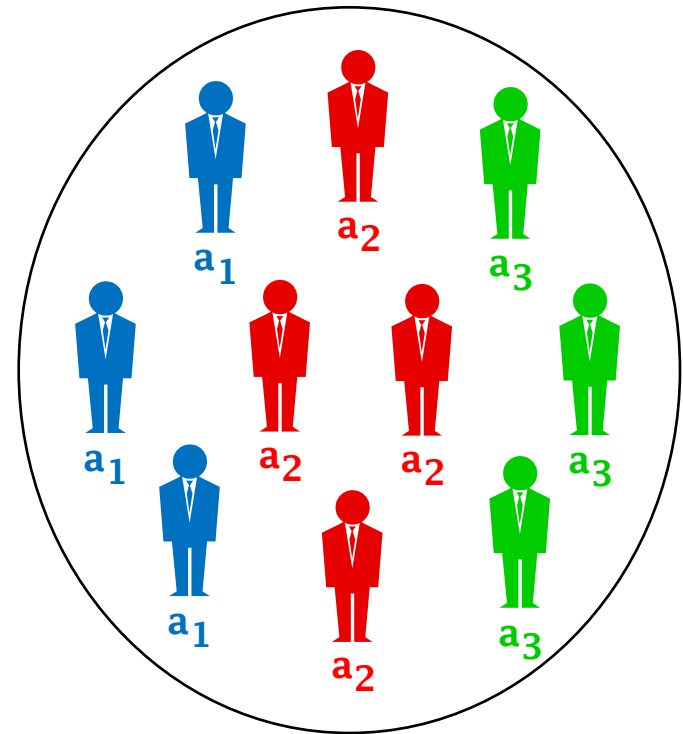
Agent-type Representation

In the general case: each agent is considered to be unique.



10 agents

In the agent-type representation, some agents are identical, i.e., of the same “type”




10 agents with 3 types

Agent-type Representation (continued...)

Formally:

The set of agents is partitioned into subsets A^1, \dots, A^T , each subset is called a **type**:

the number
of types



- for any type, A^i , every two agents in A^i make the same contribution to any coalition that they join, i.e.,

$$\forall a_j, a_k \in A^i, \forall C: a_j, a_k \notin C, v(C \cup \{a_j\}) = v(C \cup \{a_k\})$$

Agent-type Representation (continued...)

A **coalition** is defined by the *number of agents* of each type that it contains.

In the general case, we deal with “coalitions”, e.g.,

$$C = \{ a_1, a_4, a_5, a_7, a_8 \}$$

In the agent-type representation, we deal with “coalition-types”, e.g.

$$\psi = \langle n^1, \dots, n^T \rangle$$

the number of coalition members that are of type A^1

Note: the number of possible “*coalition types*” is $O(|A|^T)$., because:

$$(|A^1|+1) \times \dots \times (|A^T|+1) < |A|^T$$

So, instead of defining a value $v(C)$ for every “*coalition*” (which are $2^{|A|}$ in total), we define a value $v^t(\psi)$ for every “*coalition type*” (which are $O(|A|^T)$ in total).

This number is **polynomial** in the number of agents

Agent-type Representation (continued...)

Given a *coalition-type*, $\psi = \langle n^1, \dots, n^T \rangle$, a partition of the agents in ψ is called a *type-partition*.

Example:

Given a coalition type $\psi = \langle 4, 4, 4 \rangle$, one possible **type-partition** of ψ is:

$$\{\langle 0, 1, 2 \rangle, \langle 4, 3, 2 \rangle\}$$

the value of this type-partition is:

$$V^t(\{\langle 0, 1, 2 \rangle, \langle 4, 3, 2 \rangle\}) = v^t(\langle 0, 1, 2 \rangle) + v^t(\langle 4, 3, 2 \rangle)$$

Based on this: In the agent-type representation:

coalition structure generation = find a *type-partition* of \mathbf{A} that maximizes V^t

Agent-type Representation (continued...)

Finding a *type-partition* of \mathbf{A} that maximizes \mathbf{V}^t can be done using **dynamic programming**. This is based on the following:

- Let $f^t(\psi)$ be the value of the optimal type-partition of ψ .
- We can compute $f^t(\psi)$ recursively as follows:

$$f^t(\psi) = \begin{cases} 0 & \text{if } n_i = 0 \text{ for } i=1, \dots, T \\ \max \{ f^t(\langle n^1 - x^1, \dots, n^T - x^T \rangle) + v^t(\langle x^1, \dots, x^T \rangle) : \\ \quad x_i \leq n_i \text{ for } I = 1, \dots, T \} & \text{otherwise} \end{cases}$$

The dynamic programming algorithm computes $f^t(\mathbf{A})$ recursively using the above formula.

