ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

TABAJARA KRAUSBURG RODRIGUES

SEQUENCES OF COALITION STRUCTURES IN MULTI-AGENT SYSTEMS APPLIED TO DISASTER RESPONSE

Porto Alegre (Brazil)
2022

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

SEQUENCES OF COALITION STRUCTURES IN MULTI-AGENT SYSTEMS APPLIED TO DISASTER RESPONSE

TABAJARA KRAUSBURG RODRIGUES

Doctoral dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of doctor of philosophy.

Advisor: Prof. Dr. Rafael H. Bordini (PUCRS)
Prof. Dr. Jürgen Dix (TUC)

Porto Alegre (Brazil)
2022

Ficha Catalográfica

R696s Rodrigues, Tabajara Krausburg
Sequences of coalition structures in multi-agent systems applied to disaster response / Tabajara Krausburg Rodrigues. – Porto Alegre, 2022.
145 f.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica, PUCRS.

Orientador: Prof. Rafael Heitor Bordini
Orientador: Prof. Jürgen Dix (Clausthal University of Technology)

1. Coalitional Games. 2. Coalition Structure Generation. 3. Interdependence of Coalition Structures. 4. Practical Applications of Cooperative Games. 5. Disaster Response. I. Bordini, Rafael Heitor. II. Dix, Jürgen. III. Título.

Bibliotecária responsável: Loiva Duarte Novak – CRB10/2079

TABAJARA KRAUSBURG RODRIGUES

SEQUENCES OF COALITION STRUCTURES IN MULTI-AGENT SYSTEMS APPLIED TO DISASTER RESPONSE

This doctoral dissertation has been submitted in partial fulfillment of the requirements for the degree of doctor of philosophy, of the Computer Science Graduate Program, School of Technology of the Pontifical Catholic University of Rio Grande do Sul.

Sanctioned on January 29th, 2022.

COMMITTEE MEMBERS:

Prof. Dr. Sven Hartmann (TUC)

Prof Dr. Jörg P. Müller (TUC)

Prof. Dr. Avelino F. Zorzo (PUCRS)

Prof. Dr. Rafael H. Bordini (PUCRS - Advisor)

Prof. Dr. Jürgen Dix (TUC - Advisor)

Sequences of Coalition Structures in Multi-Agent Systems Applied to Disaster Response

Doctoral Thesis
(Dissertation)

to be awarded the degree of

Doctor rerum naturalium (Dr. rer. nat.)

submitted by

Tabajara Krausburg Rodrigues
from Caxias do Sul, Brazil

approved by the

Faculty of Mathematics/Computer Science and Mechanical Engineering
Clausthal University of Technology

Date of oral examination
29.03.2022

Chairperson of the Board of Examiners
Prof. Dr. Sven Hartmann (TUC)

Chief Reviewer
Prof. Dr. Jürgen Dix (TUC)
Prof. Dr. Rafael H. Bordini (PUCRS)

Reviewer
Prof Dr. Jörg P. Müller (TUC)
Prof. Dr. Avelino F. Zorzo (PUCRS)

SEQUÊNCIAS DE ESTRUTURAS DE COALIZÕES EM SISTEMAS MULTIAGENTES APLICADAS A RESPOSTA A DESASTRES

RESUMO

A formação de coalizões é um tópico de interesse da comunidade científica que estuda sistemas multiagentes devido aos desafios emergentes na utilização dessa técnica em aplicações práticas, assim como em virtude da complexidade envolvida para computar uma solução para o problema. Uma coalizão é uma organização de curta duração de agentes formada para atingir um objetivo em comum de seus integrantes. A teoria dos jogos cooperativos estabelece um mecanismo formal para análise dos grupos formados por diferentes agentes: as coalizões. Assim, o problema é modelado utilizando jogos de funções características (do inglês *Characteristic-Function Game* (CFG)) no qual o produto final de tal jogo é chamado de estrutura de coalizões: uma partição de um conjunto de agentes em coalizões. Entretanto, nem todos os problemas encontrados na prática podem ser resolvidos eficientemente utilizando uma única estrutura de coalizões. Por exemplo, pode ser necessário a formação de uma hierarquia de grupos na qual uma estrutura de coalizões é requerida por nível hierárquico. Na presente tese, problemas de formação de coalizões que são interdependentes são investigados. Especificamente, jogos de formação de coalizões são resolvidos individualmente e existe uma interdependência entre as soluções dos diferentes jogos. Visto a escassez de trabalhos científicos nesse tópico, um novo jogo é proposto, chamado de jogos sequenciais de funções características (do inglês *Sequential Characteristic-Function Game* (SCFG)), o qual visa modelar o relacionamento entre estruturas de coalizões subsequentes para o problema descrito por uma sequência de CFGs correspondente. O novo jogo proposto é estendido para modelar restrições induzidas sobre cada CFG na sequência de jogos. Além disso, por meio de uma análise teórica conclui-se que o problema subjacente ao SCFG é **PSPACE**-completo. Considerando uma perspectiva algorítmica, um algoritmo exato para computar soluções de instancias SCFG, assim como dois algoritmos heurísticos, são propostos. O desafio final do presente trabalho é modelar uma operação de resposta a desastres que emprega o sistema de comando de incidentes (do inglês *incident command system*), utilizando as técnicas e algoritmos propostos.

Palavras-Chave: Jogos Coalizionais, Geração de Estruturas de Coalizões, Interdependência de Estruturas de Coalizões, Applicações Práticas de Jogos Cooperativos, Resposta a Desastres.

SEQUENCES OF COALITION STRUCTURES IN MULTI-AGENT SYSTEMS APPLIED TO DISASTER RESPONSE

ABSTRACT

Coalition formation has long been an interesting topic of research in Multi-Agent Systems, either for its practical applications or complexity issues. A coalition is commonly understood as a short-lived and goal-directed structure, in which the agents join forces to achieve a goal. Cooperative game theory has been used as a formal mechanism to analyse the problem of grouping agents into coalitions. The problem is then modelled by a Characteristic-Function Game (CFG) in which the outcome is a coalition structure: a partition of agents into coalitions. However, not all problems can be efficiently solved using a single coalition structure. For instance, one might be interested in a group hierarchy in which a coalition structure per level is required. In this thesis, we investigate coalition formation problems that are interdependent. In particular, we focus on the interdependence among solutions (i.e., coalition structures) produced by each game individually. Given the lack of work on this topic, we propose a novel game named Sequential Characteristic-Function Game (SCFG), which aims to model the relationships between subsequent coalition structures in a sequence of CFGs. We approach the resulting problem under both theoretical and practical perspectives. We extend the proposed game to allow fine-grained constraints being induced over each CFG in the sequence. Also, we show that the underlying SCFG problem is **PSPACE**-complete. From an algorithmic viewpoint, we propose an exact algorithm based on dynamic programming, as well as two heuristic algorithms to compute solutions for SCFG instances. We show that there exists a trade-off in choosing one algorithm over the others. Moreover, we model a disaster response operation that employs the incident command system framework, and we show how one can apply our proposed framework and algorithms to solve such an interesting problem.

Keywords: Coalitional Games, Coalition Structure Generation, Interdependence of Coalition Structures, Practical Applications of Cooperative Games, Disaster Response.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1. INTRODUCTION

Multi-Agent Systems (MAS) is a sub-area in Artificial Intelligence that investigates the interaction between different software entities. Such entities are called agents and can be better described as computer systems capable of independent actions to achieve their goals. They can also control physical devices, like a robot. Each agent can then perceive its environment, reason about it, and perform actions on it (Wooldridge, 2009). Both humans and software entities can fulfil these requirements. Whenever more than one agent are acting and interacting in the same environment, we have a MAS. The interaction might take place in different manners, for instance, by sharing a common resource or exchanging messages. As a possible outcome of this interaction, the agents in a MAS might decide to cooperate to bring about a particular state of affairs. For instance, whenever two buttons must be pushed simultaneously at two different locations. Therefore, mechanisms are needed to model how those software entities establish cooperation.

Coalition formation has long been an interesting topic of research, either for its practical applications or complexity issues. In order to increase the effectiveness of the agents in a MAS, we can organise them into coalitions, in which the agents collaborate with each other in order to achieve individual, common, or global (i.e., system level) goals (Wooldridge, 2009, Chapter 13). A coalition is usually a short-lived and goal-directed structure (Horling and Lesser, 2004). To guide the decision process of how to group agents, game theoretic constructs provide powerful tools. From this viewpoint, a coalition is a set of agents who may or may not work together. Each coalition has the ability to obtain a certain utility represented by a numeric value. Then we aim to maximise the overall value of all coalitions in the system. This set of coalitions is known as a coalition structure. Coalition formation approaches have been applied to several areas such as ride-sharing (Bistaffa et al., 2017b), cloud federation formation (Hadjres et al., 2020), collective electricity consumption shifting (Akasiadis and Chalkiadakis, 2017), E-commerce (Sukstrienwong, 2018), supply chain (Ben Jouida et al., 2017), and in task allocation problems (Mouradian et al., 2017; Mousavi et al., 2019).

However, not all problems can be efficiently solved using a single coalition structure. Consider for instance the Incident Command System (ICS) (Irwin, 1989), a popular system for disaster management. A part of this system establishes a modular hierarchy that adjusts itself according to the demands of a given disaster response operation. For instance, just one aspect of it requires resources (e.g., experts and equipment) to be distributed to a *group hierarchy*. A group hierarchy may be modelled by a sequence of CFGs, one per level. Nonetheless, picking the optimal solution of each level may not lead to a feasible overall solution: optimal coalition structures may not be compatible with one another. Therefore, we should avoid solving each game in isolation. Given this background, in this thesis, we investigate coalition formation problems that are interdependent. In particular, we focus on the interdependence among solutions (i.e., coalition structures) produced by each game individually. We shall look how

interdependence is addressed in coalition formation frameworks available in the literature and compare it with our approach. To demonstrate its generality, we aim to apply it to real-world problems. In particular, to disaster response which has a major social and economical impact on the communities affected by disaster events.

1.1 Main Contributions

Our main contributions are:

1. We propose a novel game named Sequential Characteristic-Function Game (SCFG), which aims to model the relationship between subsequent coalition structures computed by a corresponding sequence of CFGs. We show that our game can be used to model a variety of extended CFGs, like constrained and task-based CFGs. This contribution is partially published in (Krausburg et al., 2021b).

2. We integrate SCFG with Valuation Structure (VS) (Greco and Guzzo, 2017) to express game-specific constraints. The new form of the game is named Sequential CFGs induced by VSs (SEQVS) and is published in (Krausburg et al., 2021a). By using VS one can model special agents that are supposed to stay in different coalitions and also represent additional constraints in the form of an interaction graph among the agents in the system. We equip each CFG in our sequence of games with a corresponding VS to constrain the coalitions that can be formed on that particular game.

3. We further expand the VS framework to consider constraints on the size of coalitions. Greco and Guzzo (2017) pointed it out as an appealing extension to the VS framework. We claim that size constraints are an important feature for many applications. For instance, consider a ride-sharing problem (Bistaffa et al., 2017b) in which a car has a limited number of seats available. Thus, we propose Sized Valuation Structure (SVS) and Sequential CFGs induced by SVSs (SEQSVS) by combining VS with the size constraints proposed in (Rahwan et al., 2011).

4. We give an exact algorithm based on dynamic programming that solves SEQVS instances (Krausburg et al., 2021a). This new algorithm can be adapted to deal with any SCFG and SEQSVS instances which makes it the first exact algorithm for sequential CFGs. We empirically evaluate our algorithm and show that its running time is by several orders of magnitude better than a brute-force algorithm.

5. We show that the corresponding SCFG decision problem is **PSPACE**-complete.

6. We propose a hierarchical-clustering algorithm for solving SCFG instances named MC-Link; it is published in (Krausburg et al., 2021b). It is inspired by C-Link (Farinelli et al., 2016) a near-optimal algorithm for coalition structure generation problems in CFGs. We

show that MC-Link computes a solution for SCFG-based games faster than other approaches introduced in this thesis, however it lacks the completeness property.

7. We then investigate how Monte Carlo Tree Search (MCTS) can be applied to solve SCFG problems. We propose UCT-Seq which is based on the UTC algorithm (Kocsis and Szepesvári, 2006). We then evaluate it under general conditions and show that although (usually) slower than MC-Link, it can compute solutions of grater utility values.

8. We model the ICS hierarchy problem as a sequence of coalition formation problems by using SEQSVS. In particular, we model the River Roaring Flood scenario introduced in the ICS training course (U.S. Department of Agriculture, 2021) and compute a hierarchy of resources that can be deployed for the disaster response operation.

9. We make all experimental material gathered during this work available in the repository https://github.com/smart-pucrs/SCFG. Hopefully, the repository above will continue to be improved with different results, approaches and applications resulting from the use of the SCFG framework.

To the best of our knowledge, there are no other algorithms for solving SCFG instances except the ones proposed here, as it is the first time this sort of game has been addressed.

1.2 Thesis Outline

This thesis is organised as follows. In Chapter 2, we describe the background for understanding our work. Moreover, we discuss some related work on coalition formation literature as well as how one can express interdependence in similar fields. Chapter 3 introduces our formal framework to study the interdependence of CFGs, that is, Contribution 1. We discuss how it relates to the existing literature and provide useful extensions to it, namely, SEQVS (Contribution 2) and SEQSVS (Contribution 3). In Chapter 4, we investigate the cost of solving SCFG-based problems. We introduce an exact algorithm to solve SEQVS, Contribution 4, and prove that it is correct. Moreover, we provide a complexity result showing that SCFG is **PSPACE**-complete (Contribution 5). Given this bad news, in Chapter 5 we discuss heuristic approaches to address such a difficult problem. We propose two heuristic algorithms and extensively experiment with them in general scenarios. These are Contribution 6 and 7. Our next step, in Chapter 6, is to evaluate how a sequence of CFGs can be applied to model a real-world problem. Precisely, the problem of forming a hierarchy of resources based on the ICS framework. We experiment with two distinct manners of forming this hierarchy in a synthetic scenario leading to Contribution 8. Finally, in Section 7 we make some final considerations about our overall work and discuss the open venues in this exciting and difficult problem. In addition, in Appendix A we provide a glossary containing the main variables (and their corresponding meaning) used throughout this work.

