Constrained Coalition Formation

Talal Rahwan[1], Tomasz Michalak[1,2], Edith Elkind[3], Piotr Faliszewski[4], Jacek Sroka[2], Michael Wooldridge[5], Nicholas R. Jennings[1]

[1] School of Electronics and Computer Science, University of Southampton, UK
[2] Institute of Informatics, University of Warsaw, Poland
[3] Division of Mathematical Sciences, Nanyang Technological University, Singapore
[4] Faculty of Elec. Eng., Automatics, Comp. Sci. and Elec., AGH University of Science and Technology, Poland
[5] Department of Computer Science, University of Liverpool, UK

Abstract

The conventional model of coalition formation considers every possible subset of agents as a potential coalition. However, in many real-world applications, there are inherent constraints on feasible coalitions: for instance, certain agents may be prohibited from being in the same coalition, or the coalition structure may be required to consist of coalitions of the same size. In this paper, we present the first systematic study of *constrained coalition formation* (CCF). We propose a general framework for this problem, and identify an important class of CCF settings, where the constraints specify which groups of agents should/should not work together. We describe a procedure that transforms such constraints into a structured input that allows coalition formation algorithms to identify, without any redundant computations, all the feasible coalitions. We then use this procedure to develop an algorithm for generating an optimal (welfare-maximizing) constrained coalition structure, and show that it outperforms existing state-of-the-art approaches by several orders of magnitude.

Introduction

In multi-agent systems, agents can often benefit form working together, i.e., forming coalitions (Shehory and Kraus 1998). Generally speaking, the coalition formation process involves three main activities (Sandholm et al. 1999): (1) determining the value of each coalition; (2) finding a *coalition structure*, i.e., a partition of the set of agents into disjoint coalitions; and (3) dividing the payoffs from collaboration.

Now, the conventional model of coalition formation views every possible subset of agents as a potential coalition. However, in many real-world applications, there are inherent constraints that enforce or, conversely, prohibit the co-existence of certain agents in any coalition. For example, in many countries, anti-trust laws prohibit the formation of certain coalitions of companies (cartels) to prevent such coalitions obtaining and exploiting an unfair market position (e.g., a monopoly). Other types of constraints are also possible. For example, constraints may be placed on coalition sizes, where certain sizes are permitted/prohibited; and certain companies may have preferred contractors – companies they would prefer to work with in a coalition. Clearly, whenever such constraints arise they must be accounted for during all three activities of the coalition formation process.

To date, there have been a number of attempts in the game-theoretic literature to analyze coalition formation with constraints (see the Related Work section). However, this literature only deals with the strategic aspects of the problem. In contrast, the computational aspects, which are of particular interest to the multi-agent systems research community, have been largely ignored: the only work we are aware of was carried out by Shehory and Kraus (1998). They considered limits on the size of the largest coalition that could be formed, but did not consider any other types of constraints. It seems likely that the lack of models, languages, or algorithms for dealing with constraints on coalitional membership hindered the use of coalition formation techniques in many realistic applications where such constraints are present.

To fill this major gap in the literature, we provide: (1) a framework for *constrained coalition formation* (CCF); and (2) an algorithm for optimal coalition structure generation for an important special case of this framework. Central to our algorithmic endeavour is the ability to avoid checking every possible coalition (2^n in total, given n agents) to verify whether it is feasible, i.e., whether it satisfies the constraints. This is crucial, first, because the number of feasible coalitions can be significantly smaller than 2^n and, second, because each one of those 2^n coalitions requires performing a potentially large number of operations—depending on the number of constraints—before its feasibility is determined. A desirable approach should, then, focus only on the feasible coalitions, and should be able to generate them directly from the constraints. One of the main challenges here is to avoid redundant computations, e.g., multiple generation of any coalition that satisfies multiple constraints. Another important challenge in CCF is how the feasible coalitions can be combined to efficiently generate valid coalition structures. Indeed, going through every possible combination of feasible coalitions, and checking whether this combination contains mutually disjoint coalitions can be costly, even for a relatively small domain.

The remainder of the paper is structured as follows. In the following section, we present a general model of constrained coalition formation, and then we identify a simple, but expressive set of constraints that allows for a practical

coalition structure generation algorithm. We then develop a procedure that transforms the specified set of constraints into another, isomorphic, set such that it is possible to identify, without any redundant computations, all the feasible coalitions. Building upon this, we provide a novel algorithm for optimal coalition structure generation in CCF. Specifically, the algorithm exploits our new set of constraints to search through only the feasible coalition structures (i.e., those in which every coalition is feasible). We test our algorithm and show that it outperforms other state-of-the-art algorithms for several orders of magnitude. We conclude the paper by discussing related work, summarizing our results, and proposing directions for future research.

General CCF Model

Given a finite set A of agents, let 2^A denote the set of all subsets of A, i.e., the set of all possible *coalitions*. Moreover, let $\Pi(A)$ denote the set of all partitions of A, i.e., the set of all possible *coalition structures*.

