Computing Sequences of Coalition Structures

Tabajara Krausburg
School of Technology; Department of Informatics
PUCRS; Clausthal University of Technology
Porto Alegre, Brazil; Clausthal, Germany
tabajara.rodrigues@edu.pucrs.br

Jürgen Dix
Department of Informatics
Clausthal University of Technology
Clausthal, Germany
dix@tu-clausthal.de

Rafael H. Bordini
School of Technology
PUCRS
Porto Alegre, Brazil
rafael.bordini@pucrs.br

***Abstract*—We consider the problem of finding optimal sequences of coalition-structures in realistic applications where organisational structure is an important factor. In particular, we analyse sequential characteristic-function games induced by valuation structures, and define *feasible coalition structure sequences* as a solution concept for such games. Besides the restrictions on feasible sequences, the valuation structures allow us to express further constraints on the formation of coalitions. We present a new dynamic-programming algorithm for the computation of sequences of feasible coalition structures and evaluate it on several scenarios. Although this is a hard problem, we show that constraints inspired by real-world applications reduce the size of the search space significantly.**

***Index Terms*—Coalition Structure Generation, Interdependence of Coalitional Games**

I. Introduction

Sequences of Characteristic Function Games (SCFG) is a novel topic in the coalition structure generation literature [1]. In a Characteristic Function Game (CFG), one partitions a set of agents A in which each component of this partition is called a coalition $C \subseteq A$ and the partition itself is a Coalition Structure (CS). The goal is then to maximise the value of a CS CS given a *characteristic function* $v : 2^A \to \mathbb{R}$, $V(CS) = \sum_{C \in CS} v(C)$. In an SCFG, one is interested in outcomes of a sequence of games that have constraints on what coalition structures can follow each other in a totally ordered sequence; a sequence that respects such constraints is called *feasible*. This is an exciting new topic of research addressing problems that cannot be captured by simply solving games in isolation, as not all sequences of coalition structures are allowed. Therefore, just picking each game's optimal coalition structure does not lead to an overall optimal solution. Furthermore, all games are played simultaneously, which means a CS in the first game also depends on the remaining sequence. To model such feasibility, a binary relation on the set of coalition structures is used.

Considering a single CFG, constraints have long been studied in the coalition structure generation problem [2] and a number of approaches have been proposed, for instance, graph-based games [3] and rules-based games [4]. Another approach is Valuation Structures (VS), introduced in [5], which can be used to further express constraints on each CFG in an SCFG instance. For example, with such structures we can easily add constraints on a set of *pivotal agents* (which cannot appear together in the same coalition) and constrain a coalition based on an *interaction graph* (coalitions must induce a connected sub-graph of the interaction graph). Consider, for instance, a trust relationship among agents, which can be easily described by an interaction graph.

The contributions of this paper are threefold:

1) We combine two frameworks for coalition formation available in the literature, namely, sequential CFGs and valuation structures. The latter provides the means to explicitly inform constraints to be applied on coalitions when forming a sequence of coalition structures. The new form of the game is named Sequential CFGs induced by VSs (SEQVS).
2) We give an exact algorithm based on dynamic programming that solves SEQVS instances. This new algorithm can be adapted to deal with SCFG instances which makes it the first exact algorithm for sequential CFGs (apart from the brute-force approach).
3) We empirically evaluate our algorithm and show that it is by several orders of magnitude better than brute force.

Section II presents related work and motivates our work with a real-world application in disaster rescue management. In Section III, we introduce SEQVS. In Section IV, we introduce our exact algorithm to solve SEQVSs, which is based on dynamic programming, and prove that it is correct. In Section V, we experimentally evaluate those algorithms comparing them to brute force. We summarise the main notations and terms in a glossary in the end of this paper.

II. Related Work

In this section we discuss the work our approach is based upon, particularly sequential CFGs and valuation structures. We only present the main ideas as we give the formal definition of the combination of both frameworks in the next section. After discussing some related work on the interdependence of games, we motivate our approach by a real-world example in disaster management.

A. *Sequential* CFGs

In recent work, we introduced sequential characteristic-function games (SCFG) [1], which are games that include a totally ordered sequence of CFGs and require a *feasible sequence of coalition structures* as a solution for a game

T. K. acknowledges funding by CAPES—Brasil—Finance Code 001.
R. H. B. gratefully acknowledges the support from CNPq and CAPES.

instance. Such a solution is not just the optimal coalition structure for each CFG because there are additional constraints. In particular, a game instance includes a relation $\mathcal{R}$ determining which coalition structures can follow each other and therefore are allowed in the sequence, so the optimal overall (feasible) sequence of coalition structures is what is sought. We have also shown that several real-world problems require such sequences of coalition structures, for example the ICS discussed in Section II-D.

B. Valuation Structures

The concept of valuation structure (VS) [5] helps to constrain the coalitions that can be formed in a particular CFG. The authors show that there exists problems that are solved either by rules (prohibition/permission) or by topologically-oriented means. To address that limitation, the authors proposed a framework that combines both worlds: prohibition constraints [4] and an interaction graph [3], [6]. Therefore, it is a promising direction to further constrain coalitions allowed to form in each game in a sequence of CFGs. In addition, the authors propose a structure to modify the characteristic function of a game based on whether a coalition contains or not one of the special agents called *pivotal*. The resulting framework makes it easier to model various known problems as CFG, in particular multicut, multiway cut, k-clustering, and chromatic partitioning problems [5].