2. BACKGROUND AND RELATED WORK

In this chapter we provide the background required for understanding our work. We shall introduce the frameworks commonly found in the coalition formation literature to model the problem of grouping agents to cooperate to achieve a given goal. The central topic is how to partition a set of agents into disjoint coalitions (i.e., groups of agents) and is usually approached with a game theoretical perspective called Coalitional Games. Also, we looked up in the coalition formation literature approaches that output a sequence of interrelated coalition structures. It turned out this is an unexplored topic and therefore, we could not perform any direct comparison. Instead, we discuss approaches that are close to ours. That is, some sort of interdependence appears in the frameworks of interest.

The first topic covered here is Multi-Agent System (MAS). We discuss what an agent is and how many agents in a given system organise themselves to achieve their goals. This is a broad topic, required though, to understand where our work is located in the coalition formation literature. In particular if one considers the techniques proposed in this work as part of an autonomous system. In Section 2.2, we present formal frameworks to model coalition formation problems. That is, how the agents get into groups. Then, we discuss in detail in Section 2.3 the Coalition Structure Generation (CSG) problem, which is the main focus of our work. Moreover, in this thesis we also propose an algorithm based on Monte Carlo Tree Search (MCTS) and the background on the topic is introduced in Section 2.4. Finally, in Section 2.5, we discuss some problems that require, on some level, interdependence among solutions.

2.1 Multi-Agent Systems

There is no universally accepted definition of what an agent is. Nonetheless, in this work we consider the definition proposed by Wooldridge. Wooldridge (2009, page 21) defines an agent as follows: “An agent is a computer system that is situated in some environment, and is capable of autonomous action in this environment in order to meet its delegated objective”. In other words, an agent observes its environment, reasons about it and produces an action that brings it towards its objectives. The process of observing the environment results in *perceptions* acquired by an agent. It can gather perceptions through sensors as well as from other agents. In the light of the acquired information, an agent reasons about how to act in order to accomplish its objectives, which is fundamental for autonomous agents and many approaches have been proposed to address this problem. Among them, the most successful ones are by means of Agent Programming Languages (APL); for instance, Jason (Bordini et al., 2007). The reasoning process results in actions performed by the agent. For instance, a robot controlled by an agent moving forward in open ground.

Another important definition is that of intelligent agent (Wooldridge, 2009). Intelligent agents have three main capabilities: (i) reactivity, they are able to perceive their environment and respond to the changes that occur in it; (ii) proactiveness, they take the initiative in order to satisfy their current objectives; and (iii) social ability, they are able to interact with other agents. In a MAS, we have more than one agent interacting in the same environment. These gathered agents can share the same goal, which makes themselves *cooperative* agents, or they can pursue their own agenda, making themselves *selfish* agents. The necessity of establishing cooperation and coordination among the agents in a MAS has triggered calls to develop techniques that consider intelligent agents. Therefore, in Section 2.1.1 we discuss how agents might get together and thus, achieve cooperation. In Section 2.1.2, we introduce approaches that focus on the organisation of a set of agents. Those approaches are able to express concepts that are usually found in real-world organisations, and therefore applications.

2.1.1 The Emergence of Cooperation

When cooperation is required, intelligent agents might organise themselves in order to achieve their goals. Such organisation guides how the members interact with each other. Horling and Lesser (2004) introduce several organisational paradigms, such as hierarchies, holarchies, coalitions, teams, congregations, societies, federations, markets and matrix organisations. In particular, we describe two of them: *coalitions* and *teams*; which are the most interesting for our purposes. Coalitions are goal-directed and short-lived; they are formed for accomplishing a purpose. After that, its members disband. In a coalition we may have agents that are both cooperative and self-interested. Within a coalition one member may act as a *leader*, although coalitions have no explicit hierarchy. Coalitions can be viewed as a single atomic entity, but *overlapping coalitions* can also exist. One key characteristic is that members from different coalitions do not coordinate their actions. Teams are similar to coalitions, what differs one from the other is the members' goals. A team consists of a number of cooperative agents that have agreed to work together toward a common goal (Horling and Lesser, 2004). That is, the team's members attempt to maximise the utility of the team itself. In general, each agent takes one or more roles required by the team's goal. An example of coalitions and a single team is depicted in Figure 2.1.

In multi-agent systems, there are a number of agents interacting in the same environment. These agents can execute their own tasks, but sometimes one task may require more effort than a single agent can offer. To address this issue, the agents can strategically and temporarily get together in order to improve their performance or to accomplish tasks they could not do alone. Coalition formation studies how we address the problem of how to group agents so as to maximise the reward they get for their efforts. When considering *coalitions*, we mean just a set of agents that will work together. In case an agent works alone, then we call it a *singleton*

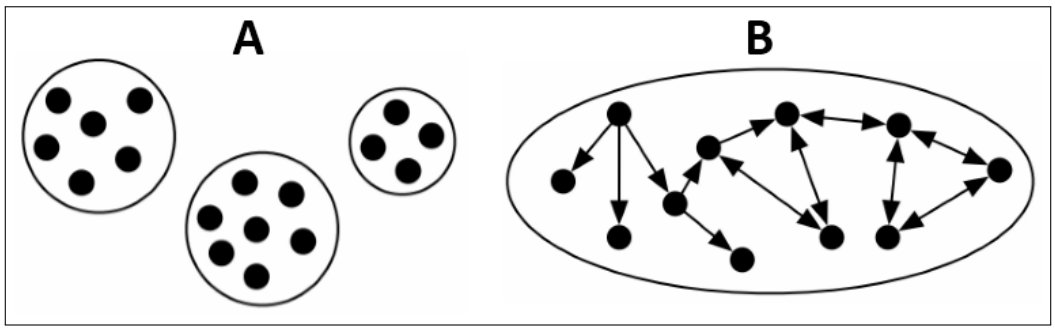


Figure 2.1: Coalition (A) and Team (B) organisational paradigms.
Source: Horling and Lesser (2004)

coalition. A partition of a set of agents into coalitions is called a *coalition structure.* In Section 2.2 we introduce a formal framework to model this problem.

Sandholm et al. (1999) defined the coalition formation process. They studied how the process of joining agents into groups occurs and what purpose those agents accomplish. They divided the coalition formation process into three activities.

1. **Coalition structure generation**: in this activity a set of agents is partitioned into disjoint coalitions. Here, we form coalitions in order to find a coalition structure that maximises the system overall performance (i.e., social welfare).

2. **Solving the optimisation problem of each coalition**: once we have defined the coalition structure, the coalitions in it begin to plan how to address their own problems and eventually come up with a plan to be carried out.

3. **Dividing the solution value among the agents**: this is related to the division of the coalition reward among the coalition members themselves right after the achievement of the coalition's goal. Usually, the outcome of the optimisation problem is used as reward[1].

Wooldridge (2009) referred to those activities as the cooperation life-cycle; we depict this cycle in Figure 2.2. It summarises how the agents may cooperate with each other to accomplish their desired goals. Given a set of agents, we put them together based on some criteria (e.g., distance to a given position); after that, the coalition starts to decide which agent will be responsible for which task, then they create a plan to accomplish the goal. Additionally, the coalition structure generation may be built endogenously or exogenously (Michalak et al., 2009). In endogenous coalition structure generation, the agents decide among themselves the best way to split the set of agents. In the exogenous approach, there is a system designer that puts the agents together.

In the remainder of this work we consider an *exogenous* approach. In many applications there exists indeed a system designer who plots what to do the with the agents available in the system. Of course, the criteria to form a coalition of agents might consider the agents' uniqueness. For instance, particular capabilities that complement each other as a requirement

[1]In order to investigate the division of the reward among the coalition members, we can use *game theory* concepts such as the *core* and the *shapley value* (Shoham and Leyton-Brown, 2009).

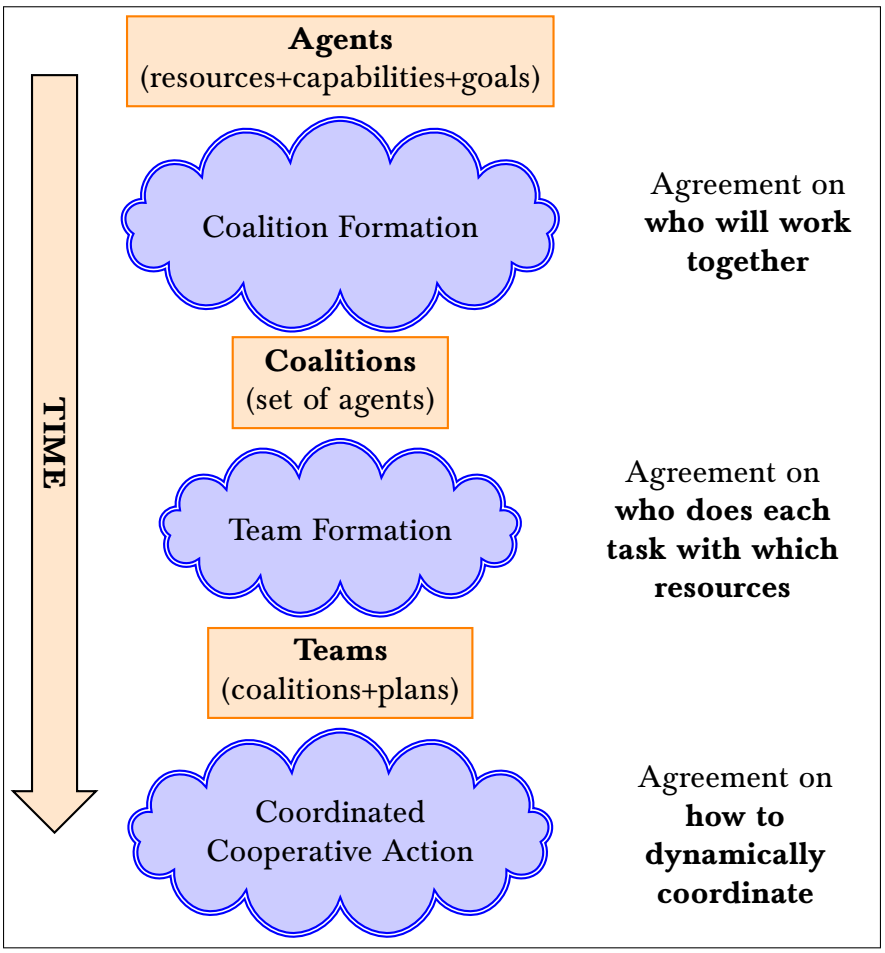


Figure 2.2: The cooperation life cycle.
Source: Wooldridge (2009)

to achieve a system goal. As a domain of interest, we refer to *disaster response operations*. In this sort of application there exists a planning section and a commander (or a unity of command) who is responsible for responding to a disaster event. The responders (i.e., trained resources) are managed by the assigned superiors to minimise the losses (in every sense). In the CSG viewpoint, the agents in a coalition formation process *represent* the set of trained resources. We discuss in depth this application in Chapter 6.

2.1.2 An Organisation of Agents

In multi-agent systems, the agents may also organise themselves in some sort of organisation and define norms that rule the agents' interaction. In addition to APLs, that define the behaviour of autonomous agents, some frameworks are available in the literature to help modelling coordination. The presence of a system designer is required to establish what is expected from this organisation of agents. Among those frameworks, we refer to Moise (Hübner et al., 2007, 2010) and Electronic Institutions (Sierra et al., 2004).

In the Electronic Institutions platform (Sierra et al., 2004), agents adopt roles and interact in order to achieve their individual and organisational goals; it is a social middleware. The interaction between agents occurs by the exchange of speech-act messages through agent group meeting (called scene) in which a role represents a standardised behaviour. When more than one scene are connected, a workflow is formed and called the performative structure. Agents can enter, leave, create, and destroy scenes. The ontology, common language, and knowledge representations are encapsulated in the dialogical framework. In this framework, the agents' behaviour are affected by obligation and prohibition rules.

Another approach, the MOISE organisation modelling language (Hübner et al., 2007), allows the specification of structural, functional, and normative dimensions of an organisation of agents. MOISE is designed to be organisational centred (i.e., the organisation exits a priori) in which the structural and function dimensions could be specified independently and the normative dimension (deontic approach) establishes the link between the two.

At the structural specification there are three levels: (i) the individual level defines the behaviour that an agent is responsible for when it adopts a role; (ii) the social level specifies the relationship (links) between the roles (either *authority*, *acquaintance*, or *communication*); and (iii) the collective level establishes the roles that compose a group. MOISE also checks when a group of agents is well-formed. For this purpose it checks the minimum and maximum number of agents that must participate in a group, the compatibility between any two roles (i.e., when one role can replace other within group), and the number of groups formed. Note that the collective level may be organised into hierarchies. For instance, in a soccer team of agents, the team as a whole is the main group and it could be split into two sub-groups, one for defence and one for attack.

The functional dimension describes how the organisation goals are to be achieved. This specification is done be means of *schemes* which are goal decomposition trees. In such models, the root is a global goal and it is decomposed into sub-goals that can be achieved by the agents. In order to fulfil a scheme, a plan, which is an instance of one goal decomposition tree, can be defined. Three operators are used for designing a plan: (i) sequence; (ii) choice; and (iii) parallelism. With a sequence operator, a goal can be achieved only after the achievement of the previous one. In the choice operator, a single goal in a the set of goals must be achieved. Finally, in the parallelism operator, all goals must be achieved, but they can be executed in parallel. Another important concept in MOISE is that of a *mission*. A mission is a set of goals from a scheme that is assigned to an agent to carry out. Once the agent is committed to a mission, it must fulfil the goals designated for that mission.

At the normative dimension, it is defined what is permitted or obliged. The roles played by the agents are connected to the missions. When a mission is permitted to a given role, an agent that adopts that role can commit to that mission. In an obligation statement, an agent playing the role must commit to that mission. The organisational specification is stored in

a XML file and the agents can reason about the organisation and modify it at run-time (Hübner et al., 2007).