Definition 1. *A* constrained coalition formation (CCF) game *is a tuple* $\mathcal{G} = \langle A, \mathcal{CS}, v \rangle$ *where:*

- $A = \{a_1, \ldots, a_n\}$ *is the set of* agents*;*
- $\mathcal{CS} \subseteq \Pi(A)$ *is the set of feasible coalition structures;*
- $v : (\cup_{C \in \mathcal{CS}} \cup_{C \in CS} \{C\}) \rightarrow \mathbb{R}$ *is the characteristic function, which assigns a real value to every coalition that appears in some feasible coalition structure.*

Note that, in general, the notion of feasibility is defined for coalition structures rather that individual coalitions: for instance, if $A = \{a_1, a_2, a_3, a_4\}$ and we define $\mathcal{CS}$ as the set of all coalition structures in which all coalitions have the same size, then the coalition structure $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ is not feasible, even though each of its component coalitions may be a part of a feasible coalition structure. However, in many settings of interest, the constraints on coalition structures implied by $\mathcal{CS}$ can be reduced to constraints on individual coalitions. More formally, we say that a CCF game $\mathcal{G} = \langle A, \mathcal{CS}, v \rangle$ is *locally constrained* if there exists a set of coalitions $\mathcal{C} \subseteq 2^A$ such that $\mathcal{CS} = \{CS \in \Pi(A) \mid CS \subseteq \mathcal{C}\}$. We will refer to the coalitions in $\mathcal{C}$ as *feasible coalitions*.

It is impractical to represent a CCF game by listing all feasible coalition structures: such a representation is exponential in the number of agents. Indeed, constraints that arise in practical domains usually have an inherent structure, and can therefore be represented succinctly in a suitable language. One obvious language is that of propositional logic in which the Boolean variables of the language correspond to agents. More formally, let us define the set of Boolean variables $B_A = \{b_i \mid a_i \in A\}$, i.e., for every agent a_i we have a corresponding Boolean variable b_i. Now, let ϕ be a propositional formula over B_A, constructed using the usual classical connectives ($\wedge, \vee, \neg, \rightarrow, \ldots$). We say that a coalition C *satisfies* ϕ (and write $C \models \phi$) if ϕ is satisfied under the truth assignment that sets all b_i with $a_i \in C$ to **true** and all b_i with $a_i \notin C$ to **false**. Furthermore, we say that a coalition structure CS satisfies ϕ (and write $CS \models \phi$) if we have $C \models \phi$ for all $C \in CS$. A CCF game $\mathcal{G} = \langle A, \mathcal{CS}, v \rangle$ is said to be *propositionally definable* if there is a logical formula ϕ over B_A such that $\mathcal{CS} = \{CS \mid CS \models \phi\}$. Our next proposition shows that not all CCF games are propositionally definable.

Proposition 1. *The class of propositionally definable CCF games is equal to the class of locally constrained CCF games.*

Proof. Let $\mathcal{C} \subseteq 2^A$ be the set of feasible coalitions in a locally constrained game. Then we can define the propositional constraint ϕ by

$$\phi = \bigvee_{C \in \mathcal{C}} \left(\wedge_{a_j \in C} b_j \right) \wedge \left(\wedge_{a_j \in A \setminus C} \neg b_j \right).$$

Conversely, given a propositional constraint ϕ, we can define the set of local constraints $\mathcal{C}$ as $\mathcal{C} = \{C \mid C \models \phi\}$. □

Proposition 1 shows that, in order to develop a representation for general CCF games, we would have to go beyond propositional logic over B_A. There are many obvious extensions to propositional logic that would be fully expressive[1]. However, our aim in the present paper is to focus on locally constrained games, and in particular, their tractable instances.

Basic CCF Model

Given a succinctly represented CCF game, a key problem is to find a coalition structure with the highest total value. This is not an easy task, even if we limit ourselves to locally constrained games. Thus, in what follows, we will try to identify a natural subclass of such games for which we can develop practical algorithms for this problem. Proposition 1 suggests that we can approach this task by specifying a class of relatively simple propositional formulas, and focusing on coalition structures that can be defined by formulas in this class. We will now describe a class of propositional formulas that, in our opinion, provides a good balance between expressiveness and tractability. As a motivation, we start with the following example.

Example 1. *Suppose that eight web-service providers,* $A = \{a_1, \ldots, a_8\}$*, consider cooperation in order to provide cloud-computing capabilities to a major client. The client knows from prior experience that certain alliances of companies are indispensable to perform this task, and these are* $\{a_1, a_5, a_8\}$*,* $\{a_2, a_5, a_7\}$*, and* $\{a_5, a_7, a_8\}$*. Thus, only coalitions involving any of these alliances are considered to be feasible. Furthermore, the client excludes any coalitions involving alliances of* $\{a_1, a_2, a_3\}$ *or* $\{a_2, a_3, a_5\}$ *due to the fact that, from prior experience, these specific combinations of providers are known to under-perform.*

The above example suggests that a natural way in which constraints arise in coalition formation is in the form of subsets of agents whose collective presence is viewed as useful/harmful. Thus, a CCF game can be specified by:

[1] One example is to allow predicates of the form $I(i, j)$, whose semantics is *"agents* a_i *and* a_j *should be in the same coalition"*. We leave it for the reader to see that such a language is indeed fully expressive.

- a set of *positive constraints* $\mathcal{P} \subseteq 2^A$ such that a coalition C satisfies a constraint $P \in \mathcal{P}$ if $P \subseteq C$;
- a set of *negative constraints* $\mathcal{N} \subseteq 2^A$ such that C satisfies a constraint $N \in \mathcal{N}$ if $N \not\subseteq C$.

It remains to describe how to combine the constraints. We will do so in the language of propositional logic over B_A, by associating every $P \in \mathcal{P}$ with a formula $\phi_P = \wedge_{a_i \in P} b_i$, and every $N \in \mathcal{N}$ with a formula $\phi_N = \neg(\wedge_{a_i \in N} b_i)$.