C. Approaches to the Interdependence of Games

Apart from SCFG, partition-function games [7] address the interdependence of a coalition and its coalition structure. That means, the value of a coalition is influenced by the remaining coalitions in the CS. In combinatorial auctions [8], the aim is to partition a set of items for a set of buyers which may evaluate differently each subset of items. In clustering, a topic of interest is when one requires k partitions of the data that maintain quality and are dissimilar from one another (*top-k* clustering [9]). We further advance SCFG by enabling constraints on individual positions in a sequence of interdependent games.

D. An Example

The practical application we intend to model is in disaster management. More precisely, we focus on the Incident Command System (ICS) [10], which was initially developed to deal with a series of wildfire events occurring in southern California in 1970. Since then, it has become so popular that it is now adopted and used by FEMA (in the US) to respond to disaster events [11].

While the ICS's overall organisational structure is much richer, we focus on the Operations Sections as it already contains several key components. It acts upon the damaged area of the disaster event. Its hierarchical organisation is composed of branches in the upper level, then divisions and groups in the second, and strike teams, task forces and single resources in the bottom level. An interesting feature of the ICS framework is called *span of control* [11], the ratio of command among supervisors and subordinate units: $\frac{1}{\lambda}$; meaning that one supervisor is responsible for at most λ subordinate units. Applying this feature to the coalition structure generation problem [2] means that a coalition in the upper organisational level may be split into at most λ coalitions in the following level (i.e., the subordinate units). This mechanism establishes a *chain of command* in the ICS, in which each resource *must* have a supervisor.

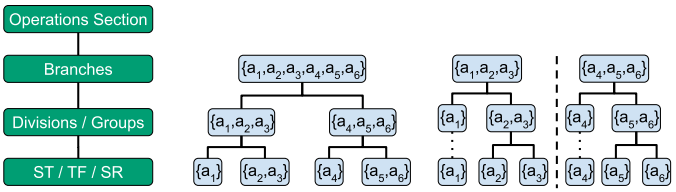

Fig. 1. The Operation Sections hierarchical organisation on the left. On the right side, two sequences of coalition structures of length three that represents a three-level hierarchy of agents starting at different CSs.

Figure 1 illustrates the problem we are facing: a series of coalition structures, further refined at each subsequent level, which has been shown that it can be modelled as an SCFG [1]. The problem is to find the best coalition structure, i.e. the best assignment of agents into branches, groups, etc., respecting several constraints coming from the overall ICS process (size of groups, supervisors in each group, etc.). The question is then, how to represent all constraints induced by an ICS instance over the sequence of coalition structures.

The success of the ICS framework in practice will depend on many factors [12, Section 4], and from those we highlight: (1) how well trained the participants are for using the ICS; (2) the pre-incident relationship between supervisors with one another as well as with their subordinates; and (3) the authority system must be recognised as legitimate. This leads to a need to specify the coalitions allowed to be formed as well as the chain of command among the agents. To this end, we extend the original SCFG so that one can use it to model the constraints coming from each particular level.

III. Sequential Games

Our aim is to model various problems, such as the ICS mentioned above, as a *sequence of characteristic-function games induced by valuation structures*. The solution of such a problem is then a sequence of coalition structures with certain properties (specific for each practical application).

Section III-A contains the main definition introduced in this paper. In Section III-B, we consider the efficient representation of relations $\mathcal{R}$ of our SCFG.

A. Main Definition

Before stating the main definition, we explain the valuation structure (VS) framework from [5], which we slightly adapt for our purposes.

A *valuation structure* σ on a CFG game $\Gamma = \langle A, v\rangle$ is a tuple $\langle G, S, \alpha, \beta, x, y\rangle$, where: (1) $G = \langle A, E\rangle$ is an undirected graph on the set of agents A (called the *interaction graph*); (2) $S \subseteq A$ is a set of *pivotal agents*; (3) α and β

are mappings $S \to \mathbb{R}$ assigning to each pivotal agent a real number; and (4) $x, y \in \mathbb{R}$ are real numbers.

In this setting, the graph G and the set of pivotal agents S determine the set of *allowed* coalitions: a coalition $C \subseteq A$ is allowed to form only if the induced sub-graph of C over G is connected and C contains at most one pivotal agent: $|C \cap S| \leq 1$. This induces the set $\boldsymbol{\mathcal{CS}}^{\sigma}$ of all allowed coalition structures over A: all coalitions in it must be allowed as described above. The mappings α, β and the constants x, y are used to modify a given valuation v as follows:

$$val_v(C) = \begin{cases} \alpha(a_l) \times v(C) + \beta(a_l), & \text{if } C \cap S = \{a_l\}; \\ x \times v(C) + y, & \text{if } C \cap S = \emptyset. \end{cases}$$

In this work, we are interested in constraints applied to coalitions and therefore, we consider only G and S and let out the remaining variables that modify a valuation. Doing so, a VS σ will be represented in the remaining of this work by a tuple $\langle G, S \rangle$. Without loss of generality, one might assume $\alpha : S \to \mathbb{R} : s \mapsto 1$, $\beta : S \to \mathbb{R} : s \mapsto 0$, and $x = 1, y = 0$.

Given an ordinary CFG $\Gamma = \langle A, v \rangle$, we let Γ^{σ} be the game $\langle A, v \rangle$ induced by a valuation structure $\sigma = \langle G, S \rangle$. Our main notion below is an ordered sequence of games $\langle \Gamma_1^{\sigma_1}, \ldots, \Gamma_h^{\sigma_h} \rangle$ which requires a corresponding sequence of coalition structures $\langle CS_1, \ldots, CS_h \rangle$ as solution.