Considering disaster response operations, the understanding of roles (e.g., from the structural specification) is important to develop rapid actions in emergency teams. Teammates need to be aware of their responsibilities (individual level) as well as the each other's responsibilities (team level) (Power, 2018). This knowledge leads to implicit coordination between the members of a team in which no communication is needed to complete a particular task. For instance, if during a disaster response operation a victim requires medical assistance, an agent in the team adopting the role of paramedic is expected to be the most skilled one to deal with that task at hand.

2.2 Formal Frameworks for Modelling Coalition Formation

We will formally describe the coalition formation process as a Characteristic-Function Game (CFG) under the perspective of *Cooperative Game Theory* (also called *Coalitional Games*) (Shoham and Leyton-Brown, 2009, Chapter12). In a 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). We use $\boldsymbol{\mathcal{C}}^A$ to denote the set of all coalitions over A.

Definition 1 (Coalition Structure CS). Given a set of agents $A = \{a_1, \ldots, a_n\}$ a coalition structure CS is a partition of A into coalitions. That is, for all $C, C' \in CS$, $C \neq C'$, $C \cap C' = \emptyset$ and $\bigcup_{C \in CS} = A$. We use $\boldsymbol{\mathcal{CS}}^A$ to denote to the set of all coalition structures over A.

To evaluate how good each coalition is, one quantifies it using a valuation function designed specifically for each application of interest.

Definition 2 (Characteristic Function $v(.)$). Given a set of agents $A = \{a_1, \ldots, a_n\}$ a *characteristic function* v is a valuation function $v : 2^A \to \mathbb{R}$. That is, v maps every coalition $C \in \boldsymbol{\mathcal{C}}^A$ onto a real value.

A CFG Γ is then a pair $\langle A, v \rangle$. A value for a coalition structure is computed as follows: $V(CS) = \sum_{C \in CS} v(C)$. The goal is to a CS that

$$CS^* = \arg \max_{CS \in \boldsymbol{\mathcal{CS}}^A} V(CS).$$

We introduce in Example 1 an instance of this problem.

Example 1. Consider a set of agents $A = \{a_1, a_2, a_3\}$. The feasible coalitions are: $\{a_1\}$, $\{a_2\}$, $\{a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_2, a_3\}$, and $\{a_1, a_2, a_3\}$. Therefore, the space of all partitions over A is $\boldsymbol{\mathcal{CS}}^A = \{\{\{a_1\}, \{a_2\}, \{a_3\}\}, \{\{a_1\}, \{a_2, a_3\}\}, \{\{a_2\}, \{a_1, a_3\}\}, \{\{a_3\}, \{a_1, a_2\}\}, \{\{a_1, a_2, a_3\}\}\}$.

In case of CFG with transferable utilities, the agents agree on how to distribute the reward received by the coalition (Shoham and Leyton-Brown, 2009, Section 12.1). The outcome

of a coalitional game is then given by a pair $(CS, \boldsymbol{x})$ where CS is a coalition structure and $\boldsymbol{x}$ is a ***payoff vector*** which distributes the value of each coalition among its members. The payoff vector is defined as $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Let k be the size of a coalition structure CS, $|CS| = k$. Two conditions hold for the payoff vector:

(i) for all $a_i \in A, x_i \geq 0$; and

(ii) $\sum_{a_i \in C_j} x_i \leq v(C_j)$ for any $1 \leq j \leq k$.

An ***imputation*** is when a payoff vector satisfies two additional conditions:

(i) it is ***efficient*** such that $\sum_{a_i \in C_j} x_i = v(C_j)$ for any $1 \leq j \leq k$; and

(ii) it satisfies the ***individual rationality***, that is $x_i \geq v(\{a_i\})$ for all $a_i \in A$.

This is the classic form of the game and to the interested reader, we refer to (Rahwan et al., 2015) for a survey on this topic. Alternatively, some extensions to this problem have been proposed in the coalition formation literature and we introduce them in the sections below.

2.2.1 Coalition Formation with Overlaps

Apart from the classical viewpoint on coalition formation, one might consider coalitions with overlaps in which the constraint of disjoint coalitions in a coalition structure is dropped. That is, any two coalitions may share members. An agent that participates in more than one coalition contributes ***something***[2] to the coalitions it is a member of. This sort of game makes no assumptions on how an agent acts to achieve the goals of all coalitions that it participates. Instead, it focus only on how to determine to which and how many coalitions an agent will contribute.

This idea is formalised as Overlapping Coalition Formation (OCF) games (Chalkiadakis et al., 2008) in which one considers only resources available to the agents. Formally a OCF game is a tuple $\langle A, v \rangle$, where $A = \{a_1, \ldots, a_n\}$ is the set of agents and $v : [0,1]^n \to \mathbb{R}$ is the characteristic function. The authors no longer use the term ***coalition***, instead, they define a ***partial coalition*** as a vector $\boldsymbol{c} = (r_1, \ldots, r_i, \ldots, r_n)$ where r_i is the amount of resource agent a_i contributes to that partial coalition $\boldsymbol{c}$. Note that $r_i \in [0,1]$ (without loss of generality) and $r_i = 0$ means agent a_i is not a member of the coalition. In order to retrieve only the agents that contribute something to a given coalition, the authors define the support of a partial coalition $supp(\boldsymbol{c}) = \{a_i \in A \mid r_i \neq 0\}$.

Definition 3. (Coalition Structure with Overlaps CS (Chalkiadakis et al., 2008)) Given a set of agents $A = \{a_1, \ldots, a_n\}$, a coalition structure CS is defined as a list of partial coalitions $CS = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k)$ (alternatively a matrix $n \times k$ (Zick and Elkind, 2011)), such that:

[2]The notion of contribution is addressed here in a abstract way, but it could be money, time, or any other feature of real-world domains.

(i) $\boldsymbol{c}_j \in [0,1]^n$;

(ii) $supp(\boldsymbol{c}_j) \subseteq A$ for all $j = 1, \ldots, k$; and

(iii) $\sum_{j=1}^{k} r_i^j \leq 1$ for all $a_i \in A$.

It is important to note also that an agent is not obligated to allocate all of its resources to the coalitions it participates. Moreover, given a partial coalition $\boldsymbol{c}$ where its support is $supp(\boldsymbol{c}) = C$, the agents from C could form various overlapping coalitions (i.e., many coalitions with the same support) (Chalkiadakis et al., 2010).

Some variations of overlapping games were proposed in the literature, among them is a discrete version of OCF (Zick et al., 2012). In this setting, a weight vector $\boldsymbol{w}$ is defined and each agent a_i has a ***positive integer*** weight $w_i \in \boldsymbol{w}$ which can be distributed to different coalitions. Moreover, one could consider coalitions that are bounded on the number of agents. In k-bounded OCF (Zick et al., 2012) there is a bound k on the size of every coalition. Thus, the assigned value to a partial coalition $\boldsymbol{c}$ is set to zero, $v(\boldsymbol{c}) = 0$, if the number of agents that contribute to $\boldsymbol{c}$ is greater than k, $|supp(\boldsymbol{c})| > k$. Chalkiadakis et al. (2008) introduce ***U-finite*** games in which a coalition structure is bound in size by U. In their setting, a game is said to be ***U-finite*** if for any outcome $(CS, \boldsymbol{x})$ such that $|CS| > U$ and $\boldsymbol{x} \in I(CS)$ where $I(CS)$ is the set of imputations over CS; there exists a $(CS', \boldsymbol{y})$ such that $|CS'| \leq U$ and $\boldsymbol{y} \in I(CS')$ and $p_i(CS, \boldsymbol{x}) \leq p_i(CS', \boldsymbol{y})$ where p_i is the payoff for each agent $i = 1, \ldots, n$. Note the ***U-value*** could be designed as a function on the number of agents; if $U(n) < \infty$. The size of a coalition structure may be either constrained on $v(.)$ or on the allowed outcomes; $|CS| \leq U$.

Generally, the collaboration between agents in a MAS is goal oriented and many studies in the literature have addressed this sort of problem using ***task allocation*** techniques. One of the first studies concerning overlapping coalitions was introduced in (Shehory and Kraus, 1998), which focused on tasks. In their approach, the agents have capabilities that quantifies their ability to perform specific actions. The problem then becomes task allocation via coalition formation in which tasks are organised in a precedence order. Precedence here means that given any two tasks $t_i, t_j \in T$, where T is a set of tasks, if there exists a precedence relation $t_i \prec t_j$, t_i must be accomplished before t_j. It is important to note that each agent has a number of capabilities and each task requires a number of capabilities. Therefore, a coalition can only achieve a given task iff it possesses all capabilities required by the task. The problem the authors aim to solve is how to assign tasks $t_i \in T$ to coalitions of agents $C_i \subseteq A$ such that $\sum_i v(C_i)$ is maximised and the precedence order is holds. Similar setting is also proposed in (Lin and Hu, 2007; Xu and Li, 2008; Zhang et al., 2010; Chalkiadakis et al., 2010; Zhang et al., 2011).

Rahwan et al. (2013) also studied the problem regarding overlapping coalitions and tasks. Their setting is similar to discrete OCF (Zick et al., 2012), however adding tasks. The number of coalitions an agent may be part of is limited by the number of resources it has. If the maximum number of resources each agent has is one, then we have the classic CFG. We could also set the maximum number of resources to 2^{n-1}, then an agent could participate in all

coalitions. One could also consider a game with overlaps in which agents are geographically dispersed. The approach proposed by Zhang et al. (2017b) is built on the top of OCF to consider distributed agents. The main difference is regarding the coalitions, an agent forms a coalition with the agents within its communication range.

The overlapping problem was also addressed in bargain models (Nguyen, 2015). A bargain model is defined in terms of rounds (or periods) in which agents execute some procedures. For instance, in each round an agent is selected to make a coalition proposal that could be accepted or rejected by the others. The main idea behind this setting is to distribute the decision on the coalition formed and the individual profits. At each period, a feasible coalition and a proposer from that coalition are chosen. Agents are asked to agree on the division of surplus, if all agents that belong to that coalition agree, then the coalition forms. Instead of resources, tasks, etc., this setting only focuses on the final value of the characteristic function. Doing so, agents can join as many coalitions as they desire (as long as the game is running).

2.2.2 Coalition Formation with Externalities

In coalitional games, a coalition is not influenced by the other coalitions that belong to the same CS. In practical terms, the valuation function assigns a value to any coalition regardless of the CS it belongs. In the opposite direction, Partition Function Games (PFG) take into account as well the coalitions that belong to the same CS. This is also called games with externalities. Therefore, the value of each coalition depends on the chosen coalition structure (Thrall and Lucas, 1963). In general, there are two ways in which a coalition is affected by others. A positive externality in which the value of a coalition might increase (e.g., two coalitions with overlapping goals, as soon as the goal is satisfied, both coalitions are rewarded). On the other hand, a coalition can also be negatively affected by others (e.g., two coalitions that share the same resource) (Chalkiadakis et al., 2011).

We follow Chalkiadakis et al. (2011, Section 5.2) to formally define a PFG. In a PFG, one assumes as input a tuple $\Gamma_P = \langle A, u \rangle$, where $A = \{a_1, \ldots, a_n\}$ is a set of agents and u is a partition function (explained below). Note that both PFG and CFG share the same underlying structure and we only change the way we interpret the valuation function. PFG introduces the concept of *embedded coalition*, which combines a coalition with the CS to which it belongs.

Definition 4. (Embedded Coalition EC) An embedded coalition EC is a pair $EC = (C, CS)$, where $C \subseteq A$, $CS \in \boldsymbol{\mathcal{CS}}^A$, and $C \in CS$.

We use $\boldsymbol{\mathcal{EC}}^A$ to denote the set of all embedded coalitions over A.

The coalition structure follows the same definition as in the classic view of the game (see Definition 1). The partition function u of the game is then defined as $u : \boldsymbol{\mathcal{EC}}^A \to \mathbb{R}$. The

overall goal is to find a coalition structure that

$$\max_{CS \in \boldsymbol{\mathcal{CS}}^A} \sum_{C \in CS} u(C, CS).$$

Note that the set of all coalition structures $\boldsymbol{\mathcal{CS}}^A$ is the same in both CFG and PFG.

In order to enumerate all the embedded coalitions, one needs $\mathcal{O}(n^n)$ space which is not tractable (Rahwan et al., 2015). In this direction, some concise representations have been proposed for PFG, for instance, an extension to MC-nets (Ieong and Shoham, 2005). An MC-rule is composed of $pattern \rightarrow value$, where pattern indicates the absence/presence of a sub-set of agents in a coalition C. Then, the value of C is computed by summing up the $value$ term in all applicable MC-rules over it. In a PFG setting a MC-rule is extended in order to state the desired coalition structure such as $P_O|P_1, \ldots, P_k \rightarrow value$, where P_O means the expected coalition structure and $P_1, \ldots, P_k$ state the presence or absence of agents in a coalition (Michalak et al., 2010a). Alternatively, partition decision trees can be used to represent PFG (Skibski et al., 2015, 2020a). In a rooted directed tree, non-leaf nodes are agents, leaf nodes are labelled as payoff vectors and edges indicate the coalitions formed by the agents. The game is then modelled by many partition decision trees. The authors show that this representation enables polynomial computation of almost all extensions to the Shapley value for PFG. In addition, different methods for solving coalition structure generation instances under partition decision tree representations were also proposed in the literature (Zha et al., 2017).

2.2.3 The Role of Constraints

In practice many applications do not need to take into account all possible combinations of agents in order to output a suitable coalition structure. For example, sometimes there are constraints that narrow down the number of different coalitions allowed. These constraints might occur for many reasons, for instance, due to short-range communication, trust relations, or physical constraints (Voice et al., 2012b). The question is then how to model them.