Constrained Coalition Formation

So far, we assumed that agents can split into subsets in any way they like!

Constrained Coalition Formation (CCF)

■ In conventional models:

- every possible subset of agents is a potential *coalition*,
- every possible partition of the set of agents is a potential *coalition structure*.

■ Many times we have constraints that enforce or prohibit the co-existence of certain agents in a coalition.

■ To date, very limited work on constraints, e.g., only permitting certain sizes.

General CCF Model

- Let $\Pi(A)$ be the set of all coalition structures. A constrained coalition formation (CCF) game is a tuple $G = \langle A, CS, v \rangle$ where:
 - $A = \{a_1, \dots, a_n\}$ is the set of agents
 - $CS \subseteq \Pi(A)$ is the set of coalition structures that are **feasible**
 - $v : (\bigcup_{CS \in CS} \bigcup_{C \in CS} \{C\}) \rightarrow \mathbb{R}$ assigns a value to every coalition that appears in some feasible coalition structure
- Feasibility is defined for coalition structures rather than for coalitions:
 - **Example:** a coalition structure is feasible only if it contains coalitions of **equal sizes**.
 - In this example, $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ is not feasible, even though each of its coalitions may be a part of some feasible coalition structure

Locally Constrained CCF Games

A CCF game $\mathcal{G} = \langle A, \mathcal{CS}, v \rangle$ is *locally constrained* if there exists a set of coalitions, $C \subseteq 2^A$, such that $\mathcal{CS} = \{CS \in \Pi(A) : CS \subseteq C\}$.

- We will refer to the coalitions in C as **feasible coalitions**.

■ Constraints are represented using propositional logic:

- Boolean variables correspond to agents: $B_A = \{b_i : a_i \in A\}$
- Let δ be a propositional formula over B_A constructed using the classical connectives ($\wedge, \vee, \neg, \rightarrow, \dots$).
- Coalition C satisfies δ iff δ is satisfied under the truth assignment that sets all $b_i : i \in C$ to **true**, and all $b_i : i \notin C$ to **false**.
- Example: $C = \{a_1, a_4, a_5, a_7\}$ satisfies $\delta = b_4 \wedge b_5$

Proposition: The class of *propositionally definable CCF* games is equal to the class of *locally constrained CCF* games.

Basic CCF Games

- A *basic CCF game* is a tuple $G = \langle A, \mathcal{P}, \mathcal{N}, S, v \rangle$ where:
 - \mathcal{P} is a set of subsets of A (*Positive constraints*)
 - \mathcal{N} is a set of subsets of A (*Negative constraints*)
 - $S \subseteq \mathbb{N}$ (permitted sizes)
- A coalition C is feasible if:
 - $P \subseteq C$ for some $P \in \mathcal{P}$
 - $N \not\subseteq C$ for all $N \in \mathcal{N}$
 - $|C| \in S$
- We will denote the feasible coalitions as $c(A, \mathcal{P}, \mathcal{N}, S)$

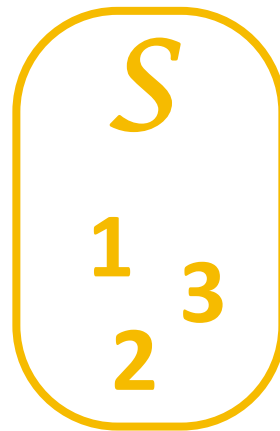
Basic CCF Games

Example: We are organizing a conference, and we need to determine from which restaurant(s) we are going to order the food.



Basic CCF Games

- \mathcal{N} : we don't want 2 restaurants together providing chilli food, or together providing vegetarian food.
- \mathcal{S} : we do not want to coordinate between more than 3 restaurants.
- \mathcal{P} : any combination of restaurants must either contain Nandos, Zizzi, or the two Japanese restaurants that, together, provide a nice menu.



General CCF games

Constraints are placed **on Coalition Structures**

Local CCF games

Constraints are placed **on Coalitions**

- Constraint can be expressed using **propositional logic**

Basic CCF games

Constraints are placed **on Coalitions** In the form of:

- **Size constraints**
- **Positive constraints**
- **Negative constraints**

How to Generate Feasible Coalitions?