Definition 1 (Sequential CFGs induced by VSs (SEQVS))**.** *A sequential* CFG *is a tuple $\mathcal{G} = \langle A, \mathcal{H}, \Pi, \mathcal{R} \rangle$, where:*

- A *is a set of agents;*
- $\mathcal{H}$ *is a totally ordered set $\Gamma_1 = \langle A, v_1 \rangle, \ldots, \Gamma_h = \langle A, v_h \rangle$ of* CFGs*;*
- Π *is a totally ordered set consisting of $\sigma_i = \langle G_i, S_i \rangle$ with the* VS *parameters described above, one for each game $i = 1, \ldots, h$;*
- $\mathcal{R}$ *is a binary relation on the set of all coalition structures $\boldsymbol{\mathcal{CS}}^A \cup \{\emptyset\}$, so that $(CS_i, CS_{i+1}) \in \mathcal{R}$ iff coalition structure CS_{i+1} may follow CS_i in a sequence of coalition structures forming a solution to the game (as defined below); $(\emptyset, CS_i) \in \mathcal{R}$ means that CS_i is allowed to appear in the beginning of a solution sequence.*

This tuple determines the sequence $\boldsymbol{\Gamma} = \langle \Gamma_1^{\sigma_1}, \ldots, \Gamma_h^{\sigma_h} \rangle$ of CFGs *induced by* VSs*, with $\Gamma_i = \langle A, v_i \rangle$ and $\sigma_i = \langle G_i, S_i \rangle$, where v_i are characteristic functions and $\sigma_i \in \Pi$, for $i = 1, \ldots, h$ (h denotes the length of both sequences).*

We now define a solution concept for such games.

Definition 2 (SEQVS Optimisation Problem)**.** *We call a sequence $\boldsymbol{CS} = \langle CS_1, \ldots, CS_h \rangle$ of coalition structures from $\boldsymbol{\mathcal{CS}}^A$, a potential solution for $\langle \Gamma_1^{\sigma_1}, \ldots, \Gamma_h^{\sigma_h} \rangle$ if: (1) each CS_i is a solution of $\Gamma_i^{\sigma_i}$, and (2) it respects the relation $\mathcal{R}$: $CS_i \; \mathcal{R} \; CS_{i+1}$, $0 \leq i < h$ (we set $CS_0 = \emptyset$). We call such a sequence a Feasible Coalition-Structure Sequence (*FCSS*). In case a sequence does not follow the constraints above, we simply omit the* F *and call it a Coalition-Structure Sequence (*CSS*). We use a function $\mathcal{V}$ to determine a value $\mathcal{V}(\boldsymbol{CS})$ for any* FCSS *(and* CSS*) $\boldsymbol{CS}$, which is given by $\sum_{i=1}^{h} V_i(CS_i)$, where $V_i(CS_i) = \sum_{C \in CS_i} v_i(C)$, $1 \leq i \leq h$.*

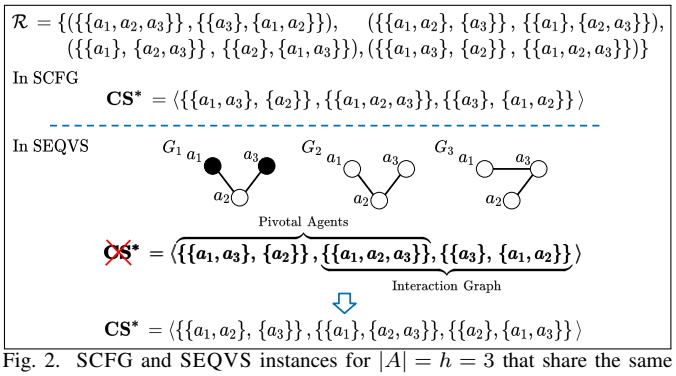

Fig. 2. SCFG and SEQVS instances for $|A| = h = 3$ that share the same $\mathcal{R}$. An optimal solution in the SCFG instance is no longer feasible in the SEQVS one due to the pivotal agents in the first game (solid black nodes in G_1) and to the graph G_3 in the third game. For readability, we omit in $\mathcal{R}$ the pairs that start with $\emptyset$; assume that for all CSs in it such pair exists.

A solution for a SEQVS *game instance is an optimal* FCSS $\boldsymbol{CS}^* = \arg\max_{\boldsymbol{CS}} \mathcal{V}(\boldsymbol{CS})$.

The pair $(\emptyset, CS)$ extends the original definition of an SCFG by determining which CSs may begin a sequence. In practical applications, this might reduce significantly the search space as we do not search in the space of *pairs* of coalition structures, instead, in the space of single coalition structures. Moreover, one should think of the sequence $\langle CS_1, \ldots, CS_h \rangle$ as increasingly refined or interdependent coalition structures at subsequent levels[1]. The relation $\mathcal{R}$ puts constraints on which structures are allowed at the next level.

Furthermore, by adding Valuation Structures to a SCFG instance, one can further specify which CSs are allowed in each position of the sequence. Usually in an SCFG setting, any pair can be placed in any position of the sequence as long as it does not make it unfeasible. For instance, consider Figure 2 in which an optimal solution in an SCFG instance is no longer feasible in a SEQVS setting.