Observe that negative constraints usually mean that the respective agents should *never* work together, so they should be interpreted conjunctively: each coalition should satisfy each negative constraint. However, for positive constraints the conjunctive approach does not work: if we require that each coalition satisfies each positive constraint, this effectively means that only the grand coalition is feasible, since no positive constraint can be satisfied by more than one coaition simultaneously. Thus, we should interpret the positive constraints disjunctively, i.e., either by saying that each coalition should satisfy at least one constraint, or by saying that each constraint should be satisfied by at least one coalition. In this paper, we focus on the former interpretation, as it seems more closely tied to the issues of collective performance considered in this work; the latter interpretation, which views coalition formation from individual agents' perspective, appears to be more relevant in the context of coalitional stability. Thus, we say that a coalition structure CS is feasible if $CS \models \phi$, where

$$\phi = (\vee_{P \in \mathcal{P}} \phi_P) \bigwedge (\wedge_{N \in \mathcal{N}} \phi_N), \tag{1}$$

Observe that not all locally constrained games can be described by constraints of this restricted form: for instance, a CCF game with 6 agents, where the feasible coalitions are those of even size, cannot be encoded in this way (to see this, observe that since the grand coalition is feasible, we have $\mathcal{N} = \emptyset$). In particular, constraints on the coalition sizes, which are likely to be relevant in many real-life scenarios, cannot always be encoded. Thus, we explicitly add size constraints, denoted $\mathcal{S}$, to our model.

To summarize, we define a *basic CCF game* $\mathcal{G}$ as a tuple

$$\mathcal{G} = \langle A, \mathcal{P}, \mathcal{N}, \mathcal{S}, v \rangle, \tag{2}$$

where $A = \{a_1, \ldots, a_n\}$ is the set of agents, $v : 2^A \to \mathbb{R}$ is a characteristic function on A, $\mathcal{P}$ and $\mathcal{N}$ are sets of subsets of A, and $\mathcal{S} \subseteq \mathbb{N}$. A coalition $C \subseteq A$ is feasible for $\mathcal{G} = \langle A, \mathcal{P}, \mathcal{N}, \mathcal{S}, v \rangle$, if (i) $P \subseteq C$ for some $P \in \mathcal{P}$; (ii) $N \not\subseteq C$ for all $N \in \mathcal{N}$; and (iii) $|C| \in S$; we denote by $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ the set of all feasible coalitions. This is a locally constrained game, i.e, a coalition structure CS is feasible if and only if $CS \subseteq c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$. Given a feasible coalition structure CS, we set $v(CS) = \sum_{C \in CS} v(C)$.

Now, we can state the optimization problem that we would like to solve: given a basic CCF game $\mathcal{G}$, the goal is to find a feasible coalition structure CS^* that maximizes $v(CS)$ among all feasible coalition structures. In the rest of the paper, we present and evaluate an algorithm for this problem.

Constraint Transformation

In this section, we present a procedure that transforms the specified set of constraints into another, isomorphic, set: this enables us to identify, without any redundant computations, all the feasible coalitions. We first present the theoretical basis for our transformation, and then describe an algorithm that performs this transformation.

Theoretical Basis

Given two arbitrary sets of sets, $\mathcal{X}$ and $\mathcal{Y}$, let

$$\mathcal{X} \otimes \mathcal{Y} = \begin{cases} \{X \cup Y \mid X \in \mathcal{X}, Y \in \mathcal{Y}\} & \text{if } \mathcal{X} \neq \emptyset, \mathcal{Y} \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

Further, for any $a_i \in A$, let $\mathcal{P}^{a_i} = \{C \in \mathcal{P} \mid a_i \in C\}$ and $\mathcal{P}^{\overline{a_i}} = \mathcal{P} \setminus \mathcal{P}^{a_i}$. Similarly, let $\mathcal{N}^{a_i} = \{C \in \mathcal{N} \mid a_i \in C\}$ and $\mathcal{N}^{\overline{a_i}} = \mathcal{N} \setminus \mathcal{N}^{a_i}$. Finally, define $\widetilde{\mathcal{P}^{a_i}}$ and $\widetilde{\mathcal{N}^{a_i}}$ as follows:

$$\widetilde{\mathcal{P}^{a_i}} = \{C \in 2^A \setminus \{a_i\} \mid C \cup \{a_i\} \in \mathcal{P}^{a_i}\}, \tag{3}$$

$$\widetilde{\mathcal{N}^{a_i}} = \{C \in 2^A \setminus \{a_i\} \mid C \cup \{a_i\} \in \mathcal{N}^{a_i}\}. \tag{4}$$

For example, given $\mathcal{P} = \{\{a_1, a_2\}, \{a_1, a_3\}, \{a_4, a_5\}\}$, we would have $P^{a_1} = \{\{a_1, a_2\}, \{a_1, a_3\}\}$, $\mathcal{P}^{\overline{a_1}} = \{\{a_4, a_5\}\}$, and $\widetilde{\mathcal{P}^{a_1}} = \{\{a_2\}, \{a_3\}\}$.

With these definitions in place, we can now present the main theorem for subdividing the set $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$. Intuitively, this theorem allows us to decompose the constraints, without losing any information, by choosing a *branching agent* a_i and dividing the problem into two smaller subproblems—with and without a_i, respectively. We omit the proof due to space limitations.

Theorem 1. *For any agent $a_i \in A$, the following holds:*

$$c(A, \mathcal{P}, \mathcal{N}) = c(A \setminus \{a_i\}, \mathcal{P}^{\overline{a_i}}, \mathcal{N}^{\overline{a_i}}) \cup \left(\{a_i\} \otimes c(A \setminus \{a_i\}, \mathcal{P}^{\overline{a_i}} \cup \widetilde{\mathcal{P}^{a_i}}, \mathcal{N}^{\overline{a_i}} \cup \widetilde{\mathcal{N}^{a_i}}) \right).$$

Transformation Algorithm

Our algorithm is based on *Divide and Conquer* — a design paradigm that involves breaking a problem into several subproblems such that they are similar to the original one but smaller in size. This process repeats recursively until the sub-problems become *base cases*, i.e., sub-problems that are small enough to be solved in a straightforward manner. The solutions to those base cases are then combined to create a solution to the original one. Particularly, in our case:

- The *problem* is how to generate $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$.
- The *division* is performed according to Theorem 1.
- A *base case* is a problem where $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ satisfies the following conditions: $|\mathcal{P}| = 1$, $\cap \mathcal{N} = \emptyset$, and $|\{N \in \mathcal{N} \mid |N| > 1\}| \leq 1$.