Given a set of constraints, how do we generate the feasible coalitions?

$\mathcal{P} = \{\emptyset\}$, $\mathcal{N} = \{\{1,2,3,4\}, \{1,2,4,5\}, \{5,6\}\}$



size=1	size = 2	size = 3	size = 4	size = 5	size = 6
1	1, 2	1, 2, 3	1, 2, 3, 4	1, 2, 3, 4, 5	1, 2, 3, 4, 5, 6
2	1, 3	1, 2, 4	1, 2, 3, 5	1, 2, 3, 4, 6	
3	1, 4	1, 2, 5	1, 2, 3, 6	1, 2, 3, 5, 6	
4	1, 5	1, 2, 6	1, 2, 4, 5	1, 2, 4, 5, 6	
5	1, 6	1, 3, 4	1, 2, 4, 6	1, 3, 4, 5, 6	
6	2, 3	1, 3, 5	1, 2, 5, 6	2, 3, 4, 5, 6	
	2, 4	1, 3, 6	1, 3, 4, 5		
	2, 5	1, 4, 5	1, 3, 4, 6		
	2, 6	1, 4, 6	1, 3, 5, 6		
	3, 4	1, 5, 6	1, 4, 5, 6		
	3, 5	2, 3, 4	2, 3, 4, 5		
	3, 6	2, 3, 5	2, 3, 4, 6		
	4, 5	2, 3, 6	2, 3, 5, 6		
	4, 6	2, 4, 5	2, 4, 5, 6		
	5, 6	2, 4, 6	3, 4, 5, 6		
		2, 5, 6			
		3, 4, 5			
		3, 4, 6			
		3, 5, 6			
		4, 5, 6			

Divide & Conquer

For any $i \in A$, we divide the feasible coalitions $c(A, \mathcal{P}, \mathcal{N}, S)$ into:

■ Coalitions that **do not contain** a_i : For those, we can:

- Remove a_i from A
- Remove every $P \in \mathcal{P}$ such that $a_i \in P$
- Remove every $N \in \mathcal{N}$ such that $a_i \in N$

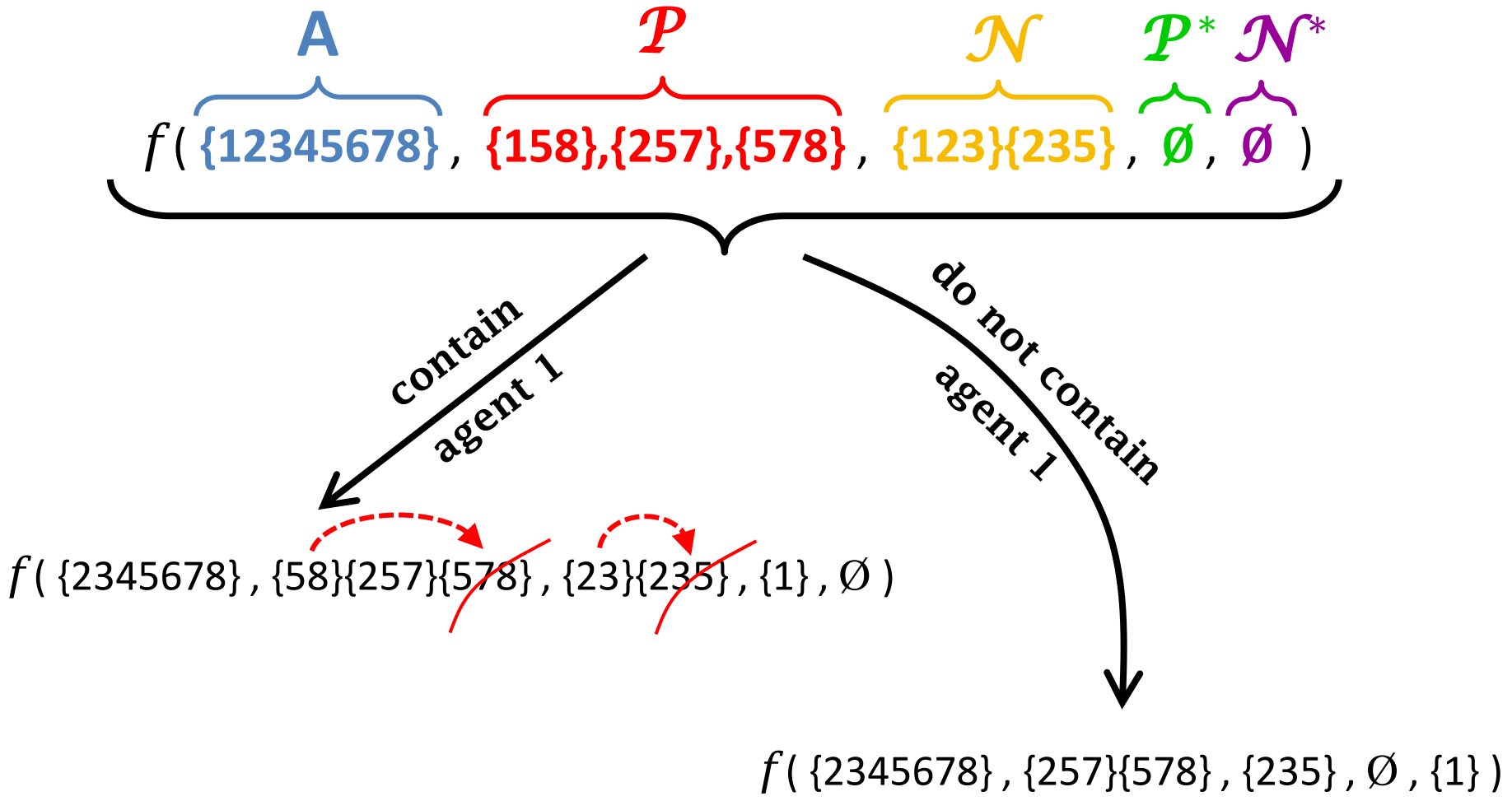
■ Coalitions that **contain** a_i : For those, we can:

- Remove a_i from A
- Remove a_i from every $P \in \mathcal{P}$ such that $a_i \in P$
- Remove a_i from every $N \in \mathcal{N}$ such that $a_i \in N$

Repeat the above recursively until we reach:

- an **impossible case**, where: $\mathcal{P} = \emptyset$ or $\mathcal{N} \ni \emptyset$
- or a **base case**: $|\mathcal{P}| = 1$, $\cap \mathcal{N} = \emptyset$, $|\{N \in \mathcal{N} : |N| > 1\}| \leq 1$

Divide & Conquer



Divide & Conquer

Repeat the division recursively until we reach:

- an **impossible case**, where:

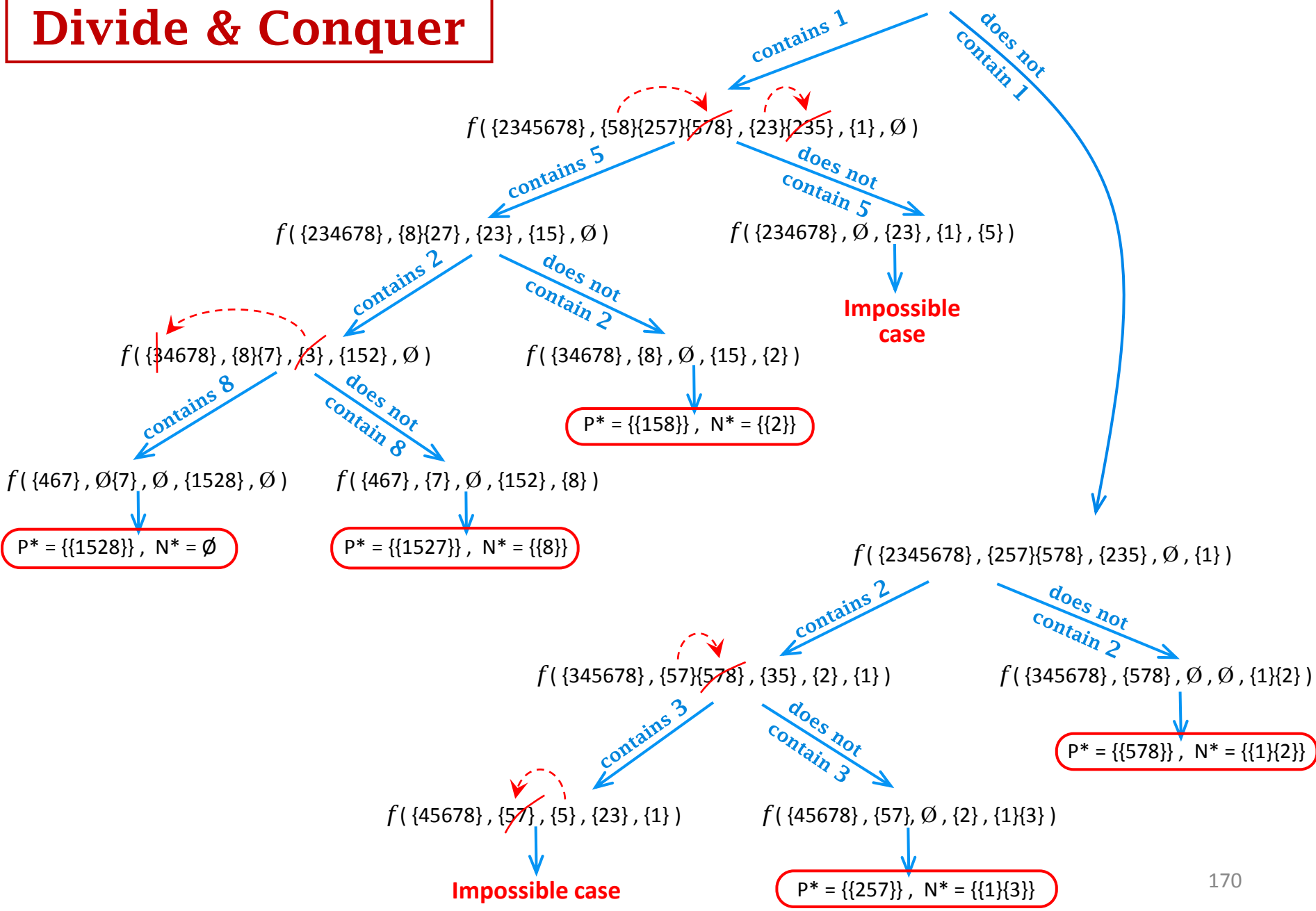
$$\mathcal{P} = \emptyset \quad \text{or} \quad \mathcal{N} \ni \emptyset$$

- or a **base case**, where:

$$|\mathcal{P}| = 1 \quad \text{and} \quad \cap \mathcal{N} = \emptyset \quad \text{and} \quad |\{N \in \mathcal{N} : |N| > 1\}| \leq 1$$

$$f(A, P, N, P^*, N^*) = f(\{12345678\}, \{158\}\{257\}\{578\}, \{123\}\{235\}, \emptyset, \emptyset)$$

Divide & Conquer



Back to our Example