B. Examples of Relations on Coalition Structures

In previous work [1], we have defined various instances of $\mathcal{R}$ to model certain kinds of SCFGs with interesting applications. Here, we concentrate on the following four relations:

Definition 3 ($\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$)**.** *Given any two coalition structures $CS, CS' \in \boldsymbol{\mathcal{CS}}^A$, we define*

- $\mathcal{R}_1$: $(CS, CS') \in \mathcal{R}_1$ *iff* $CS \neq CS'$;
- $\mathcal{R}_2$: $(CS, CS') \in \mathcal{R}_2$ *iff* $|CS| = |CS'|$;
- $\mathcal{R}_3$: $(CS, CS') \in \mathcal{R}_3$ *iff for all $C' \in CS'$ there is a $C \in CS$ such that $C' \subseteq C$ and $|CS| < |CS'|$;*
- $\mathcal{R}_4$: $(CS, CS') \in \mathcal{R}_4$ *iff* $CS = CS'$.

$\mathcal{R}_3$ is the relation that corresponds to Figure 1: it models a hierarchical structure, which is important for various application scenarios. The first two relations (representing constraints that are quite general) are used in later sections for our empirical evaluation, as worst-case scenarios. $\mathcal{R}_1$ forces us

[1]Note that we use *level* and *position* interchangeably to denote the position of a CS in the sequence of length h.

to compare with the maximum number of coalition structures, while $\mathcal{R}_2$ reduces this search space significantly. The last relation, although synthetic, allows us to investigate how exact algorithms behave in case no branching during the search is allowed, that is, only one option of CS is feasible.

In many cases we use an implicit representation of $\mathcal{R}$, as opposed to an exhaustively generated list of all possible pairs. This allows us to keep $\mathcal{R}$ compactly represented and only make calls-by-need using a particular algorithm (whereby allowed pairs of CSs can be either *checked* or *generated* on-the-fly by following a few simple rules).

IV. An Exact Algorithm for SEQVS

An optimal FCSS can be computed by searching the entire space of sequences of coalition structures exhaustively. We introduce two exact algorithms that compute an optimal FCSS. The first one performs a brute-force search and is therefore exponential in the number of agents and games. We then improve on its performance by constructing an algorithm based on dynamic programming. We later experiment with these two exact algorithms in the next section.

A. Brute-Force Algorithm

To search for an FCSS, we need to consider the number of agents and games, the valuation structures, and the binary relation $\mathcal{R}$. The idea of the brute force algorithm is to construct the FCSS iteratively, and at each iteration, to check the feasibility of introducing a new CS' at level l in the sequence, following a coalition CS. The feasibility consists of (1) $(CS, CS') \in \mathcal{R}$; and (2) $CS' \in \boldsymbol{\mathcal{CS}}^{\sigma_l}$, that is, all pivotal agents at level l remain in different coalitions in CS', and given a coalition $C \in CS'$, the induced sub-graph of C over the corresponding interaction graph at level l is connected.

Definition 4 (X_l^{CS})**.** *Given a coalition structure CS and a particular level l of the sequence, $1 \leq l \leq h$, we let X_l^{CS} be the set of all coalition structures CS' such that conditions (1) and (2) above are satisfied.*

Once a sequence is complete, that is, the length of a candidate FCSS matches the number of games in $\mathcal{H}$, we compare its value against the best FCSS found so far and keep the one with the highest value.

B. Dynamic Programming Algorithm

The motivation for using a dynamic programming technique is that the feasibility of inserting a new CS into a sequence is given only by the last element of the sequence (a pair of CSs that is in $\mathcal{R}$). So our problem displays *optimal substructure* regarding CS, because we can determine an optimal subsequence of coalition structures to CS, then use memoisation to keep such partial results, and avoid recomputing the optimal subsequences when we need to try the various possible next CS$'$ that may follow each of those subsequence. Our algorithm then constructs such sequences iteratively from their beginning. That means we only need to keep in memory the subsequences at one level of the game sequence—in particular those subsequences that have optimal value up to the CSs at that level—besides the ones for the next level. We give a recursive definition of the value of an optimal FCSS below.

Lemma 1. *Given a coalition structure CS, the value of an optimal $\boldsymbol{CS}$ starting with CS (where CS is feasible at level $l = 1$) can be computed by $f(CS, 1)$, where $f(CS, l)$ is defined recursively as follows:*

$$\begin{cases} V_h(CS) & \text{if } l = h; \\ -\infty & \text{if } X_{l+1}^{CS} = \emptyset; \\ V_l(CS) + \max_{CS' \in X_{l+1}^{CS}}(f(CS', l+1)) & \text{otherwise.} \end{cases}$$

In the algorithm, to compute the value of an optimal $\boldsymbol{CS}$ for a given SEQVS instance, we iterate over the pairs in $X_1^{\emptyset}$ and use Lemma 1, as follows: $\boldsymbol{CS}^* = \max_{CS \in X_1^{\emptyset}}(f(CS, 1))$.

Note that the recursive definition given above needs to return a particular value (i.e., $-\infty$) for the non-feasible sequences. We address that differently in the algorithm itself.

We use four auxiliary tables in Algorithm 1 to determine an optimal FCSS. Two tables store the values of sequences for two subsequent iterations. The other two tables record the optimal subsequences themselves. Note that the size of the tables is bounded by the number of pairs in $\mathcal{R}$ and VS for that particular iteration.