Our choice of the base case is motivated as follows. Let g_s be a function that takes as input a set of agents $C \subseteq A$, and returns all the possible subsets of C that are of size s. That is, $g_s(C) = \{C' : C' \subseteq C, |C'| = s\}$.[2] Then, given a base case

[2]This can be implemented efficiently using the techniques in (Rahwan and Jennings 2007).

with $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$, we can generate the required coalitions directly, i.e., without checking any constraints, based on the following lemma:

Lemma 1. *Let us define $\mathcal{N}' = \{N \in \mathcal{N} \mid |N| > 1\}$ and $N'' = \mathcal{N} \setminus \mathcal{N}'$. Moreover, let us define $A' = \cup\mathcal{N}'$ and $A'' = A \setminus \cup(\mathcal{P} \cup \mathcal{N}' \cup \mathcal{N}'')$. Then, given a base case, the following holds (where $\times$ denotes the cartesian product):*

$$c(A, \mathcal{P}, \mathcal{N}) = \mathcal{P} \times \bigcup_{s \in \mathcal{S} \cup \{0\}, s < |A'|} g_s(A') \times \bigcup_{s \in \mathcal{S} \cup \{0\}, s \leq |A''|} g_s(A'')$$

We omit the proof due to space limitations.

The pseudo-code of our divide-and-conquer procedure is presented in Algorithm 1, and an example of how the algorithm works is illustrated in Figure 1. To save space, where there is no risk of confusion, we omit commas and brackets, and refer to agents by using their indices only (e.g. we write $\{12\}\{3\}$ as a shorthand for $\{\{a_1, a_2\}, \{a_3\}\}$). The main idea is to divide the problem of generating $c(\mathcal{A}, \mathcal{P}, \mathcal{N}, \mathcal{S})$ into two sub-problems, where the coalitions that contain a certain agent a_i are separated from those that do not. Based on Theorem 1, when generating the coalitions that contain a_i, we can remove a_i from every constraint in $\mathcal{P}$ and $\mathcal{N}$. Furthermore, when generating the coalitions that do not contain a_i, we can remove every constraint in $\mathcal{P}$ and $\mathcal{N}$ that contains a_i. Since this division is performed recursively in our algorithm, we introduce two sets, $\mathcal{P}^*$ and $\mathcal{N}^*$, to keep track of the agents that must (respectively, must not) be contained in the coalitions. This is depicted in Figure 1, where we initially have $\mathcal{P} = \{158\}\{257\}\{578\}$, $\mathcal{N} = \{123\}\{235\}$, and $\mathcal{P}^* = \emptyset$ and $\mathcal{N}^* = \emptyset$. In this example, we divide the coalitions into those that contain agent 1 and those that do not:

- For the coalitions that contain agent 1, we update $\mathcal{P}$, $\mathcal{N}$, and $\mathcal{P}^*$ as follows: $\mathcal{P} = \{\{58\}\{257\}\{578\}\}$, $\mathcal{N} = \{\{23\}\{235\}\}$ and $\mathcal{P}^* = \{\{1\}\}$.
- For the coalitions that do not contain agent 1, we update $\mathcal{P}$, $\mathcal{N}$, and $\mathcal{N}^*$ as follows: $\mathcal{P} = \{\{257\}\{578\}\}$, $\mathcal{N} = \{\{235\}\}$, and $\mathcal{N}^* = \{\{1\}\}$.

Note that the coalitions in every sub-problem must satisfy not only the updated constraints in $\mathcal{P}$ and $\mathcal{N}$, but also those in $\mathcal{P}^*$ and $\mathcal{N}^*$.

Next, we provide a brief description of the main stages of Algorithm 1:

- **Removing Redundant Constraints:** here we remove every negative constraint that is a superset of another negative one, and every positive constraint that is a superset of another positive. This occurs at every call of the recursive function.
- **Dealing with Special Cases:** here we deal with two special cases, where only one constraint, either positive or negative, is left to be satisfied.
- **Checking Termination Conditions:** these are based on the definitions of $\mathcal{P}$ and $\mathcal{N}$, which imply that the constraints in $\mathcal{P}$ are always satisfied if $\mathcal{P} \ni \emptyset$, and the constraints in $\mathcal{N}$ are always satisfied if $\mathcal{N} = \emptyset$. Now, if both $\mathcal{P}$ and $\mathcal{N}$ are satisfied, then we store $(\mathcal{P}^*, \mathcal{N}^*)$ in a set called $\mathcal{T}^*$. The definitions of $\mathcal{P}$ and $\mathcal{N}$ also implies that