Rahwan et al. (2011) introduced the first systematic study of Constrained Coalition Formation (CCF). In this setting not all coalition structures are feasible (i.e., can be formed). A CCF game Γ_C in its general form is given by a triple $\langle A, \boldsymbol{\mathcal{CS}}_{cst}, v \rangle$, where A is the set of agents, $\boldsymbol{\mathcal{CS}}_{cst}$ is the set of allowed coalition structures and v is the characteristic function. The idea is the same as in a CFG, however now the solution is one of those CSs given as input. Moreover, the authors noted that in some settings the constraints can be reduced to the ones applied only to coalitions instead of CSs. For instance, two agents $a_1, a_2 \in A$ must belong to different coalitions. This constraint must hold in all feasible CS. Let $\boldsymbol{\mathcal{C}}_{cst}$ be the set of all feasible coalitions. Then, $\boldsymbol{\mathcal{CS}}_{cst} = \{CS \in \boldsymbol{\mathcal{CS}}^A \mid CS \subseteq \boldsymbol{\mathcal{C}}_{cst}\}$; that is, the set of all coalition structures that are *locally constrained*.

Although CCF games are expressive (i.e., one can state only the CSs of interest), it might be intractable to list out all feasible coalition structures. A modified version of the locally-constrained CCF problem was proposed to address this; it is called the basic CCF model (Rahwan et al., 2011). The constraints are now expressed in the form of: (i) sizes of the coalitions; and (ii) subsets of agents whose presence in any coalition is viewed as desirable[3] or prohibited. The size constraint $\mathcal{S} \subseteq \mathbb{N}$ is the set of permitted coalition sizes. The set of desirable constraints (called positive constraints), which are denoted as $\mathcal{P} \subseteq 2^A$ such that a coalition C satisfies a constraint $P \in \mathcal{P}$ if $P \subseteq C$. In the same way, the set of prohibited constraints (negative constraints), which are denoted as $\mathcal{N} \subseteq 2^A$ such that C satisfies a constraint $N \in \mathcal{N}$ if $N \nsubseteq C$. A coalition is considered feasible if it has at least one positive constraint, no negative constraint, and the size of the coalition is permitted. Formally, a coalition $C \subseteq A$ is feasible, if: (i) $P \subseteq C$ for some $P \in \mathcal{P}$; (ii) $N \nsubseteq C$ for all $N \in \mathcal{N}$; and (iii) $|C| \in \mathcal{S}$. A coalition structure is only feasible if it contains only feasible coalitions. Therefore, a basic CCF model is given by a tuple $\langle A, \mathcal{P}, \mathcal{N}, \mathcal{S}, v \rangle$.

Another sort of modelling constraints is to model the relationship between any two agents in the system using a graph. The underlying model is given by an undirected graph $G = (A, E)$, where $A = \{a_1, \ldots, a_n\}$ is a set of agents and $E \subseteq A \times A$ is a set of edges connecting any two agents. A coalition C is allowed to form iff the induced sub-graph of C over G is connected. Similarly to a classic CFG game, a characteristic function $v : 2^A \to \mathbb{R}$ is assumed as input to evaluate the coalitions; and hence the coalition structures. In graph-based coalition formation some properties can be drawn to help finding a solution for the problem. For instance, if a valuation function v follows the independence of disconnected members property given below.

Definition 5 (Independence of Disconnected Members (IDM)). Given a graph $G = (A, E)$, a function $v : 2^A \to \mathbb{R}$ is independent of disconnected members if for all $a_i, a_j \in A$ with $(a_i, a_j) \notin E$, and coalition C with $a_i, a_j \notin C$,

$$v(C \cup \{a_i\}) - v(C) = v(C \cup \{a_i, a_j\}) - v(C \cup \{a_j\}).$$

That property states that a_i and a_j do not have any synergy, hence placing both in the same coalition does not contribute to improving the marginal contribution of a_i to the coalition. Interestingly, any game in which the valuation function is given by the sum of edges mapped onto weights (i.e., real numbers) has the IDM property (Voice et al., 2012a). That is, suppose that the graph G is weighted. That is, $G = (A, E, w)$, where $w : E \to \mathbb{R}$ is the weight function. Then the valuation function is given by $v(C) = \sum_{a_i, a_j \in C} w(a_i, a_j)$ and follows the IDM property.

The rule-based (i.e., basic CCF model) and graph-based constrained games do not represent the same set of problems. To show why, we reproduce an example introduced by Greco and Guzzo (2017). Assume a set of agents $A = \{a_1, a_2, a_3\}$.

[3]Desirable here refers to a subset of agents that ***must*** be in the same coalition. A feasible coalition must fulfil at least one of the desirable constrains.

Example 2 (Limitation of Graph-Constrained Games). Suppose we want to model a constraint in which all coalitions are feasible but the ones in which a_1 and a_3 are together. That is, the coalitions $\{a_1, a_3\}$, and $\{a_1, a_2, a_3\}$ are unfeasible. If we try to model this problem as a graph-based game $G = \langle \{a_1, a_2, a_3\}, E \rangle$, there must not exist a path from a_1 to a_3 in E. However, one cannot model that as the edge (a_1, a_2) is required for the feasible coalition $\{a_1, a_2\}$, and similarly, the edge (a_2, a_3) for the feasible coalition $\{a_2, a_3\}$. Therefore, the coalition $\{a_1, a_2, a_3\}$ is feasible, which should not be the case.

This problem can easily be modelled by a basic CCF assuming the following tuple $\langle \{a_1, a_2, a_3\}, \mathcal{P}, \mathcal{N}, \mathcal{S}, v \rangle$, where $\mathcal{P} = \{\{a_1\}, \{a_2\}, \{a_3\}\}$, $\mathcal{N} = \{\{a_1, a_3\}\}$, and $\mathcal{S} = \{1, 2, 3\}$.

Example 3 (Limitation of basic CCF Games). Suppose now that we want to model a constraint in which all coalitions are feasible but the one in which a_1 and a_3 are ***alone*** in the same coalition. That is, the coalition $\{a_1, a_3\}$ is unfeasible. If we try to model this problem with a basic CCF using $\mathcal{N} = \{\{a_1, a_3\}\}$, then the coalition $\{a_1, a_2, a_3\}$ becomes unfeasible; it should not be case. If we let the negative constraints empty, $\mathcal{N} = \emptyset$, then we need to model the intended constraint in the set of positive constraints. However, if $\{a_1\} \in \mathcal{P}$, then $\{a_1, a_3\}$ is feasible; it should not be the case. If $\{a_1\} \notin \mathcal{P}$, then $\{a_1\}$ is unfeasible; it should not be the case.

This problem can easily be modelled by a graph-based game. We just have to make sure $(a_1, a_3) \notin E$ and one can reach a_3 from a_1. Therefore, $G = \langle \{a_1, a_2, a_3\}, \{\{a_1, a_2\}, \{a_2, a_3\}\} \rangle$.

Aware of this limitation, Greco and Guzzo (2017) proposed the concept of Valuation Structures (VS) to combine both worlds. A VS is induced over a CFG, that is, it modifies how the original game is played. The general idea is that some agents are incompatible with one another, called ***pivotal agents***, in such a way that they must stay in different coalitions; a negative constraint over a coalition. Moreover, they assume the relation between any two agents is modelled using an undirected graph; called ***interaction graph***.

Definition 6 (Valuation Structure σ (Greco and Guzzo, 2017)). Given a CFG $\Gamma = \langle A, v \rangle$, a valuation structure is a tuple $\sigma = \langle G, S, \alpha, \beta, x, y \rangle$ induced over Γ, Γ^σ, where:

- $G = (A, E)$ is a undirected graph, where $E \subseteq A \times A$;
- $S \subseteq A$ is a set of pivotal agents;
- $\alpha, \beta : S \to \mathbb{R}$ are mappings assigning to pivotal agents a real value; and
- $x, y \in \mathbb{R}$ are constants to modify the value of any coalition that does not contain pivotal agents.

The combination of a graph with a set of pivotal agents poses the constraints of the game. The semantics of this game state that a coalition C is allowed iff: (i) the induced sub-graph of C over G is connected; and (ii) $|C \cap S| \leq 1$. The remaining variables are used to

modify the given valuation function v. This is done using Equation 2.1.

$$\boldsymbol{val}_{\sigma}(v, C) = \begin{cases} \alpha(a_i) \times v(C) + \beta(a_i) & \text{if } \{a_i\} = C \cap S \\ x \times v(C) + y & \text{otherwise} \end{cases} \tag{2.1}$$

Pivotal agents may have important characteristics that make them different from the remaining agents. For this reason, the authors provide a way to manipulate the valuation function to take that into account.

Another way to consider negative constraints over a graph game is signed-graph games (Skibski et al., 2020b). In this modelling, a game is a tuple $\langle A, v, E_+, E_- \rangle$, where $\langle A, v \rangle$ is a CFG and $G^{\pm} = \langle A, E_+, E_- \rangle$ is a signed graph. The edges in E_+ have the same semantic as regular edges in an interaction graph, that is, the induced coalition must be connected. On the other hand, an edge $(a_i, a_j) \in E_-$ indicates that agents a_i and a_j cannot be placed in the same coalition. One can see that the set of pivotal agents can also be represented in this game by having $S \times S \subseteq E_-$.

Another approach view all coalitions as feasible, nonetheless the coalition structure size is bounded by a constant k (Skibski et al., 2016); it goes on the opposite direction as locally-constrained games (Rahwan et al., 2011). The inspiration for this sort of game is that many physical and organisational restrictions are in the coalition structures rather than the coalitions. For instance, in a organisation, any coalition is feasible, but no coalition structure may contain more coalitions than the number of leaders available. Moreover, Skibski et al. (2016) investigate this setting in the context of PFG in which the value of a coalition is affected by the CS it is member of. Similarly, Präntare et al. (2021) consider a k number of tasks and then aim to form a coalition structure to carry out the k-tasks; therefore, the size of a CS is bounded by k. However, the authors approach this problem under the perspective of a permutation problem. Formally, the game is given by a triple $\langle A, T, v \rangle$, where $A = \{a_1, \ldots, a_n\}$ is a set of agents, $T = \langle t_1, \ldots, t_k \rangle$ is an ordered set of tasks, and $v : 2^A \times T \to \mathbb{R}$. Thus, one produces an ordered coalition structure $CS = \langle C_1, \ldots, C_k \rangle$ according to the set of tasks T. That is, task t_i is assigned to coalition C_i for all $1 \leq i \leq k$. In case t_i is not allocated to any coalition, then $\emptyset$ appears at position i in CS.

2.3 Algorithms for the Coalition Structure Generation Problem

Given the frameworks introduced above, many algorithms have been proposed to solve the coalition structure generation problem. That is, the problem of finding a coalition structure that maximises the joint value of its coalitions given a valuation function. First, we introduce graphical (classic) ways of representing the search space for this problem. Then, we introduce some well-known algorithms that solve a CFG. We explain in details C-Link, which is used in the remainder of this work. Our next step is to introduce some algorithms that take constraints

into account. In particular CFSS, which also plays an important role in the remainder of this work.

2.3.1 Search Space Representation

To find an optimal coalition structure CS^*, one might firstly organise the search space in a representation that guides the search. Using this representation one should be able to exhaustively enumerate the set $\boldsymbol{\mathcal{CS}}^A$ without repeating any CS. The first representation was proposed by Sandholm et al. (1999) and it is called *coalition structure graph*. The search space is viewed as a graph in which each vertex correspond to a coalition structure. The graph is divided into levels, $L_1, \ldots, L_n$. Each CS in level L_i contains exactly i coalitions. For instance, consider Figure 2.3 which depicts a coalition structure graph for four agents. One edge links any two vertices iff: (i) they belong to two consecutive levels L_i and L_{i-1}; (ii) a coalition structure at level L_{i-1} is formed by performing a pairwise union operation on coalitions in a CS at level L_i.

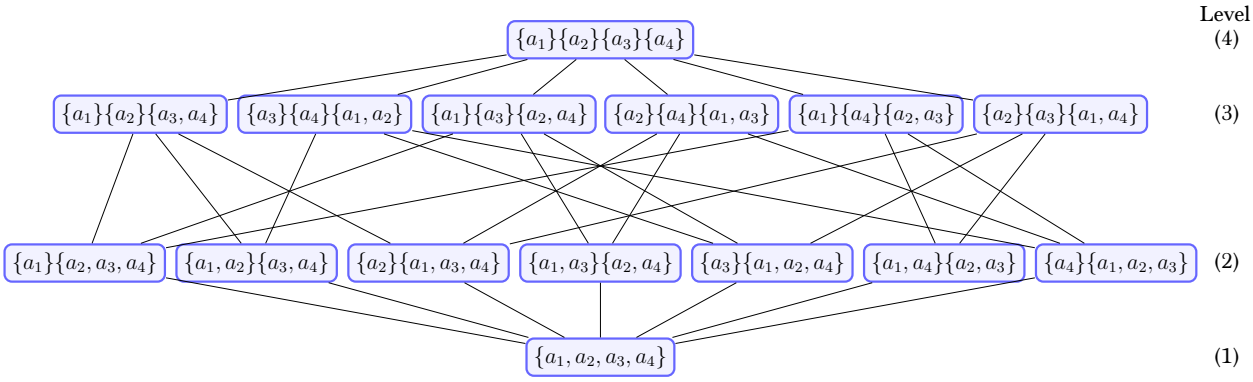


Figure 2.3: The coalition structure graph for four agents.
Source: Sandholm et al. (1999)

Another approach for representing the search space was proposed by Rahwan et al. (2007b) and is based on the *coalition structure configuration*. A configuration is a grouping of coalition structures according to the size of their coalitions. For instance, both coalition structures $\{\{a_1\}, \{a_2, a_3\}\}$ and $\{\{a_3\}, \{a_1, a_2\}\}$ follow the configuration $\{1, 2\}$. This encoding is then represented by an integer partition graph (Rahwan and Jennings, 2008a) in which each vertex correspond to a configuration (i.e., a sub-space). Levels are also considered in this representation in which each level L_s represents a sub-space containing s coalitions per coalition structure. For instance, level L_2 in Figure 2.4 contains only CS of size 2 (vertices $\{3, 1\}$ and $\{2, 2\}$). Moreover, the sub-space $\{3, 1\}$ contains all CSs that have one coalition with three members and a singleton coalition. An edge connects any two vertices if a pairwise sum operation on a configuration results in a integer of the destination vertex. For instance, configuration $\{2, 2\}$, in Figure 2.4, can reach only configuration $\{2 + 2\}$.