Algorithm 1 SDP

Input:
A, a set of agents
$\mathcal{H}$, a sequence of games
Π, a sequence of VSs
$\mathcal{R}$, a binary relation
Output: $\boldsymbol{CS}^*$, an optimal FCSS

```
1:  INITIALISE_LIST(f, t)
2:  for all CS ∈ X_1^∅ do
3:  |  f[CS] ← V_1(CS)
4:  |  t[CS] ← ⟨CS⟩
5:  for i ← 2 to h do
6:  |  INITIALISE_LIST(f', t')
7:  |  for all CS ∈ t do
8:  |  |  CS ← CS[i − 1]
9:  |  |  for all CS' ∈ X_i^CS do
10: |  |  |  value ← f[CS] + V_i(CS')
11: |  |  |  if value > f'[CS'] then
12: |  |  |  |  f'[CS'] ← value
13: |  |  |  |  t'[CS'] ← CS · ⟨CS'⟩
14: |  (f, t) ← (f', t')
15: return t[arg max(f)]
```

The first step is to go through the feasible starting CSs and store them and their values in the two tables (lines 2-4). To check the feasibility of a sequence, we use the same procedure as in the brute force algorithm (Section IV-A). We always use as index for the tables the last CS inserted in the subsequence: we know precisely which game we should solve for a given CS based on its position in the subsequence.

The second step is to iterate from the second game until we reach h. We record the values and sequences in two temporary tables so as not to overwrite the last iteration's records while they are still needed. For all subsequences recorded in the last iteration, we get the last element CS of it and iterate over X_i^{CS} where i represents the next position in the sequence (lines 7-9). For each $CS' \in X_i^{CS}$, we concatenate it with the subsequence and record the new sequence and value in the two temporary tables only if its value is higher than the one already stored for that CS' (i.e., the index). At the end of each iteration, we overwrite the content of the tables from the last iteration (i.e., f and t) with the contents of the ones for the current iteration.

To retrieve an optimal FCSS $\boldsymbol{CS}^*$, we search for the maximum value in table f and retrieve its index. The retrieved index corresponds to the optimal FCSS in table t.

Theorem 1 (Correctness)**.** *For any given* SEQVS*, SDP determines an optimal* FCSS $\boldsymbol{CS}$.

Proof. Assume an optimal FCSS $\boldsymbol{CS}^* = \langle CS_1, \ldots, CS_h \rangle$. By the definition of the set X, we have $CS_1 \in X_1^{\emptyset}$, $CS_2 \in X_2^{CS_1}$, ..., $CS_h \in X_h^{CS_{h-1}}$. For all $CS \in X_1^{\emptyset}$, SDP stores them in memory (i.e., at index CS, in t, it stores CS as a subsequence and in f its value); they represent subsequences of length 1. Let I denote the set of all indexes currently in t. Then, in the next iteration i, SDP goes through the set $X = \bigcup_{CS' \in I} X_i^{CS'}$, that is, the set of all the coalition structures that are allowed to follow all current subsequences. For every $CS \in X$, we search in t the subsequences that are feasible when CS is appended to them and pick the one of highest value in f. Note that CS is the end of a CSS and we only need to keep one of its precedent subsequences; the one of highest value. We do that by properly updating the tables f'$[CS]$ and t'$[CS]$. At the end of each iteration, tables f and t are always updated with the optimal subsequences and values. We repeat the procedure until we reach level h and therefore we construct $\boldsymbol{CS}^*$ (i.e., it is in t). SDP essentially uses the same process as in Lemma 1, but filtering out subsequences that are not feasible. To select the $\boldsymbol{CS}^*$ as the outcome, we just need to return the sequence for index $\arg\max$ f. □

We note that the optimal substructure property discussed above is inherent of SCFG instances, and therefore SDP is also a suitable tool to solve them.

V. Empirical Evaluation

In this section, we experiment with the two algorithms described above and with MC-Link [1], which we adapt from the original paper to our experiments. MC-Link is a heuristic hierarchical-clustering algorithm (without quality guarantees) that outputs an FCSS (MC-Link is inspired by C-Link [13]). It was initially designed to check the feasibility of a sequence based only on $\mathcal{R}$, but we here extended it to handle VSs and the relations introduced in Section III-B. Furthermore, we insert a further check to make sure a sequence of CSs holds the feasibility property. For instance, in $\mathcal{R}_2$, the presence of pairs of CSs in $\mathcal{R}_2$ is based on their sizes, such a transition is a known limitation of MC-Link [1]. We implemented all algorithms in Python (version 3.8.5). All the experiments were performed on a virtual machine containing 32 GB of RAM and a CPU with four single cores of 2095 MHz each. All material is available at https://github.com/smart-pucrs/SCFG.

All four relations allow every CS to begin an FCSS: for every $CS \in \boldsymbol{CS}^A$ it is the case that $(\emptyset, CS) \in \mathcal{R}$ (to not reduce the search space, i.e., CSs that might start a sequence, in our experiments). We implemented *generative* and *checking* algorithms for all four relations. We use the former for the brute force (short bf in the charts) and SDP algorithms, and the latter as input to MC-Link. To improve readability, we omit the lines when $h = 4$ in Figure 3 and 4, as the behaviour in that setting is similar the other hs depicted in those charts.

For each game $\Gamma \in \mathcal{H}$, we draw a value for any coalition $C \in 2^A$ from $v(C) \sim N(\mu, \sigma^2)$ where $\mu = |C|$ and $\sigma = \sqrt{|C|}$. This distribution is called **NDCS** and was introduced in [14]. We use **NDCS** because it had intermediate results compared to other distributions in the experiments with MC-Link [1]. Regarding the VSs, We generate all interaction graphs randomly: an edge connects any two agents if $p \leq 60$ where $p \sim U(0, 100)$. Regarding the pivotal agents, we randomly pick, from A, q agents and insert them into the corresponding set of pivotal, where $q \sim U(0, \lceil \frac{n}{3} \rceil)$. We use $\frac{n}{3}$ to avoid picking up all agents from A as pivotal agents.