We wanted to generate the feasible coalitions using the base cases

$$\mathcal{P} = \{\emptyset\}, \quad \mathcal{N} = \{\{1,2,3,4\}, \{1,2,4,5\}, \{5,6\}\}$$



size=1	size = 2	size = 3	size = 4	size = 5	size = 6
1	1, 2	1, 2, 3	1, 2, 3, 4	1, 2, 3, 4, 5	1, 2, 3, 4, 5, 6
2	1, 3	1, 2, 4	1, 2, 3, 5	1, 2, 3, 4, 6	
3	1, 4	1, 2, 5	1, 2, 3, 6	1, 2, 3, 5, 6	
4	1, 5	1, 2, 6	1, 2, 4, 5	1, 2, 4, 5, 6	
5	1, 6	1, 3, 4	1, 2, 4, 6	1, 3, 4, 5, 6	
6	2, 3	1, 3, 5	1, 2, 5, 6	2, 3, 4, 5, 6	
	2, 4	1, 3, 6	1, 3, 4, 5		
	2, 5	1, 4, 5	1, 3, 4, 6		
	2, 6	1, 4, 6	1, 3, 5, 6		
	3, 4	1, 5, 6	1, 4, 5, 6		
	3, 5	2, 3, 4	2, 3, 4, 5		
	3, 6	2, 3, 5	2, 3, 4, 6		
	4, 5	2, 3, 6	2, 3, 5, 6		
	4, 6	2, 4, 5	2, 4, 5, 6		
	5, 6	2, 4, 6	3, 4, 5, 6		
		2, 5, 6			
		3, 4, 5			
		3, 4, 6			
		3, 5, 6			
		4, 5, 6			

Back to our Example

This can be done using the base cases as follows:

$$\mathcal{P} = \{\emptyset\}, \quad \mathcal{N} = \{\{1,2,3,4\}, \{1,2,4,5\}, \{5,6\}\}$$