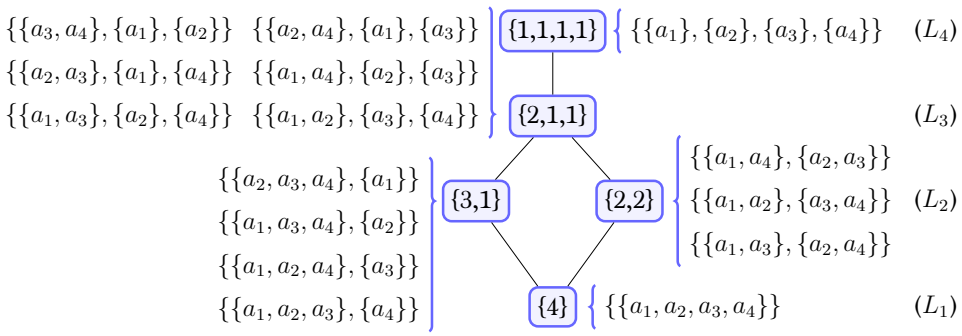


Figure 2.4: The integer partition graph for four agents.
Source: Rahwan and Jennings (2008a)

2.3.2 Algorithms for Coalition Structure Generation

Rahwan and Jennings (2008b) divide the algorithms that solve the coalition structure generation problem into classes. They are: (i) based on dynamic programming; (ii) heuristic; and (iii) anytime optimal algorithms. A number of algorithms have been proposed in each of these classes and we discuss some of them below.

In anytime optimal algorithms an initial solution, usually guaranteed to be within a bound from the optimum, is generated. After that, the quality of the solution is improved establishing progressively better bounds as the search goes on, until an optimal solution is found (Rahwan and Jennings, 2008b). Rahwan et al. (2007a, 2009b) proposed the IP algorithm, designed to follow the properties of optimality, ability to prune, discrimination, anytime, and worst-case guarantee. The space of possible coalition structures is represented as integer-partition subspaces. The amount of search needed is reduced with a branch-and-bound technique. A minimum and a maximum bound are evaluated for each subspace. These bounds are used to speed up the search in two ways: (i) if the minimum bound of a subspace is greater than the maximum of another subspace, the latter subspace must not be searched; and (ii) if the current best solution is greater than the maximum of a subspace, this subspace is cut off. The IP algorithm is guaranteed to find an optimal solution when run to completion (Rahwan et al., 2009b).

A distributed version of IP was introduced in (Michalak et al., 2010b), the first decentralised algorithm for solving the CSG problem. This algorithm uses filters on the input (reducing the communication) and applies the techniques from (Rahwan and Jennings, 2007) to distribute the calculation of possible coalitions among the agents. Other algorithms developed to find an optimal coalition structures are (Sandholm et al., 1999; Dang and Jennings, 2004). Sandholm et al. (1999) introduce an algorithm that searches through the first two levels and the last level of the coalition structure graph. After that, a breadth-first search is used and it continues as long as there is time left or until the entire graph has been searched. Dang and Jennings (2004) improve on the algorithm proposed by Sandholm et al. (1999). It searches through the same three levels as the algorithm in (Sandholm et al., 1999) does. After that, in-

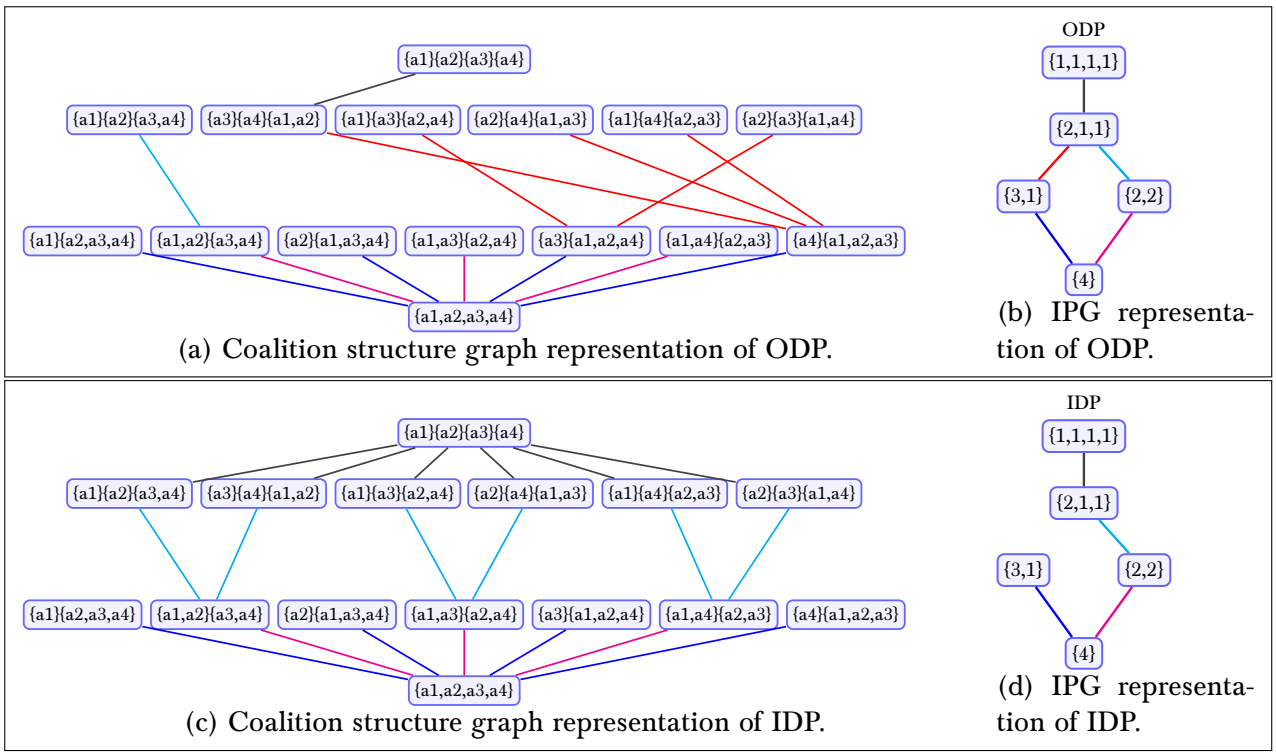
stead of searching for all the remaining levels, it searches some subsets of each remaining levels based on the coalition structure cardinality within the level.

In the dynamic programming class of algorithms the main advantage is the lower time complexity. Some algorithms are guaranteed to find an optimal solution in $\mathcal{O}(3^n)$ for n agents. The drawbacks are: (i) it requires more memory; and (ii) it needs to complete the execution of the algorithm in order to output a solution. The first algorithm that used the dynamic programming paradigm is DP (Yeh, 1986). It is designed to solve the ***complete set partitioning*** problem. The time complexity and space complexity were shown to be $\mathcal{O}(3^m)$, where m is the number of rows of an auxiliary table necessary to represent the number of elements in binary notation. For instance, for seven elements m is equal to three. DP iterates over all coalitions of size 1, then coalitions of size 2, and so on, up to the grand coalition. When the size of a coalition is greater than 1, the algorithm compares the value of the coalition with the sum of the previous coalitions' sizes that compose this bigger coalition. For instance, DP evaluates the coalition $\{a_2, a_3\}$ with the sum of coalitions $\{a_2\}$ plus $\{a_3\}$ and it stores the best way of splitting the coalition.

Rahwan and Jennings (2008a) showed that the DP algorithm performs many redundant operations. They introduced the ODP algorithm which avoids approximately two thirds of the operations of DP (Rahwan et al., 2015). They show these redundant operations when DP is evaluated on the coalition structure graph: an optimal coalition structure can be reached from different paths. ODP identifies ***a priori*** what are the relevant movements to be executed without any need for extra memory requirements. If there exists in the coalition structure graph one path from any node to any other node in the graph, ODP eventually finds an optimal coalition structure.

The fastest version of ODP was proposed by Michalak et al. (2015). It combines both ODP and IP algorithms, resulting in the ODP-IP algorithm. This hybrid algorithm avoids the limitations of its two components and improves on the advantages of each. First the authors show that the movements made by DP in the coalition structure graph can also be visualised on the integer partition graph, which is used by IP. When DP makes a movement from coalition structure node $\{\{a_1\}, \{a_2, a_3, a_4\}\}$ to $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$, it also moves from subspace node $\{3, 1\}$ to $\{2, 1, 1\}$ in an integer partition graph as depicted in Figure 2.5. One edge in the integer partition graph is represented by some edges in the coalition structure graph. ODP can avoid only some edges from a coalition structure graph of a given size which means the edge that links two subspaces in the integer representation cannot be cut off. In order to address this problem, Michalak et al. (2015) proposed a size-based version of ODP which evaluates all or none of the movements that link two nodes in an integer partition graph; it is called sb-ODP or IDP. Combining IP and sb-ODP results in ODP-IP, which is an anytime algorithm and its time complexity is $\mathcal{O}(3^n)$.

Another approach uses the inclusion-exclusion principle (Björklund et al., 2009) to solve the set partitioning problem. In such an approach, the information is encoded into ex-



(a) Coalition structure graph representation of ODP.

(b) IPG representation of ODP.

(c) Coalition structure graph representation of IDP.

(d) IPG representation of IDP.

Figure 2.5: Comparison of search space representation between coalition structure graph and integer partition graph in an example of four agents for ODP and IDP algorithms.

tremely large numbers and has a theoretical time complexity $\mathcal{O}(2^n)$. However, it does not take into account the time required to manipulate such huge numbers. In practice the algorithm runs in $\mathcal{O}(6^n)$ (Rahwan et al., 2015).

When it comes to PFGs, one approach is to prune parts of the search space whilst computing an optimal solution. For instance, Michalak et al. (2008) propose the use of bounds on the value of a coalition (regardless of its coalition structure) in games with positive and negative externalities with super- and sub-additivity valuations. Rahwan et al. (2009a) determines bounds based on the partition of a coalition instead of the coalition itself. Moreover the bounds developed can be applied to any positive and negative externality classes.

Metaheuristic algorithms are used when the number of agents is very large, hence the problem becomes harder to solve. This kind of algorithm is usually fast; however, it may not provide any guarantees on the quality of its solutions. Shehory and Kraus (1998) proposed some greedy distributed anytime algorithms to the set partitioning and set covering problems. First, the algorithms compute the value of all possible coalitions. Then a distributed greedy procedure is run. It re-calculates the coalitional values; after that, the agents form the preferred coalitions. The time complexity of the algorithm is $\mathcal{O}(2^n)$, but it can be reduced to (n^k) by inserting a constraint on the coalition size, in which k represents the highest size allowed. However, it still remains a NP-Complete problem (Shehory and Kraus, 1998).

A genetic algorithm was introduced by Sen and Dutta (2000). The algorithm starts with a randomly generated coalition structure. It iterates through three stages: (i) evaluation; (ii)

selection; and (iii) recombination. It evaluates all the members of the coalition structure, selects members based on these evaluations and modify or exchange their content, so new members are generated. An algorithm based on simulated annealing was proposed in (Keinänen, 2009). The algorithm starts generating random coalition structures. It takes one CS and evaluates its neighbourhood. For a neighbour CS', the algorithm compares if it is better than the current coalition structure. If it is, then the algorithm moves to CS'. Otherwise, the algorithm only moves to the neighbour based on a probability that decreases over time.

The greedy algorithm proposed by Mauro et al. (2010) is based on Greedy Randomized Adaptive Search Procedures (GRASP) (Feo and Resende, 1995). GRASP is an iterative process composed of two phases: (i) a constructive phase in which a feasible solution is produced; and (ii) a searching phase in which a local search is performed on the neighbourhood of the constructed solution. The algorithm starts off with an empty coalition structure. Then, one agent is added to it, as a singleton coalition or as member of some coalition. After that, a local search is performed in the neighbourhood of this coalition structure. The neighbourhood is defined according to five operations: split, merge, shift, exchange, and extract operations. The algorithm finishes when all the agents are added to the coalition structure.

Integer programming can also be used for computing CSs in CSG problems. Once a formulation of the problem is designed, it can be applied to any integer programming solver (Rahwan et al., 2015). However, this approach is slower than both IDP and IP, and it runs out of memory for problems with around 20 agents in the experiments reported by Rahwan et al. (2007a).

2.3.3 The Coalition Linkage Algorithm

C-Link (Farinelli et al., 2016) is a heuristic algorithm (anytime under some conditions) with time complexity $\mathcal{O}(n^3)$. It starts off from the coalition structure of singletons, and to produce a new coalition structure it evaluates all the moves resulting of a merger of any two coalitions (see Figure 2.6). It selects the move that produces the greatest numeric gain, which cannot be undone. Gain is a clustering concept that, when parsed to the coalition formation problem, is described by Equation 2.2.

$$gain(C_i, C_j) = v(C_i \cup C_j) - v(C_i) - v(C_j) \tag{2.2}$$

The equation above states that the gain of merging two coalitions is the difference between the coalition values with and without placing the agents in the same coalition. After evaluating all available moves, the algorithm merges the two most suitable coalitions. This is determined through a ***linkage function***; four such functions are proposed: (i) single-link; (ii) complete-link; (iii) average-link; and (iv) gain-link. Functions (i), (ii), and (iii) are based on pairwise relations between the agents that belong to the same coalition, whilst (iv) considers

PL^t	a_1	a_2	a_3	a_4
a_1	-∞	5	-5	5
a_2	--	-∞	-15	-10
a_3	--	--	-∞	10
a_4	--	--	--	-∞

PL^{t+1}	a_1	a_2	a_3,a_4
a_1	-∞	5	-20
a_2	--	-∞	-15
a_3,a_4	--	--	-∞

PL^{t+2}	a_1,a_2	a_3,a_4
a_1,a_2	-∞	-20
a_3,a_4	--	-∞

Figure 2.6: An example run with four agents for C-Link. The red ovals represent the selected column and row to be merged, and t represents an iteration.

the coalition itself and not individual members. For the remainder of this work, we consider only the gain-link (GL) defined in Equation 2.3, as it obtained the best results in the experiments reported by Farinelli et al. (2016).

$$lf_{\text{GL}}(C_i, C_j) = gain(C_i, C_j) \tag{2.3}$$

The algorithm iteratively updates a partition linkage matrix PL (initially a $n \times n$ matrix, that is, a CS of singletons) which is filled out with the value returned by the linkage function $lf(C_i, C_j)$ for entry (i, j) (note that the diagonal of the matrix is not relevant). The algorithm picks the entry in the table PL resulting in the greatest value of the linkage function and performs a merger. Once this is done, the matrix is updated with the new coalitions and their respective values from the linkage function. The algorithm stops when there is no advantage in merging any two coalitions—when the linkage function for all coalitions in the matrix has zero or a negative value.