For all experiments, we use a table to store in advance the values for all coalitions according to how many games are required (the value for a coalition is different in each game). Besides, for all experiments, we set a *timeout* of one hour.

A. Time Analysis

We experiment with the number of agents and games for all algorithms. We show the results in Figure 3.

We see in the results that SDP is faster by several orders of magnitude in $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ if compared to the brute force approach for any h. In $\mathcal{R}_4$, as only one candidate is possible for the next position in the sequence, both algorithms had a similar running time. The same occurs when $h = 2$, as this means that an FCSS is a single pair of CSs in $\mathcal{R}$, and therefore SDP cannot take advantage of an optimal substructure during the construction of a sequence. Besides, as we increase the number of games h for any given n, the running time of SDP does not increase significantly; this is because it constructs the sequences keeping in memory the subsequences. However, the number of agents plays a significant role, and therefore is an issue to be addressed. MC-Link, as a heuristic approach, is faster by at least one order of magnitude than both exact algorithms, usually trading off the quality of the outcome for a shorter running time, as we depict in Figure 5. Moreover, we note that MC-Link is not even complete as in some cases, namely $\mathcal{R} = \mathcal{R}_3$ for $|A| = 5$ and $|A| = 10$, it does not find an existing FCSS.

B. Memory Analysis

In this experiment, we record the peak of memory used by each algorithm. It represents how much memory we need

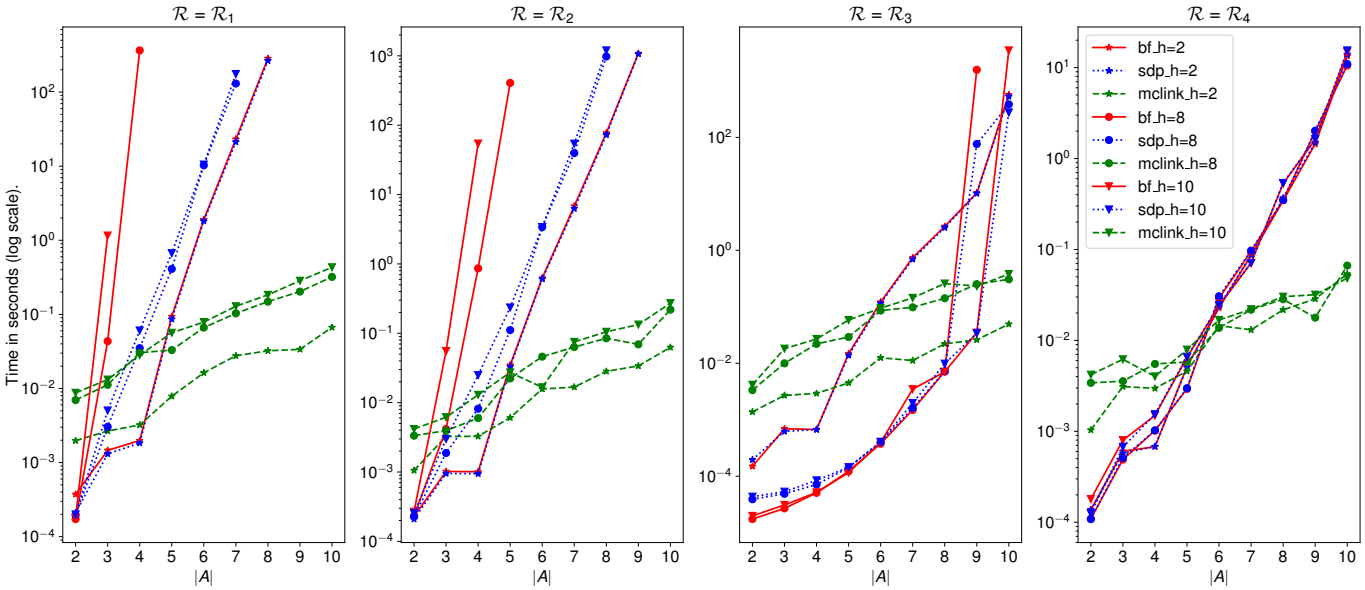

Fig. 3. Comparison of running time in log scale between the brute-force (bf), SDP, and MC-Link algorithms. In case a timeout is fired, we do not depict the corresponding point in the $|A|$ axis for that particular h. For instance, line `bf_h=8` when $\mathcal{R} = \mathcal{R}_1$ had timeouts from $|A| = 5$ to $|A| = 10$.