Algorithm 1 : $f(A, \mathcal{P}, \mathcal{N}, \mathcal{P}^*, \mathcal{N}^*)$

{---------- REMOVE REDUNDANT CONSTRAINTS ----------}
for $N \in \mathcal{N}$ **do**
  if $\exists N' \in \mathcal{N} : N' \supset N$ **then**
    $\mathcal{N} \leftarrow \mathcal{N} \backslash \{N'\}$ {remove N' from $\mathcal{N}$}
  end if
end for
for $P \in \mathcal{P}$ **do**
  if $\exists P' \in \mathcal{P} : P' \supset P$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \backslash \{P'\}$ {remove P' from $\mathcal{P}$}
  end if
end for
{---------- DEAL WITH SPECIAL CASES ----------}
for $N \in \mathcal{N} : |N| = 1$ {if there is a constraint in $\mathcal{N}$ with exactly one agent} **do**
  $A \leftarrow A \backslash N$ {remove the agent from A}
  $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup N$
  $\mathcal{N} \leftarrow \mathcal{N} \backslash \{N\}$ {remove the constraint from N}
end for
if $\mathcal{P} \ni \emptyset$ and $|\mathcal{N}| = 1$ **then**
  $\mathcal{N}^* \leftarrow \mathcal{N}^* \cup \mathcal{N}$
  $\mathcal{N} \leftarrow \emptyset$
end if
if $|\mathcal{P}| = 1$ and $\mathcal{N} = \emptyset$ **then**
  $\mathcal{P}^* \leftarrow \{\cup_{P \in \mathcal{P} \cup \mathcal{P}^*} P\}$
  $\mathcal{P} \leftarrow \{\emptyset\}$
end if
{---------- CHECK TERMINATION CONDITIONS ----------}
if $\mathcal{P} \ni \emptyset$ and $\mathcal{N} = \emptyset$ **then**
  $\mathcal{T}^* \leftarrow \mathcal{T}^* \cup \{(\mathcal{P}^*, \mathcal{N}^*)\}$
  exit {because we have satisfied all membership constraints}
end if
if $\mathcal{P} = \emptyset$ or $\mathcal{N} \ni \emptyset$ **then**
  exit {because the membership constraints cannot be satisfied}
end if
if ($|\cup\mathcal{P}^*| > Max(\mathcal{S})$) or ($|A| - |\mathcal{N}^*| < Min(\mathcal{S})$) **then**
  exit {because the size constraints cannot be satisfied}
end if
{---------- INITIALIZE DIVIDE & CONQUER ----------}
$a_i \leftarrow select(A, \mathcal{P}, \mathcal{N}, \mathcal{P}^*, \mathcal{N}^*)$ {select an agent}
if $\mathcal{P}^* = \emptyset$ **then**
  $A^* \leftarrow A^* \cup \{a_i\}$
end if
$\mathcal{N}^{\overline{a_i}} \leftarrow \emptyset$, and $\widetilde{\mathcal{N}^{a_i}} \leftarrow \emptyset$ {initialize $\mathcal{N}^{\overline{a_i}}$ and $\widetilde{\mathcal{N}^{a_i}}$}
for $N \in \mathcal{N}$ **do**
  if $a_i \in N$ **then**
    $\widetilde{\mathcal{N}^{a_i}} \leftarrow \widetilde{\mathcal{N}^{a_i}} \cup \{N \backslash \{a_i\}\}$
  else
    $\mathcal{N}^{\overline{a_i}} \leftarrow \mathcal{N}^{\overline{a_i}} \cup \{N\}$
  end if
end for
$\mathcal{P}^{\overline{a_i}} \leftarrow \emptyset$, and $\widetilde{\mathcal{P}^{a_i}} \leftarrow \emptyset$ {initialize $\mathcal{P}^{\overline{a_i}}$ and $\widetilde{\mathcal{P}^{a_i}}$}
for $P \in \mathcal{P}$ **do**
  if $a_i \in P$ **then**
    $\widetilde{\mathcal{P}^{a_i}} \leftarrow \widetilde{\mathcal{P}^{a_i}} \cup \{P \backslash \{a_i\}\}$
  else
    $\mathcal{P}^{\overline{a_i}} \leftarrow \mathcal{P}^{\overline{a_i}} \cup \{P\}$
  end if
end for
{---------- APPLY DIVIDE & CONQUER ----------}
$f(A \backslash \{a_i\}, \mathcal{P}^{\overline{a_i}} \cup \widetilde{\mathcal{P}^{a_i}}, \mathcal{N}^{\overline{a_i}} \cup \widetilde{\mathcal{N}^{a_i}}, \{\cup_{P \in \mathcal{P}^* \cup \{a_i\}} P\}, \mathcal{N}^*)$
$f(A \backslash \{a_i\}, \mathcal{P}^{\overline{a_i}}, \mathcal{N}^{\overline{a_i}}, \mathcal{P}^*, \mathcal{N}^* \cup \{a_i\})$

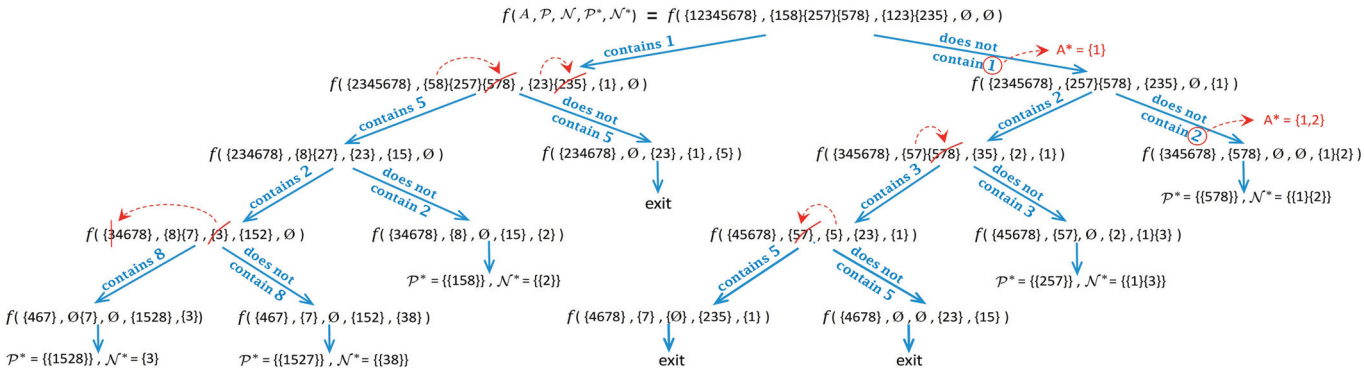

Figure 1: How the elements in $\mathcal{T}^*$ are generated given $A = \{1,2,3,4,5,6,7,8\}$, $\mathcal{P} = \{1,5,8\}\{2,5,7\}\{5,7,8\}$, and $\mathcal{N} = \{1,2,3\}\{2,3,5\}$.