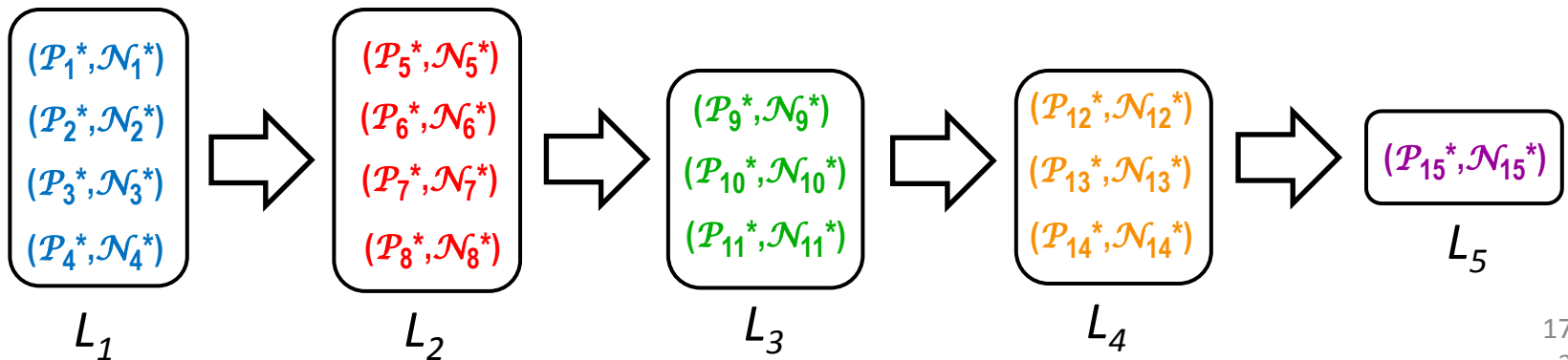
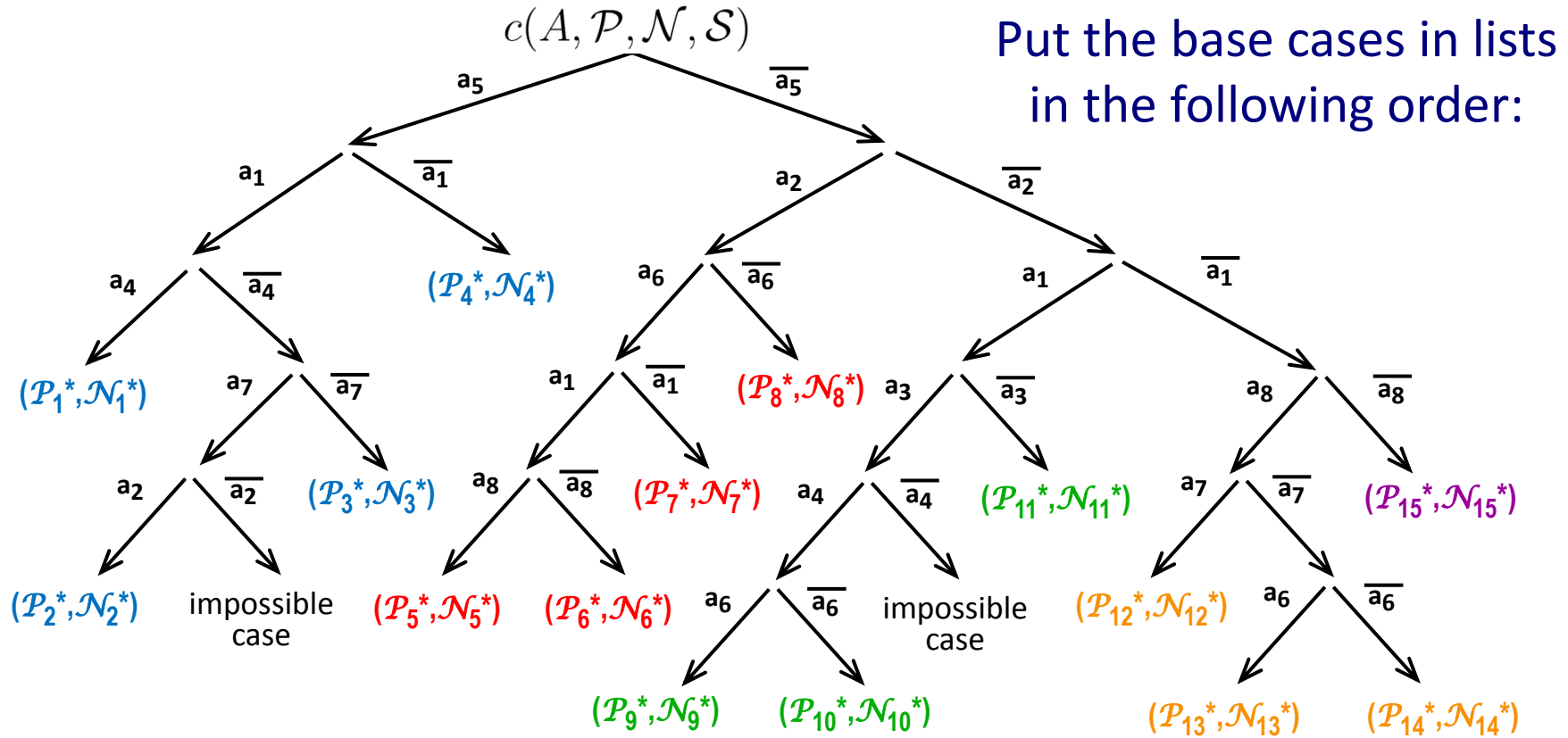
○	$\mathcal{P}_1^* = \{\emptyset\}$	$\mathcal{N}_1^* = \{\{1\}, \{5,6\}\}$
★	$\mathcal{P}_2^* = \{\{1\}\}$	$\mathcal{N}_2^* = \{\{2\}, \{5,6\}\}$
+	$\mathcal{P}_3^* = \{\{1,2\}\}$	$\mathcal{N}_3^* = \{\{3\}, \{4\}, \{5,6\}\}$
△	$\mathcal{P}_4^* = \{\{1,2,4\}\}$	$\mathcal{N}_4^* = \{\{3\}, \{5\}\}$
×	$\mathcal{P}_5^* = \{\{1,2,3\}\}$	$\mathcal{N}_5^* = \{\{4\}, \{5,6\}\}$