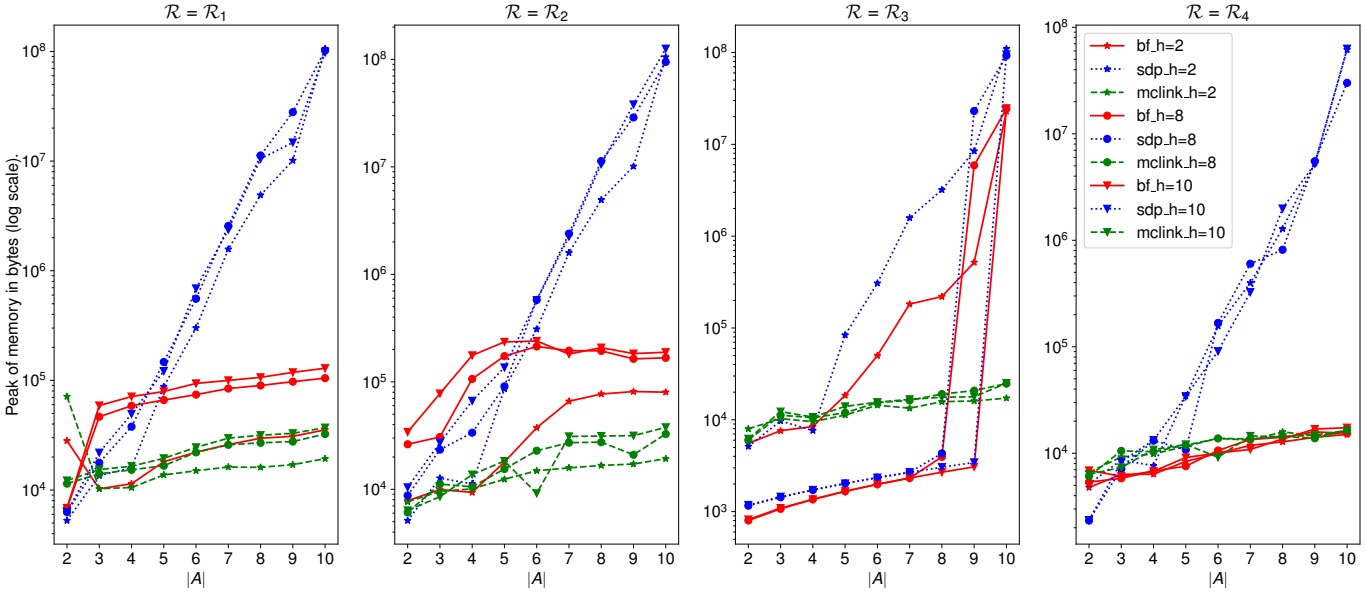

Fig. 4. Comparison of peak of memory in log scale between the brute-force (bf), SDP, and MC-Link algorithms.

to have available to run them. For that purpose, we use the Python package `malloc-tracer` (version 1.7.0)[2] to capture the memory consumption of the process that is running the algorithm instance. We depict the results in Figure 4.

Note that brute-force and SDP swap their positions in comparison to the running-time charts. As expected, SDP consumes more memory, by several orders of magnitude, than the brute-force. We also note that the amount of memory needed by each relation is very similar. That is because all CSs are feasible in the beginning of a sequence. This holds independently of the number of games h, as we use the most memory in the transition from one level to the next one. We expect that real-world applications will further reduce the number of CSs allowed to start a sequence.

We also note that the memory consumption for $\mathcal{R}_3$ is low when there is no solution. For instance, when the number of agents is $|A| = 7$, we can have a hierarchy of, at most, 7 levels.

[2]https://pypi.org/project/malloc-tracer/

VI. Conclusions and Future Work

We introduced a new problem where a sequence of characteristic-function games are induced by valuation structures (SEQVS). Moreover, we introduced an algorithm based on dynamic programming that solves SEQVSs instances. We empirically evaluated the algorithm as well as an adaption of a heuristic one to work on SEQVS, and we showed through

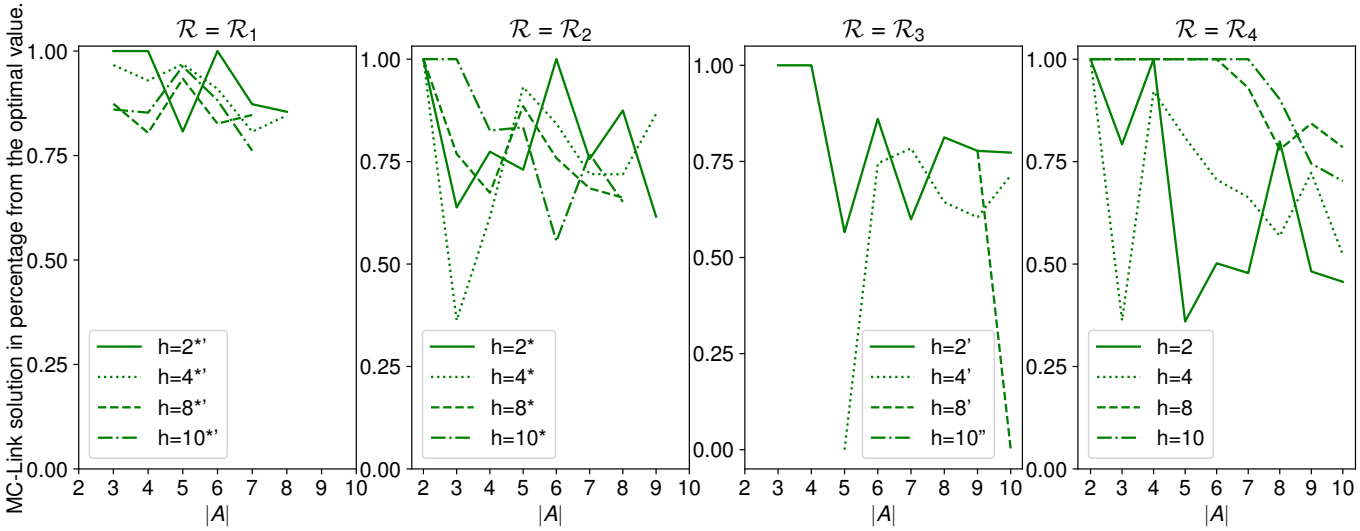

Fig. 5. Ratio of the quality of an FCSS returned by MC-Link regarding the optimal value generated by SDP. We distinguish between two states of affairs: (i) SDP had a timeout (see Figure 3); and (ii) there exists no solution, given the constraints. In case (i), we indicate it by marking the corresponding legend of the line with *. For instance, line `h=8` when $\mathcal{R} = \mathcal{R}_1$ and $|A| = 8, \ldots, |A| = 10$. Similarly in case (ii), we mark the legend of the line with $'$ when *some* solutions for a given h do not exist. For instance, instance $|A| = 2$ and $h = 8$ when $\mathcal{R} = \mathcal{R}_1$ has no solution. If there exists no solution for a given h in all instances $|A| = 2, \ldots, |A| = 10$, then the corresponding line is not depicted and its legend is marked with $''$ (e.g., line `h=10` when $\mathcal{R} = \mathcal{R}_3$).

empirical testing that for binary relations inspired by real-world applications the size of the search space is significantly reduced.