2.3.4 Reduction in the Search Space

After introducing some algorithms that solve the coalition structure generation problem, we discuss constraints applied to the process of forming coalitions. This is an important topic as by reducing the number of feasible coalitions and CSs, one reduces hence the search space.

Rahwan et al. (2011) introduced an algorithm to solve locally constrained coalition formation games using the basic CCF framework. It first transforms the set of agents and constraints (i.e., the set of desirable/prohibitive rules and sizes of the allowed coalitions) into an isomorphic representation that outputs only the feasible coalitions. After that, it starts searching for an optimal coalition structure.

Voice et al. (2012b) argued that the current state of the art on coalition formation algorithms is not designed for problems over synergistic graphs. To address this, they proposed an algorithm to enumerate all the feasible coalitions, titled D-SlyCE and another algorithm to find an optimal feasible coalition structure, titled DyCE. In their proposed model, coalition formation with sparse synergies, the set of feasible coalitions is constrained by a graph where vertices represent the agents. The coalition is considered feasible if and only if it is an induced sub-graph on the synergistic graph.

The SlyCE algorithm conducts a depth-first search over a tree representation of the set of feasible coalitions. SlyCE has been proved to be correct (i.e., it only evaluates feasible coalitions), complete (i.e., it can enumerate all feasible coalitions), and non-redundant (i.e., it never evaluates the same coalition twice). Moreover, the search space can be distributed to the agents in the system so that the computation of different coalitions are performed simultaneously. This version of the algorithm called is D-SlyCE (Voice et al., 2012b) and is shown to have the same properties as SlyCE; that is, it is correct, complete, and non-redundant.

DyCE (Voice et al., 2012b) uses a dynamic programming technique to find an optimal coalition structure. It is based on IDP (Rahwan and Jennings, 2008b) with the addition of an initial phase in which the feasible coalitions are enumerated using SlyCE. Then, the authors construct a feasible coalition structure graph (a directed graph) containing nodes that correspond to feasible CSs (i.e., CSs that contain only feasible coalitions). Two nodes are connected by an edge in the resulting graph if a coalition in the first coalition structure is split into two smaller feasible coalitions in the second node. DyCE evaluates only some edges that connect the nodes of a coalition structure graph, like IDP does. To summarise, SlyCE provides the feasible coalitions to DyCE. After that, DyCE goes through each coalition of size $1, \ldots, n$ calculating a value for them. Here, SlyCE is used again in order to provide the feasible subsets of the coalition that is under evaluation.

2.3.5 Graph Colouring-based Approach

DyCE has some drawbacks, as mentioned in (Bistaffa et al., 2014): (i) it requires an exponential amount of memory in the number of agents; and (ii) it is not an anytime algorithm. To overcome those limitations, the authors proposed a new way of searching through the search space of CSs in the CSG problem. The search space is represented in a tree-like manner and each node corresponds to a feasible coalition structure. The root is the coalition structure of singletons connecting compatible agents (that is, their singleton coalitions). They use the *edge contraction* technique to go from one coalition structure to another. The edges connecting coalitions in a given node of the tree are coloured using the notion of *2-coloured graph* (Cormen et al., 2009). Precisely, the colours *green* and *red*. A red colour means the edge cannot be contracted, and a green edge means the otherwise.

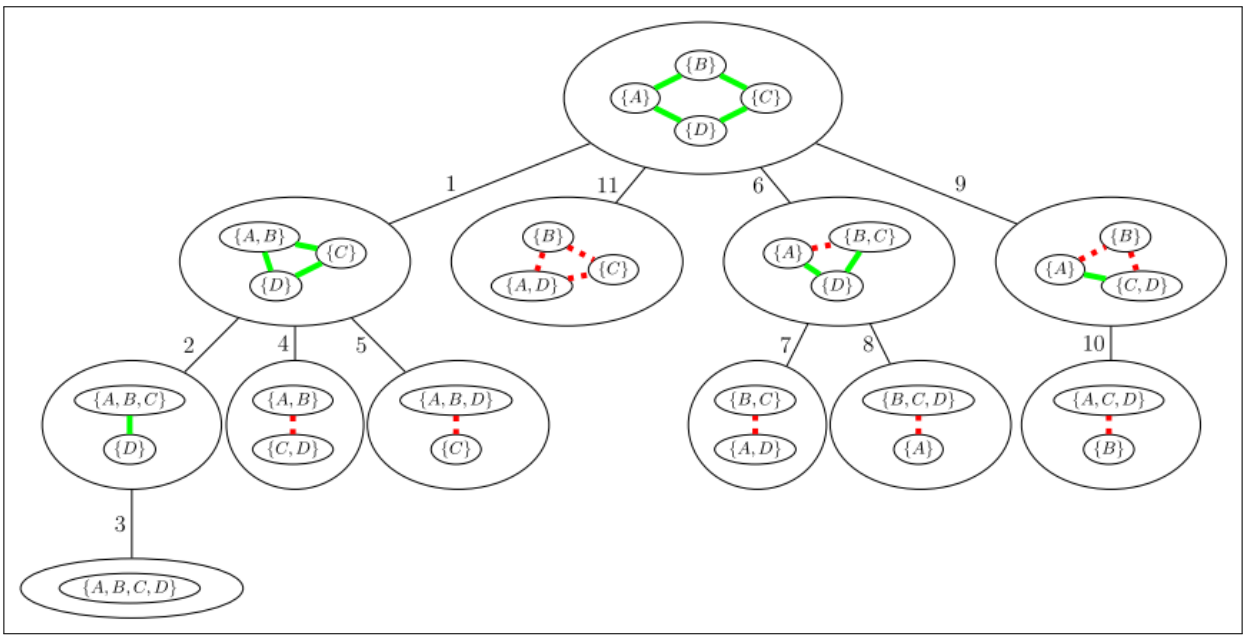


Figure 2.7: A tree-like representation of the space of coalition structures.
Source: Bistaffa et al. (2014)

To generate a new node, one green edge is chosen to be contracted, meaning two coalitions are merged. After merging the coalitions, a new graph representing the resulting CS is generated and the contracted edge is coloured red at the parent node. Moreover, parallel edges are merged and the resulting edge is coloured red if one of its parallel edges is red-coloured; it is coloured green otherwise. This procedure is repeated for all the remaining green edges. For instance, consider the root node in Figure 2.7, and let G be its corresponding graph and CS its corresponding coalition structure (by parsing the nodes in G to coalitions). When the edge between coalitions $\{A\}$ and $\{B\}$ is contracted, a new graph is generated with the corresponding CS $CS' = \{\{A, B\}, \{D\}, \{C\}\}$. This edge is then coloured red in G. After some iterations, the edge connecting coalitions $\{B\}$ and $\{C\}$ in graph G is contracted and a new graph G'' is generated; the corresponding CS is $CS'' = \{\{B, C\}, \{A\}, \{D\}\}$. Note that the edge coloured previously in red (i.e., edge $(\{A\}, \{B\})$) determines the colour of edge $(\{A\}, \{B, C\})$ in graph G''. The algorithm paints the contracted edge in red and goes on with the execution.

The CFSS algorithm uses a branch-and-bound technique that focuses on $m{+}a$ (monotonic-antimonotonic) functions; a general class of characteristic functions. An $m + a$ function is the sum of a *supperadditive* and a *subadditive* functions (Bistaffa et al., 2017a). A supperadditive function has a monotonic behaviour. That is, given two coalitions $C, C' \subset A : C \cap C' = \emptyset$, the value of the union of these coalitions is no less than the sum of the coalitions' separate values, $v(C \cup C') \geq v(C) + v(C')$. Conversely, a subadditive function has an antimonotonic behaviour. The value of the union of these coalitions is no greater than the sum of the coalitions' separate values, $v(C \cup C') \leq v(C) + v(C')$. Then, a bound is assessed for each subtree rooted at any given node in the tree and is computed based on subadditive value of the current CS plus the supperadditive value of the CS containing the grand coalition. If the bound for a subtree is

lesser than the best solution found so far, then it is pruned. For anytime performance, in the worst-case scenario, the CFSS algorithm provides solutions that are at least 88% of the optimum.

2.4 The Monte Carlo Tree Search Method

Monte Carlo Tree Search (MCTS) is usually applied to problems that contain intractable search spaces. Instead of a single algorithm, it is better described as a family of algorithms (Browne et al., 2012) due to the changes required in the searching mechanism to meet each problem's requirements. An algorithm that implements MCTS iteratively builds a tree considering at each expansion (i.e., a new node being added to it) the statistical information collected during former explorations (e.g., following a given path in it). This current knowledge is then used to perform a best-first search to determine where to expand the tree. We shall introduce how the MCTS works and after that, we discuss two approaches that have used this method to solve the coalition structure generation problem.

2.4.1 The Overall Method

In the MCTS method, one may interpret a node in the tree as one of the possible states of a problem. A transition from the current node to one of its child nodes is given by an action being carried out at that state. As MCTS is based on a tree search, we always assume an initial state (i.e., the root of the tree) and conduct a best-first search on its branches (i.e., select iteratively the branch that leads to the greatest immediate reward). To follow this method, one performs the four steps explained below to implement an MCTS-based algorithm. A single iteration is depicted in Figure 2.8.

Selection: Starting *always* at the root node, one recursively selects child nodes until a terminal or not fully expanded node is reached (e.g., there exists actions that have not been carried out in the current node). Note that the selected node might not be a leaf node. To select a child node among the options, we use a utility function to establish priority. The greater the utility value, the greater the priority. This priority order might change as the number of iterations increases leading to different parts of the tree being explored during the search.

Expansion: Once a node has been selected, the algorithm adds a child node of it to the tree. To create a child node, the algorithm needs to consider the available actions for that particular state. To avoid keeping adding nodes that result from the same action, one should keep track of the actions that have not been carried out in it yet. The standard procedure is to select a random action among the options. Note that a single child node or more can be added to the tree per iteration (Browne et al., 2012). Moreover, in some problems, the

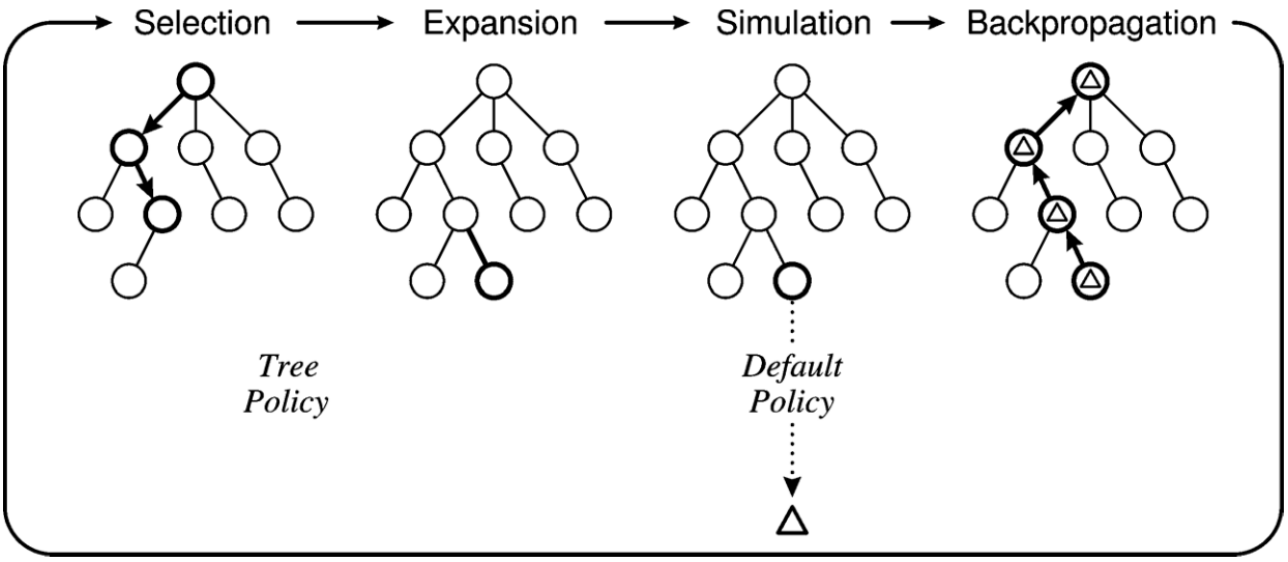


Figure 2.8: An iteration in the MCTS method.
Source: Browne et al. (2012)

space of actions can be either continuous or finite (intractably large tough) (Kim et al., 2020). In those cases, a common approach is to progressively expand a node's frontier once the corresponding number of visits to it is above a given threshold (Lee et al., 2020).

Simulation: In a simulation, one aims to evaluate quickly how promising a subtree rooted at the current node is. This procedure is called *default policy* (aka. play-out or role-out) and results in a *reward*. In its simplest form, one considers a reward to be a scalar, which, for instance, might represent either a victory or a defeat condition (Browne et al., 2012). In some cases, a reward might be a vector of values.

Backpropagation: The simulation step results in statistics that are propagated back throughout the path that led to the simulated node. It consists of the reward that is used to update a node's attributes and an increment in the number of visits to each node in the path.