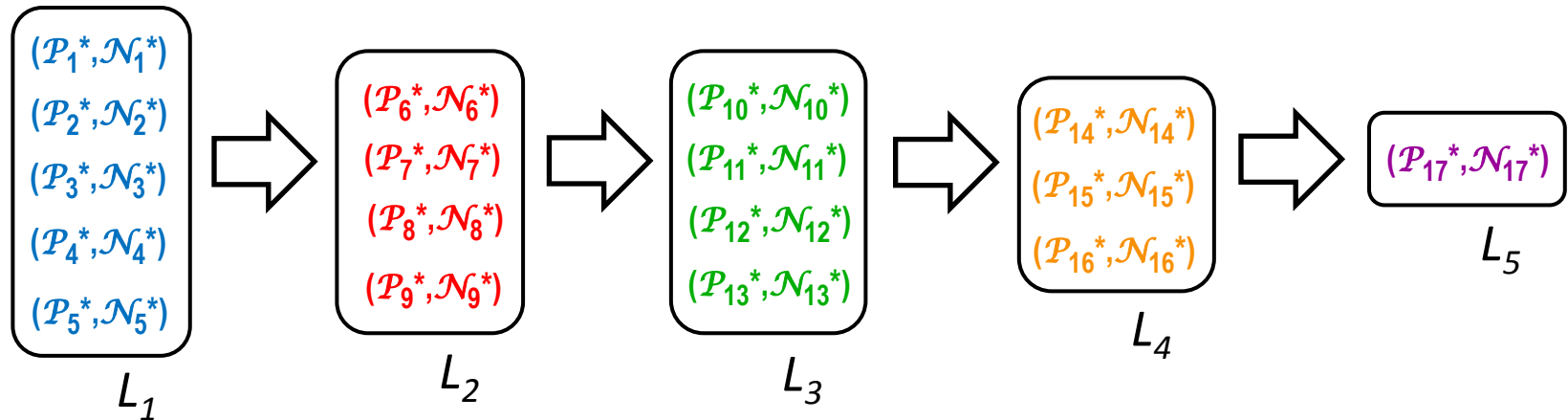
size=1	size = 2	size = 3	size = 4	size = 5	size = 6
★ 1	+	×	1, 2, 3, 4	1, 2, 3, 4, 5	1, 2, 3, 4, 5, 6
○ 2	★ 1, 3	△ 1, 2, 4	×	1, 2, 3, 4, 6	
○ 3	★ 1, 4	+	×	1, 2, 3, 5, 6	
○ 4	★ 1, 5	+	1, 2, 4, 5	1, 2, 4, 5, 6	
○ 5	★ 1, 6	★ 1, 3, 4	△ 1, 2, 4, 6	1, 3, 4, 5, 6	
○ 6	○ 2, 3	★ 1, 3, 5	1, 2, 5, 6	2, 3, 4, 5, 6	
	○ 2, 4	★ 1, 3, 6	★ 1, 3, 4, 5		
	○ 2, 5	★ 1, 4, 5	★ 1, 3, 4, 6		
	○ 2, 6	★ 1, 4, 6	1, 3, 5, 6		
	○ 3, 4	1, 5, 6	1, 4, 5, 6		
	○ 3, 5	○ 2, 3, 4	○ 2, 3, 4, 5		
	○ 3, 6	○ 2, 3, 5	○ 2, 3, 4, 6		
	○ 4, 5	○ 2, 3, 6	2, 3, 5, 6		
	○ 4, 6	○ 2, 4, 5	2, 4, 5, 6		
	5, 6	○ 2, 4, 6	3, 4, 5, 6		
		2, 5, 6			
		○ 3, 4, 5			
		○ 3, 4, 6			
		3, 5, 6			
		4, 5, 6			