In future work, we aim to investigate heuristics that can be employed by algorithms to generate pairs of coalition structures belonging to the SCFG relations on-the-fly. We also aim to investigate tractable classes of instances of SEQVS.

References

[1] T. Krausburg, J. Dix, and R. H. Bordini, "Feasible coalition sequences," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, (Richland, SC), p. 719–727, IFAAMAS, 2021.

[2] T. Rahwan, T. P. Michalak, M. Wooldridge, and N. R. Jennings, "Coalition structure generation: a survey," *Artificial Intelligence*, vol. 229, pp. 139–174, Dec. 2015.

[3] T. Voice, S. D. Ramchurn, and N. R. Jennings, "On coalition formation with sparse synergies," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 223–230, 2012.

[4] T. Rahwan, T. P. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N. R. Jennings, "Constrained coalition formation," in *Proceedings of the 25th International Conference on Artificial Intelligence*, pp. 719–725, AAAI Press, 2011.

[5] G. Greco and A. Guzzo, "Constrained coalition formation on valuation structures," *Artificial Intelligence*, vol. 249, pp. 19–46, Aug. 2017.

[6] F. Bistaffa, A. Farinelli, J. Cerquides, J. Rodríguez-Aguilar, and S. D. Ramchurn, "Anytime coalition structure generation on synergy graphs," in *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems*, pp. 13–20, 2014.

[7] R. M. Thrall and W. F. Lucas, "N-person games in partition function form," *Naval Research Logistics Quarterly*, vol. 10, no. 1, pp. 281–298, 1963.

[8] P. Krysta and C. Ventre, "Combinatorial auctions with verification are tractable," *Theoretical Computer Science*, vol. 571, pp. 21 – 35, 2015.

[9] G. P. Guedes, E. Ogasawara, E. Bezerra, and G. Xexeo, "Discovering top-k non-redundant clusterings in attributed graphs," *Neurocomput.*, vol. 210, p. 45–54, Oct. 2016.

[10] R. L. Irwin, "The incident command system (ICS)," in *Disaster response: Principles of preparation and coordination* (E. Auf der Heide, ed.), St. Louis, MO: C.V. Mosby, 1989.

[11] FEMA, *National Incident Management System*. Independently Published, 3 ed., Oct. 2017.

[12] J. Jensen and W. L. Waugh Jr, "The united states' experience with the incident command system: What we think we know and what we need to know more about," *Journal of Contingencies and Crisis Management*, vol. 22, no. 1, pp. 5–17, 2014.

[13] A. Farinelli, M. Bicego, F. Bistaffa, and S. D. Ramchurn, "A hierarchical clustering approach to large-scale near-optimal coalition formation with quality guarantees," *Engineering Applications of Artificial Intelligence*, vol. 59, pp. 170–185, Dec. 2016.

[14] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci, "An anytime algorithm for optimal coalition structure generation," *Journal of Artificial Intelligence Research*, vol. 34, pp. 521–567, Mar. 2009.

Acronym	Var	Meaning
—	A	a set of agents
—	C	a subset of agents (coalition)
CS	CS	a partition of A (coalition structure)
—	$\boldsymbol{\mathcal{CS}}^A$	the set of all coalition structures over A
—	$v(C)$	a function mapping $2^A \to \mathbb{R}$
—	$V(CS)$	$\sum_{C \in CS} v(C)$
CFG	Γ	a characteristic-function game
—	G	an interaction graph
—	S	a set of pivotal agents
VS	σ	a tuple $\langle G, S \rangle$
—	$\boldsymbol{\mathcal{CS}}^\sigma$	the set of all CSs induced by a VS σ
—	Γ^σ	a CFG game induced by a VS σ
—	$\mathcal{H}$	a totally ordered sequence of CFGs
—	h	the length of $\mathcal{H}$
—	$\mathcal{R}$	a binary relation on $\boldsymbol{\mathcal{CS}}^A$
CSS	$\boldsymbol{CS}$	a sequence of CSs
FCSS	$\boldsymbol{CS}$	a feasible sequence of CSs
—	$\boldsymbol{CS}^*$	an optimal FCSS
—	$\mathcal{V}(\boldsymbol{CS})$	$\sum_{i=1}^{h} \sum_{C \in CS_i} v_i(C) : CS_i \in \boldsymbol{CS}$
SCFG	—	a tuple $\langle A, \mathcal{H}, \mathcal{R} \rangle$
—	Π	a totally ordered set of VSs
—	$\boldsymbol{\Gamma}$	a sequence of CFGs induced by VSs
SEQVS	$\mathcal{G}$	a tuple $\langle A, \mathcal{H}, \Pi, \mathcal{R} \rangle$
—	X_l^{CS}	a set of CSs compatible with the CS CS at level l