Algorithm 2 : *findOptimalCS*$(A^*, \mathcal{T}^*)$

```
for i = 1 to |A*| do
    L_i ← ∅ ; UB_{L_i} ← −∞ {initialize L_i and UB_{L_i}}
for (P*, N*) ∈ T* do
    foundList ← false
    for i = 1 to |A*| {if P* contains a*_i then update L_i, UB_{L_i}} do
        if a*_i ∈ P* then
            L_i ← L_i ∪ (P*, N*)
            for C ∈ c(P*, N*) do
                if v(C) > UB_{L_i} then
                    UB_{L_i} ← v(C) {update UB_{L_i}}
            foundList ← true
            break;
    if foundList = false then
        L_{|A*|+1} ← L_{|A*|+1} ∪ (P*, N*)
searchLists(1, L_1, ∅)
```

Algorithm 2.1 : *searchLists*(k, L, CS)

```
for (P*, N*) ∈ L do
    for C ∈ c(P*, N*) do
        CS' = CS ∪ {C}
        if ⋃CS' = A then
            if V(CS') > V(CS**) then
                CS** ← CS' {update CS**}
        else
            if (k ≤ |A*|) and (V(CS') + Σ_{j=k+1}^{|A*|+1} UB_{L_j} > V(CS**)) then
                L' ← L_{k+1}
                {add agents in CS' to the negative constraints in L'}
                for (P*, N*) ∈ L' do
                    for C' ∈ CS' do
                        for a_i ∈ C' do
                            N* ← N* ∪ {{a_i}}
                searchLists(k + 1, L', CS')
```

the constraints cannot be satisfied if $\mathcal{P} = \emptyset$ or $\mathcal{N} \ni \emptyset$. Thus, we check if this is the case. We also check whether the size constraints can no longer be satisfied (Note that the size constraints are also dealt with once a base case is reached, see Lemma 1).

- **Initializing Divide & Conquer:** First, we select a branching agent, which is stored in an ordered set A^* only if $\mathcal{P}^* = \emptyset$ (see Figure 1). The element at the i^{th} location in A^* will be denoted a_i^*, and the rationale behind using A^* will be made clear in the next section. Finally, in this stage, we set $\mathcal{P}^{\overline{a_i}}$, $\mathcal{N}^{\overline{a_i}}$, $\widehat{\mathcal{P}^{a_i}}$ and $\widehat{\mathcal{N}^{a_i}}$, as they are needed in the next stage.
- **Applying Divide & Conquer:** We divide the current problem into two sub-problems based on Theorem 1.

Due to the way the constraints (i.e., subsets) in $\mathcal{P}$ and $\mathcal{N}$ are updated, which involves either the removal of agents from subsets or the removal of entire subsets, the algorithm is guaranteed to reach one of the following termination conditions: $\mathcal{P} = \emptyset, \mathcal{N} \ni \emptyset$, or $\mathcal{P} \ni \emptyset \wedge \mathcal{N} = \emptyset$. Also note that if the constraints in $\mathcal{P}$ and $\mathcal{N}$ are satisfied, i.e., if $\mathcal{P} \ni \emptyset \wedge \mathcal{N} = \emptyset$, then the only constraints that need to be satisfied are those in $\mathcal{P}^*$ and $\mathcal{N}^*$. Furthermore, the way $\mathcal{P}^*$ and $\mathcal{N}^*$ are constructed guarantees that these constraints, i.e., $(\mathcal{P}^*, \mathcal{N}^*)$, are always base cases.

Coalition Structure Generation in CCF

Recall that our goal is to find a coalition structure CS^* that maximizes $v(CS)$ among all feasible coalitions. One of the main challenges when tackling this problem is how to combine the feasible coalitions in order to efficiently generate valid coalition structures. Indeed, going through every possible combination of feasible coalitions, and checking whether it contains mutually disjoint coalitions can be costly, even for a relatively small problem. Our approach makes use of the transformation outlined in the previous section. Specifically, we divide the set of feasible coalitions $c(A, \mathcal{P}, \mathcal{N}, \mathcal{S})$ into lists $L_1, \ldots, L_{|A^*|+1}$, where $L_i = \{(\mathcal{P}^*, \mathcal{N}^*) : \mathcal{P}^* \ni a_i^*\}$ for $i = 1, \ldots, |A^*|$, and $L_{|A^*|+1} = \{(\mathcal{P}^*, \mathcal{N}^*) : \mathcal{P}^* \cap A^* = \emptyset\}$. The rationale behind this division is the following lemma.