The selection and expansion steps constitute the *tree policy* which is an important decision point in MCTS. It not only determines which nodes are to be expanded, but also which path is to be explored next. This leads to the well-known problem to trade off exploration and exploitation. That is, continue exploring a single path that is leading to good results, or explore new paths in hope to achieve better results. A standard algorithm for MCTS is called UTC (Kocsis and Szepesvári, 2006) and its main contribution is to consider the node selection step as a multi-armed bandit problem (Auer et al., 2002). To do so, they proposed the heuristic UCB1 as introduced in Equation 2.4:

$$UCB1 = \bar{X}_j + 2C\sqrt{\frac{2\ln N}{N_j}} \tag{2.4}$$

where N is the number of times the current node has been visited, N_j is the number of times the child node j has been visited and $\bar{X}_j$ its value, and $C > 0$ is a constant. The first component of the equation represents the exploitation term and second the exploration. One can see that the exploration term increases in value as the number of visits to that child node remains constant.

Many different strategies exist for dealing with particular characteristics of MCTS. For instance, Yoshizoe et al. (2011) consider an MCTS as a depth-first search instead of a best-first search. After a simulation in a given node, the procedure only returns to the root node if one of its child nodes has now a more promising path. Otherwise, it continues the search at deeper levels of the tree. Variations exist also to view an MCTS as a directed acyclic graph (Childs et al., 2008). This is motivated by the fact that different nodes may represent the same state.

As in MCTS one deals with intractable search spaces, we repeat the above four steps until a halting criteria is met. Usually, this criterium may represent a budget in terms of time or number of iterations. Once it is achieved, one returns a winning action. There are different strategies to decide which action to return. One can choose the action that has the greatest node's value, the root's child visited more frequently, etc. We refer to (Browne et al., 2012) for a broad discussion on MCTS.

2.4.2 MCTS approach to the Coalition Structure Generation Problem

MCTS has been applied recently to the coalition structure generation problem. Wu and Ramchurn (2020) apply the method discussed above to find a coalition structure in the context of CFG. They propose the CSG-UCT algorithm, which is based on UCT (Kocsis and Szepesvári, 2006). In their formulation, each node in the tree corresponds to a CS $CS \in \boldsymbol{\mathcal{CS}}^A$. The root node is the CS containing only singleton coalitions. The set of actions available at each node is a set containing all pair-wise union operations that can be carried out on the given CS. For instance, the CS $\{\{a_1\}, \{a_2\}, \{a_3\}\}$ has the child nodes $\{\{a_1, a_2\}, \{a_3\}\}$, $\{\{a_1, a_3\}, \{a_2\}\}$, and $\{\{a_1\}, \{a_2, a_3\}\}$. Consider as a more comprehensive example the coalition structure graph in Figure 2.3.

To select a node, the authors apply the UCB1 heuristic where the exploitation component is the maximum value found for a node in the corresponding subtree. A roll-out on a node representing the CS CS consists in merging two coalitions that results in the greatest gain:

$$C_1, C_2 = \arg \max_{C_1', C_2' \in CS} [v(CS') - v(CS)] \tag{2.5}$$

where $CS' = CS \cup \{C_1' \cup C_2'\} \setminus \{C_1'\} \setminus \{C_2'\}$. Note that a roll-out always ends at the CS of the grand coalition. Once the iteration budget is reached, one only needs to conduct a best-first search to find the best solution found.

MCTS is also applied to a similar problem in which each coalition is allocated to a task in an *ordered coalition structure* (Präntare et al., 2021). Given a vector of tasks $T = \langle t_1, \ldots, t_m \rangle$,

one aims to form an ordered partition of A, $CS = \langle C_1, \ldots, C_m \rangle$, in such a way that each t_i is assigned to C_i for all $1 \leq i \leq m$. In case t_i is not allocated to any coalition, $\emptyset$ appears at position i in CS. The search is conducted over one agent at time choosing a position in the CS to place it in. This choice is performed by picking the best position after a predefined number of iterations (i.e., finished roll-outs). Therefore, a node is a composition of agents already allocated to coalitions and a permutation of the remaining agents to the m positions. The root node always has m child nodes which represents the assignment of the current agent to one of the positions. To define a priority among child nodes, Präntare et al. (2021) add a third component to UCB1 (originally proposed in (Schadd et al., 2012)), which represents a possible deviation of the child node. The exploitation component is the average value of previous roll-outs through that node. In a roll-out, the authors select a position to assign an agent based on an uniform distribution. The most visited child node of the root is selected as the position where to place the current agent.

2.5 Interdependence of Solutions

The interdependence of solution refers to how two or more problems in which their solutions (i.e., the outcome of an algorithm) depend on each other. As this is the main topic in this thesis, it is important to understand how correlated areas have addressed it.

Many variations of coalitional games have been proposed, for instance, considering tasks (Rahwan et al., 2013), constraints (Rahwan et al., 2011), etc. (for a more comprehensive list we refer to Section 2.2). Our new framework, to be discussed in the next chapter, extends the idea of coalition structure in the literature (Rahwan et al., 2015) to a total order of such structures in which the interdependence between them is established by a binary relation. In fact, some coalitional games are a specialisation of this framework as we will show in Section 3.3.

The idea of interdependence between CFGs are reminiscent of combinatorial auctions (Feldman et al., 2015; Krysta and Ventre, 2015) and overlapping coalition formation (Zick et al., 2012). Combinatorial auctions address the problem where, given a set of items, a set of buyers place bids for a subset of such items (each buyer may evaluate differently each item) and the goal is to maximise the overall buyers' social welfare given a partition of items (Krysta and Ventre, 2015). On the other hand, coalition formation with overlaps drops the constraint of disjoint coalitions in a given coalition structure (Zick et al., 2012). A coalition becomes a vector (of length $|A|$) and each agent establishes a desired contribution to it; a contribution of 0 means the agent does not participate in that coalition. It can be noted that both approaches output a single coalition structure and therefore address a different problem than ours.

A related field to coalition formation is clustering. It plays an important role, especially in dealing with databases, when one aims to group objects based on given criteria. Different approaches for this problem have been proposed and some of them are related to ours, for instance, *meta-clustering* (Ferone and Maratea, 2020). In that setting, the aim is to group different

clustering solutions for a given database and therefore, a meta-clustering algorithm works over solutions of clustering algorithms. Another interesting problem is *top-k* clustering (Guedes et al., 2016). In that problem, given a three-tuple graph having a set of vertices, edges, and attributes, the goal is, from a set of candidate solutions, to select k partitions that maintain quality and are dissimilar from each other.

3. SEQUENCES OF CHARACTERISTIC-FUNCTION GAMES

Our main contribution in this chapter is a new sort of game called Sequential Characteristic Function Game (SCFG). It provides the underlying framework for the remaining work developed in this thesis. The current literature in coalition formation, as we discussed in Chapter 2, lacks mechanisms to model the interdependence among the solutions produced by different CFGs. In fact, to the best of our knowledge, Partition Function Game (PFG) is the only game in which a sort of interdependence is exploited: more precisely, a dependence between a coalition and its coalition structure. The games we introduce here allow us to consider coalition structures that are somehow related to each other and therefore should not be evaluated in isolation. Moreover, although we describe it as a sequence of games, one should bear in mind that all games are not to be solved in isolation. Otherwise, it could lead to sub-optimal outcomes.

Prior to introducing our formal framework, we explain in Section 3.1, by using a simple example, the core ideas of why one should relate different coalition structures. Based on a better understanding of the problem we aim to solve, in Section 3.2 we propose the SCFG framework. We show by introducing some running examples how one could explore the core concepts in it to model different applications. In Section 3.3, we discuss again the coalition formation literature comparing it with the proposed framework. We show that indeed many of the existing frameworks are special cases of SCFG. Our final step in Section 3.4 is to further study constraints applied to an SCFG setting. Two extensions to our game are then proposed to refine which coalitions are allowed to form in each game in the sequence.

3.1 Motivation and Main Notions

Andy (a), Bobby (b), and Carol (c) are planning to go shopping. To do so, they need to travel to a store and there buy certain articles. We assume that each agent has a desire to visit a different store and some of them expect help from the others to help choosing its articles. Two situations occur in this example that require the agents to make a joint decision:

1. to which store are they supposed to go? and
2. to whom should each agent ask for an opinion to buy its articles?

We analyse this particular scenario considering two distinct CFGs.

The first game models their preferences for travelling, where different coalitions travel to different stores. $v_1(\{a\}) = v_1(\{b\}) = v_1(\{c\}) = 1$, $v_1(\{a,b\}) = v_1(\{b,c\}) = 2$, and $v_1(\{a,c\}) = v_1(\{a,b,c\}) = 4$. So Andy enjoys travelling with Carol, and even larger coalitions are also preferred because they will lower the costs. The second game models the shopping itself. Andy is hoping for Bobby's or Carol's help to decide the best article to buy, although Bobby knows much more about those items than Carol. However, if both Bobby and Carol are

together, they only talk about life and are of no help to Andy. Thus, $v_2(\{a\}) = v_2(\{b,c\}) = v_2(\{a,b,c\}) = 0$, $v_2(\{b\}) = v_2(\{c\}) = v_2(\{a,c\}) = 1$, and $v_2(\{a,b\}) = 3$.

The optimal CS CS in game 1 (namely $\{\{a,c\},\{b\}\}$ with a value of 5) is not optimal in game 2 (where the optimal is $\{\{a,b\},\{c\}\}$, with a value of 4). If we take $\{\{a,c\},\{b\}\}$ (optimal in game 1) for both games, we get 5 in game 1 but 2 in game 2, so a total of 7. Similarly, if we consider the optimal structure in game 2 ($\{\{a,b\},\{c\}\}$) and pick it for both games we get also a total of 7. We may choose different structures for the two games and take the optimal in each round: $v_1(\{\{a,c\},\{b\}\}) + v_2(\{\{a,b\},\{c\}\}) = 9$. Both coalition structures are optimal as there is no other with a greater value in each game separately. It means that Andy and Carol travel together to the same store, and Bobby goes to another one (first round). In the second round, Andy and Bobby are supposed to buy articles together, but this is no longer ***feasible*** since they are at different stores.

In this problem, there is an interdependence between the solutions of the two games that must be considered in order to output the final overall solution. We note that this interdependence might be related to the order in which each outcome (i.e., coalition structure) is used. In the example above, the agents form each CS at different points in time; the first to travel to a store and then to buy articles. However, that might not always be the case. Another sort of interdependence might correspond to related structures adopted by the agents. For instance, consider a hierarchical organisation. The decisions made at upper levels of the hierarchy affect all subsequent levels, that is, all the agents placed in lower levels. We study the connection between different games by analysing possible solutions produced by each game separately and checking whether or not they are feasible together.

3.2 Sequential CFGs

To solve the problem stated above, we model the interdependence between the games with a binary relation $\mathcal{R}$ on the set of all coalition structures $\boldsymbol{\mathcal{CS}}^A$. We do so to state that the agents buying articles together must have travelled to the same place. This is important when there are restrictions on the coalition formation so that the sequence of individually optimal structures is not feasible. In our motivational example, the optimal sequence of CSs has a value of $v_1(\{\{a,b,c\}\}) + v_2(\{\{a,b\},\{c\}\}) = 8$; travelling together to the same store but buying at different departments.

We now define sequences of CFGs as a formalisation for scenarios of the type described above, where the set of agents is $\{a,b,c\}$, we have two characteristic function games ordered as Γ_1 and then Γ_2, defined by their characteristic functions v_1 and v_2. In that case, we have a sequence of length two. The outcome of that game should be a pair of coalition structures $\langle CS_1, CS_2\rangle$ that respects the relation $\mathcal{R}$ and results in the greatest value of $v_1(CS_1) + v_2(CS_2)$.

Definition 7 (SCFG $\mathcal{G}$). A Sequential Characteristic-Function Game (SCFG) $\mathcal{G}$ is a tuple $\langle A, \mathcal{H}, \mathcal{R} \rangle$ where:

- A is a set of agents $A = \{a_1, \ldots, a_n\}$;
- $\mathcal{H}$ is a totally ordered set of CFGs $\mathcal{H} = \langle \Gamma_1 = \langle A, v_1 \rangle, \ldots, \Gamma_h = \langle A, v_h \rangle \rangle$;
- $\mathcal{R}$ is a binary relation on $\boldsymbol{\mathcal{CS}}^A \cup \emptyset$ (all coalition structures over A); $(\emptyset, CS) \in \mathcal{R}$ means that CS is allowed to appear in the beginning of a solution sequence.

We always use h to denote the length of the game sequence.

Definition 8 (FCSS $\boldsymbol{CS}$, CSS $\boldsymbol{CS}$). A *solution* for an SCFG $\mathcal{G}$ is a Feasible Coalition-Structure Sequence (FCSS) $\boldsymbol{CS} = \langle CS_1, \ldots, CS_h \rangle$ respecting relation $\mathcal{R}$: $CS_i \, \mathcal{R} \, CS_{i+1}$, $0 \leq i < h$. We set $CS_0 = \emptyset$. We call this condition *feasibility*. If a sequence is not feasible, we omit the F and call it a Coalition-Structure Sequence (CSS).

The main point is that we cannot just take the optimal CS of each game (see Section 3.1) whenever we seek a sequence of CSs that maximise the overall value. We need to make sure that the sequence is feasible, which is specific to each particular application. Moreover, $\mathcal{R}$ can only model two subsequent positions. Consider Example 4.

Example 4. Assume a company is starting a new branch for a new product. A set of employees will work on the development of this product and to make the employees familiar with one another, the company proposes a training session[1]. The session is divided into three interconnected projects in which a coalition structure is formed for each one of them. The first two projects address participants' technical skills, hence they use the same characteristic function. The last project addresses general skills and has a specific valuation. Therefore, the company aims to form an FCSS

$$\boldsymbol{CS} = \langle CS_1, CS_2, CS_3 \rangle.$$

Out of the three CSs in the FCSS above, we can only link both CS_1 and CS_2, and CS_2 and CS_3. To promote the employees to get to know each other during the training session, one might be interested in putting them in different coalitions during the execution of each project. To do so, we might use the binary relation below.