Buy how do we search the feasible coalition structures?

Searching Feasible Coalition Structures



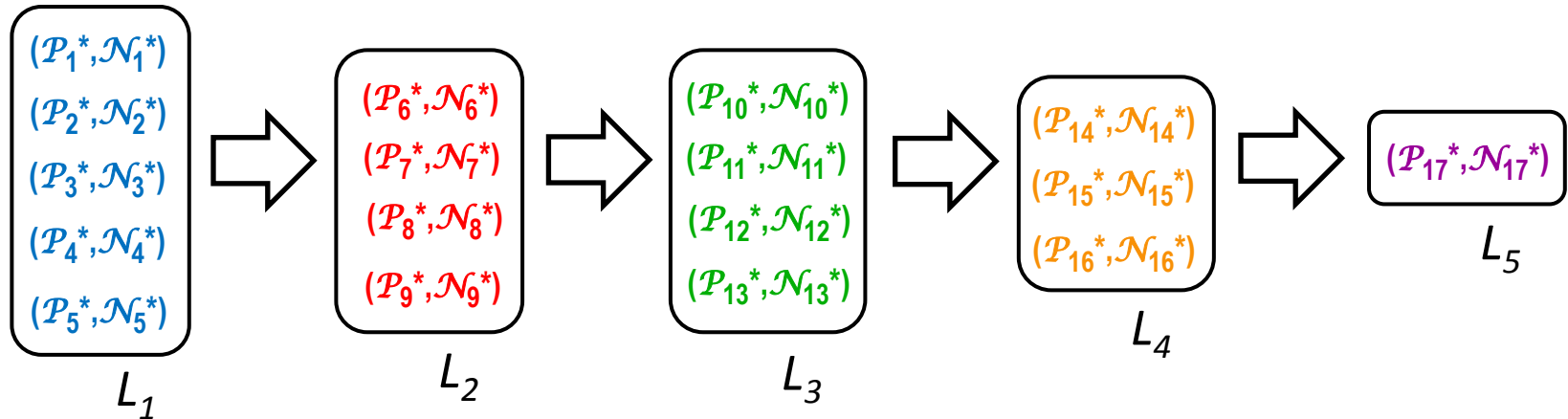
Searching Feasible Coalition Structures



Observation:

- Every feasible coalition structure contains:
 - **exactly one** coalition from L_1 , and
 - **at most one** coalition from $L_i : i = 2, 3, \dots$

Searching Feasible Coalition Structures



Algorithm:

- For every coalition $C_1 \in L_1$:
 - add C_1 to CS and, and
 - add the agents in C_1 to \mathcal{N}_j^* for every $(\mathcal{P}_j^*, \mathcal{N}_j^*) \in L_2$
- If $UCS \neq A$, repeat the process, i.e.,
 - add a coalition $C_2 \in L_2$ to CS , and
 - add the agents in $C_1 \cup C_2$ to \mathcal{N}_j^* for every $(\mathcal{P}_j^*, \mathcal{N}_j^*) \in L_3 \dots$ etc.
- To speed up the search, apply a branch-and-bound technique.