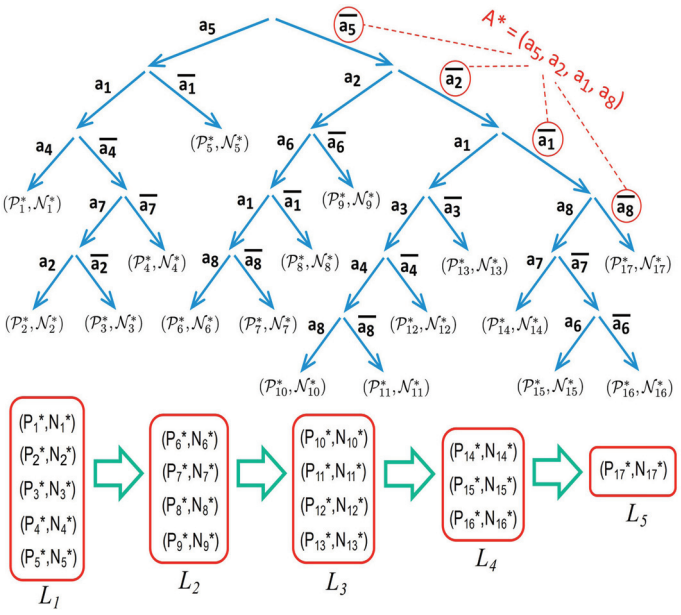

Figure 2: The order by which the sub-spaces are searched when searching for the optimal coalition structure

Lemma 2. *Every feasible coalition structure CS contains exactly one coalition from L_1 and at most one coalition from every L_i, where $i \in \{2, \ldots, |A^*| + 1\}$.*

The pseudo code of our algorithm is presented in Algorithm 2, and an example of how it operates is illustrated in Figure 2. Basically, the algorithm fills the lists $L_1, \ldots, L_{|A^*|+1}$ and computes an upper bound UB_{L_i} on the values of the coalitions from every list L_i (see Steps 1-14). In the next step it calls the function $searchLists$ which works as follows. It keeps adding coalitions to a variable CS until this variable becomes a feasible coalition structure, in which case it updates, if needed, the best solution found so far (denoted CS^{**} in the algorithm). More specifically, for every coalition C_1 from L_1, the coalition is added to CS and its members are added to $\mathcal{N}^*$ for every $(\mathcal{P}^*, \mathcal{N}^*) \in L_2$. This places further constraints on the coalitions in L_2, and that is not to contain any agent in C_1. If CS is not yet a feasible coalition structure, then the process is repeated for L_2, i.e., for every coalition C_2 from L_2, the coalition is added to CS and the agents in C_1 and C_2 are added to $\mathcal{N}^*$ for every $(\mathcal{P}^*, \mathcal{N}^*) \in L_3$, and so on. To speed up the search, the algorithm applies a *branch and bound* technique. Specifically, before moving into a list L_i, the algorithm compares the upper bounds of all remaining lists to the current best solution see whether it is worthwhile to make that move (see Step 8).

Performance Evaluation

In this section, we empirically evaluate our algorithm, and benchmark it against the state of the art in the literature. In particular, we run our experiments given: (i) different numbers of agents, ranging from 15 to 30; (ii) different numbers of randomly generated constraints varying from 100 to 1000; and (iii) different coalition-value distributions. Specifically,

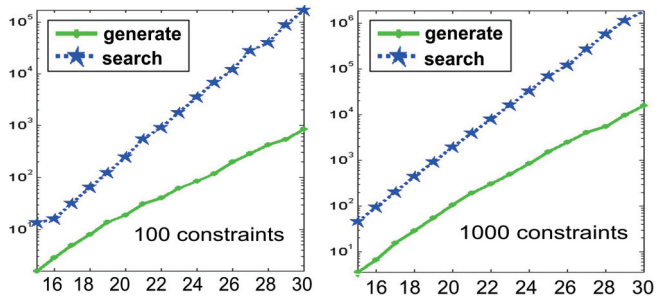

Figure 3: Given different numbers of agents, the figure shows *on a log scale* the time (in milli-seconds) to generate, or search for, the feasible coalitions.

we use the standard test distributions that are widely used in the Coalition Structure Generation (CSG) literature, namely: Normal, Uniform, and NDCS (Rahwan et al. 2009). Our experiments were carried out on a PC with 8 Intel 2.67GHz processors and 12GB of RAM.

ALGORITHM1: Given different numbers of agents, Figure 3 shows, *on a log scale*, a comparison between the time to *generate* the feasible coalitions (using our divide-and-conquer procedure), and the time to *search for* the feasible coalitions (by checking every coalition in 2^A against the constraint to verify whether it is a feasible one). As can be seen, our procedure is faster by orders of magnitude. For instance, given 30 agents, ours takes less than 1% of the time, and that is for both the 100-constraints case and the 1000-constraints one.

ALGORITHM2: We test our algorithm against:

- Rahwan et al. (2009) — the state-of-the-art algorithm for solving the CSG problem when all coalitions are feasible;
- Ohta et al. (2009) — the state-of-the-art for CSG problems with compact representations of coalitional games.

For fairness, all 3 algorithms (including ours) were given the same input: the coalition values and a list of constraints. Rahwan et al.'s algorithm was implemented such that it checks every possible coalition against the list of constraints, and stores the feasible ones in memory. After that, it proceeds as usual, except that every branch in the search tree is pruned if it contains a coalition that is not feasible. The process of checking the feasibility of coalitions is done very efficiently (using a logarithmic number of operations). As for Ohta et al., the MIP formulation for Synergy Coalition Groups was implemented using ILOG's CPLEX — a standard mixed integer programming (MIP) package.

Given different numbers of agents, Figure 4 shows *on a log scale* the termination time given the Normal distribution.[3] As shown in the figure, our algorithm is faster by several orders of magnitude, and that is for all distributions. Given 30 agents, for example, the algorithm takes only 0.002% compared to Rahwan et al. in the 100-constraints case, and only 0.4% int the 1000-constraints one. Similarly, when comparing against Ohta et al. given 30 agents, we find

[3] Similar patterns were observed for the other distributions.

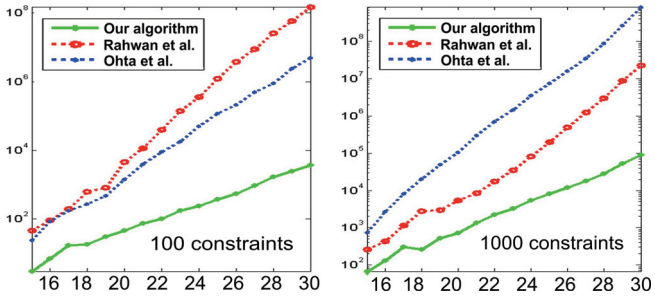

Figure 4: The time to solve the CSG problem given different algorithms, and different numbers of agents (log scale).

that our algorithm takes only 0.07% of the time given a 100 constraints, and only 0.01% given a 1000. As can be seen, these improvements grow exponentially as the number of agents increases.

It should be noted that many of these problem instances were indeed very challenging due to the large number of feasible coalitions. For instance, the average number feasible coalitions in our experiments was nearly 8×10^7 given 100 constraint with 30 agents, and nearly 6×10^8 given 1000 constraints.

Related Work

Coalitional games with constraints have been analyzed by a number of authors from a game-theoretic perspective. The dominant approach in this literature is to define the set of feasible coalitions based on convenient mathematical structures such as distributive lattices (Faigle 1989), convex geometries (Bilbao and Edelman 2000), antimatroids (E. Algaba and Jimenez-Losada 2004), and augmenting systems (Bilbao and Ordez 2009). In the general case, however, none of these structures is able to model the coalitional game with arbitrary positive and negative constraints as defined in this paper. The work that is most closely related to our setting is (Demange 2009), where a set of all feasible coalitions is entirely arbitrary. Nevertheless, this game-theoretical study focuses solely on strategic, core-related issues rather than computational analysis.

In the computer science literature, constraints on coalition sizes have been considered in the context of coalition value calculation (Shehory and Kraus 1998; Rahwan and Jennings 2007), but no other types of constraints have been explored. Also, the extensive literature on (anytime) coalition structure generation, e.g., (Rahwan et al. 2009; Sandholm et al. 1999), always treats all coalitions as feasible. Another active stream of relevant research concerns developing succinct and expressive representations for coalitional games (Deng and Papadimitriou 1994; Ieong and Shoham 2005; Conitzer and Sandholm 2006). Indeed, for locally constrained games any such formalism could be used to encode the constraints. However, it can be shown that none of these formalisms is succinct for the class of basic CCF games considered in this paper. Therefore, approaches that work directly with such representations, such as that of (Ohta et al. 2009) perform poorly compared to ours.

Conclusions and Extensions

We have introduced the framework of Constrained Coalition Formation and identified a special case of this framework, which we called basic CCF games, that captures many types of constraints typical of multiagent settings. We developed a procedure that transforms the constraints of a basic CCF game into a structured input that allows coalition formation algorithms to identify, without any redundant computations, all the feasible coalitions. Building upon this we proposed an algorithm for optimal Coalition Structure Generation in basic CCF games.

Our work makes the first steps towards a study of constrained coalition formation in the multi-agent systems domain, where agent cooperation plays a key role. Important directions of future research include generalizing our results to richer classes of constrained games and analyzing computational properties of the key stability concepts, e.g., the Core and the Shapley, adapted to accommodate constraints. Finally, it is very interesting to explore the relationship of our work to the domain of combinatorial auctions.

References

Bilbao, J. M., and Edelman, P. H. 2000. The shapley value on convex geometries. *Discrete Appl. Math.* 103(1-3):33–40.

Bilbao, J., and Ordez, M. 2009. Axiomatizations of the shapley value for games on augmenting systems. *European Journal of Operational Research* 196(3):1008–1014.

Conitzer, V., and Sandholm, T. 2006. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence (AIJ)* 170(6-7):607–619.

Demange, G. 2009. The strategy structure of some coalition formation games. *Games and Economic Behavior* 65(1):83–104.

Deng, X., and Papadimitriou, C. 1994. On the complexity of cooperative solution concepts. *Mathematical Oprerational Research* (19):257–266.

E. Algaba, J. M. Bilbao, R. v. d. B., and Jimenez-Losada, A. 2004. Cooperative games on antimatroids. *Discrete Mathematics* 282:1–15.

Faigle, U. 1989. Cores of games with restricted cooperation. *Mathematical Methods of Operations Research* 33(6).

Ieong, S., and Shoham, Y. 2005. Marginal contribution nets: A compact representation scheme for coalitional games. In *Proceedings of the 6th ACM EC'05*.

Ohta, N.; Conitzer, V.; Ichimura, R.; Sakurai, Y.; Iwasaki, A.; and Yokoo, M. 2009. Coalition structure generation utilizing compact characteristic function representations. In *CP'09*, 623–638.

Rahwan, T., and Jennings, N. R. 2007. An algorithm for distributing coalitional value calculations among cooperative agents. *Artificial Intelligence (AIJ)* 171(8–9):535–567.

Rahwan, T.; Ramchurn, S. D.; Giovannucci, A.; and Jennings, N. R. 2009. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research (JAIR)* 34:521–567.

Sandholm, T. W.; Larson, K.; Andersson, M.; Shehory, O.; and Tohme, F. 1999. Coalition structure generation with worst case guarantees. *Artificial Intelligence (AIJ)* 111(1–2):209–238.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial Intelligence (AIJ)* 101(1–2):165–200.