Definition 9 ($\mathcal{R}_H$). For any subsequent $CS, CS' \in \boldsymbol{CS}$, $CS \, \mathcal{R}_H \, CS'$ iff for all $a \in A, C \in CS, C' \in CS'$, s.t. $a \in C$, it holds that if $a \in C'$, then $C \cap C' = \{a\}$.

By enforcing $\mathcal{R}_H$, employee a will have different teammates in both first and second projects, as well as in the second and third project. However, that does not hold for the first and third project. With $\mathcal{R}$ defined as a binary relation on the set of coalition structures we cannot

[1]This application is related to the employee training problem introduced by Zhang et al. (2017a); here presented as a simplified version though.

express, for example, that two agents ***must not*** work together more than once throughout the sequence of projects. To produce such an outcome, we would have to quantify over the coalition structures along the sequence. Formally, for Example 4, given ***any*** $CS, CS' \in \boldsymbol{CS}$, we require for all $a \in A$, $C \in CS$, $C' \in CS'$: if $a \in C$ and $a \in C'$, then $C \cap C' = \{a\}$. We aim to address this in future work.

Another important notion in Definition 7 regards the outcome produced in the first CFG. The pair $(\emptyset, CS)$ extends the underlying concept of $\mathcal{R}$ (i.e., relating solutions computed by CFGs) by determining which CSs may begin a sequence. In practical applications, this might reduce significantly the search space as we do not search the space of ***pairs*** of coalition structures; instead, we search the space of single coalition structures. Consider the example below.

Example 5. A fire brigade is called to act upon a wildfire incident that is approaching a town. The entire fire brigade has enough experts and resources to fight off simultaneously the wildfire event in nine different fronts. To be ready for any new incident, the brigade supervisor decides to take all the resources and experts to the field. The brigade will be then divided initially into four coalitions, and as soon as a new demand appears, one of these coalitions split itself up to fight off the wildfire in a new front.

As one is interested in forming a new CS as new demand is detected, the binary relation connecting the outcomes can be designed as follows.

Definition 10 ($\mathcal{R}_O$). For any subsequent $CS, CS' \in \boldsymbol{CS}$, $CS\, \mathcal{R}_O\, CS'$ iff there is $C \in CS$ s.t. all other coalitions $C' \in CS$, $C' \neq C$, are also contained in CS', and CS' contains two additional coalitions C_1, C_2 with $C_1 \cup C_2 = C$.

As relation $\mathcal{R}_O$ ensures that from the initial CS only one coalition is allowed to split, as soon as a new demand appears the fire brigade advances to a new coalition structure in $\boldsymbol{CS}$. Therefore, the supervisor allocates all resources to CS $CS_1 \in \boldsymbol{CS}$ in advance. Doing so, the resources to be assigned to the next front (i.e., the ones that will form a new coalition) are already placed together. An important detail in Example 5 is that the brigade is initially divided into 4 coalitions. To model it, one can do as follows: $(\emptyset, CS) \in \mathcal{R}_O$ for all $CS \in \boldsymbol{CS}^A$ such that $|CS| = 4$.

So far, we have introduced the SCFG framework and its solution concept (i.e., an FCSS). However, we have not discussed how one compute a value for an FCSS. This is addressed in Definition 11.

Definition 11 ($\mathcal{V}$). Given an FCSS (similarly a CSS) $\boldsymbol{CS} = \langle CS_1, \ldots, CS_k \rangle$, where $1 \leq k \leq h$, the value for $\boldsymbol{CS}$ is calculated by $\mathcal{V}(\boldsymbol{CS}) = \sum_{i=1}^{k} V_i(CS_i)$, where $V_i(CS_i) = \sum_{C \in CS_i} v_i(C)$.

The goal is then to solve the optimisation problem by finding an optimal FCSS $\boldsymbol{CS}^*$:

$$\boldsymbol{CS}^* = \arg\max_{\boldsymbol{CS}} \mathcal{V}(\boldsymbol{CS}).$$

3.3 SCFG and Similar Problems

Given the formal definition of our new sort of game, we investigate how it relates to the existing literature on coalition formation. To this purpose we consider: (i) Constrained Coalition Formation Game (CCFG); (ii) Task-Based Characteristic-Function Game (TCFG); (iii) discrete overlapping coalitions; and (iv) combinatorial auctions. Then, we investigate Partition-Function Games (PFG) which carries the notion of interdependence between a coalition and the coalition structure it belongs.

We introduce the games above as *special cases* of SCFG. That means, given any instance of a particular game, it can be modelled by a corresponding SCFG instance iff:

1. there exists at least one optimal solution in the original game instance that corresponds to an optimal solution in the SCFG instance; and
2. the value of both optimal solutions are equal.

Rahwan et al. (2011) investigated constraints being applied to the coalition formation problem. They proposed a general model CCFG as a tuple $\langle A, \boldsymbol{\mathcal{CS}}_{cst}, v\rangle$ where $A = \{a_1, \ldots, a_n\}$ is the set of agents; $\boldsymbol{\mathcal{CS}}_{cst} \subseteq \boldsymbol{\mathcal{CS}}^A$ is the set of feasible coalition structures; and v assigns a real value to each coalition. An optimal coalition structure is one with the greatest value. For more details about this setting we refer to Section 2.2.3. Interestingly, this formulation to represent constraints can also be expressed using an SCFG. We show that all coalition structures CS for a CCFG Γ_C are in one-to-one correspondence to the FCSS $\boldsymbol{CS}$ for a corresponding SCFG and the values are identical.

Lemma 1. Constrained coalition formation games CCFG, as defined in (Rahwan et al., 2011), are special cases of SCFGs.

Proof. Let $\langle A, \boldsymbol{\mathcal{CS}}_{cst}, v\rangle$ be a CCFG with h coalition structures $\boldsymbol{\mathcal{CS}}_{cst} = \{CS_1, \ldots, CS_h\}$. We construct an SCFG $\mathcal{G}$ with h copies of the ordinary characteristic function game $\langle A, \frac{1}{h}v\rangle$. Therefore we get Γ_i with $1 \leq i \leq h$ where $v_i(C) \coloneqq \frac{v(C)}{h}$. This is our set $\mathcal{H}$. The relation $\mathcal{R}$ is defined by $CS\ \mathcal{R}\ CS'$ iff $CS, CS' \in \boldsymbol{\mathcal{CS}}_{cst}$. So we essentially repeat the same game h times. The modified v ensures the right value. □

Similarly, we show that Task-Based Characteristic Function Games (Rahwan et al., 2013), formally defined as $\langle A, T, v\rangle$, are also special cases of SCFGs. In this setting, we are given a set of tasks T and a characteristic function v which assigns a value to each pair coalition and task (one coalition works on one task): $v : 2^A \times T \to \mathbb{R}$. An optimal CS CS^* is one where $\sum_{C \in CS} v(C, t_C)$ is maximal, where $t_C \in T$ are different tasks in T: $t_C \neq t_{C'}$ for $C \neq C'$.

Lemma 2. Task-Based Characteristic Function Games TCFG, as defined in (Rahwan et al., 2013), are special cases of SCFGs.

Proof. Consider a TCFG $\langle A, T, v\rangle$; it suffices to construct an equivalent CCFG $\Gamma_C = \langle A', \boldsymbol{\mathcal{CS}}'_{cst}, v'\rangle$. Let $A' \coloneqq A \cup T$ and let the feasible coalition structure $\boldsymbol{\mathcal{CS}}'_{cst}$ consist of all coalition structures CS for A' that satisfy: (1) for all $C \in CS : |C \cap T| = 1$, and (2) $C \setminus T \neq \emptyset$. Thus, the feasible coalition structures for A' are exactly the coalition structures of the original A. This allows us to define $v' : A' \to \mathbb{R}$ by $v'(C') \coloneqq v(C, t)$ where $C' = C \cup \{t\}$ (which is well defined because of our definition of feasible coalition structures in $\mathcal{R}$). Clearly, the optimal $\boldsymbol{CS}^*$ of an SCFG for Γ_C corresponds to the optimal CS^* for TCFG and has the same value. □

Another interesting setting we study is called Discrete Overlapping Coalition Formation games (DOCF) (Zick et al., 2019). For more details see Section 2.2.1. DOCF drops out the constraint imposed by a CFG in which an agent can participate in only a single coalition. In this setting, a game Γ_O gets as input not only a set of agents and a valuation function, but also information about how the agents are able to ***split*** their forces and contribute to several coalitions. The agents can therefore decide to enter one coalition with 30%, others with 20%, etc; however, the overall sum is at most 100%. Zick et al. (2019) model this using a vector $(W_1, \ldots, W_n)$ where $W_i \in \mathbb{Z}_0^+$. Therefore, a DOCF is of the form $\Gamma_O = \langle A, (W_1, \ldots, W_n), v_{DOCF}\rangle$. The set $\mathcal{W} = \{0, 1, \ldots, W_1\} \times \ldots \times \{0, 1, \ldots, W_n\}$ contains all possible ways that the n agents can contribute to the coalitions. This leads to the definition of a ***partial coalition*** $\mathbf{c} = (c_1, \ldots, c_n)$, where $\mathbf{c}$ is simply an element of $\mathcal{W}$. The characteristic function v_{DOCF} in a DOCF is a mapping from $\mathcal{W}$ onto $\mathbb{R}_0^+$ (one should assume $v_{DOCF}((0, \ldots, 0)) = 0$).

A CS in an DOCF is a finite list of partial coalitions $(\mathbf{c}_1, \ldots, \mathbf{c}_k)$ subject to:

(i) $\mathbf{c}_i \in \mathcal{W}$ for all $i = 1, \ldots, k$;

(ii) for all $a_j \in A$, we have $\sum_{\mathbf{c} \in \mathrm{CS}} c_j \leq W_j : c_j \in \mathbf{c}$.

The aim is, as usual, to find a CS with a maximal value (sum of the values of all partial coalitions in it).

Lemma 3. Discrete overlapping coalition formation games DOCF, as defined in (Zick et al., 2019), are special cases of SCFGs.

Proof. Let a DOCF $\Gamma_O = \langle A, (W_1, \ldots, W_n), v_{DOCF}\rangle$ be given. The idea to construct a corresponding $\langle A', \mathcal{H}, \mathcal{R}\rangle$ is to view each contribution of an agent as a new agent ***on its own.*** Therefore, we model the fact that agents can contribute to several coalitions by replacing each agent a_i by W_i-many new agents $a_{i,1}, a_{i,2}, \ldots, a_{i,W_i}$ where each agent corresponds to a weight of 1. These agents can be seen as mini-clones of the original agent. For any chosen weight $\leq W_i$ we simply take that many agents.

Therefore, the set A' contains W_1-many agents of sort 1, W_2-many agents of sort 2, …, and W_n-many agents of sort n. Now coalitions $C \in 2^{A'}$ are in many-to-one correspondence (surjective mapping) with partial coalitions of DOCF: We simply reduce C by counting all agents of the same sort (each of these mini agents contributes exactly 1). We get a vector

$\mathbf{c}^C \in \mathcal{W}$ containing as entries exactly the number of agents of this sort, and use it to assign the characteristic function

$$v_{new}(C) := v_{DOCF}(\mathbf{c}^C).$$

Our analysis therefore shows that on the level of coalitions, we can simulate a DOCF by one single CFG.

Now we are slightly modifying the original notion of a CS in a DOCF, without changing its semantics.

1. In the original definition, the authors use a finite list, rather than a multiset. However, the ordering does not play any role so that we choose a multiset in order to get a better representation and thus a simpler equivalence to a SCFG. Both definitions are equivalent.

2. In the original definition, the authors use "$\sum_{\mathbf{c}\in \mathrm{CS}} c_j \leq W_j$", but this can be wlog replaced by "$\sum_{\mathbf{c}\in \mathrm{CS}} c_j = W_j$". The reason is that in the original definition, all the missing agents also form a coalition, but its value must be 0 (otherwise it would only increase the value of the whole CS). So from the viewpoint of the maximal CS, we might as well include this missing coalition, as it does not add anything.

So for now a CS in a DOCF is a multiset of partial coalitions $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$, such that for all $a_j \in A$, we have $\sum_{\mathbf{c}\in \mathrm{CS}} c_j = W_j$.

What about coalition structures in $\langle A', \mathcal{H}, \mathcal{R}\rangle$ and in $\langle A, (W_1, \ldots, W_n), v_{DOCF}\rangle$? As above, by identifying agents of the same sort, each CS over A' transforms into a multiset which is a CS in a DOCF (this is a many-one reduction). And this mapping is surjective, each CS in a DOCF comes from (possibly several) CS over A'.

So we end up with $|\mathcal{H}| = 1$ and the trivial reflexive $\mathcal{R}$: an optimal feasible FCSS of our SCFG are exactly in correspondence with an optimal CS in DOCF. □

Apart from coalition formation problems, we also investigate how SCFG relates to similar problems. In combinatorial auction problems (CAP) (Krysta and Ventre, 2015), we are given as input a tuple $\langle A, U, V\rangle$, where $A = \{a_1, \ldots, a_n\}$ is the set of agents, $U = \{u_1, \ldots, u_m\}$ is a set of goods, and $V = \{v_1, \ldots, v_n\}$ is the set of private valuation functions (one for each agent) $v_i : 2^U \mapsto \mathbb{R}$. The goal here is to maximise $\sum_{i=1}^{n} v_i(U_i)$ for a ***partition*** $U_1, \ldots, U_n$ of items from U between the agents.

Lemma 4. Combinatorial auction problems CAP, as defined in (Krysta and Ventre, 2015), are special cases of SCFGs.

Proof. Given a CAP tuple $\langle A, U, V\rangle$, we have to define a corresponding $\langle A', \mathcal{H}, \mathcal{R}\rangle$. To this end, we set $A' \coloneqq A \cup U$ as the new set of agents. We define $\mathcal{H}$ by the set of games $\Gamma'_i = \langle A', v'_i\rangle$

($i \in A$, ordered in that way) where v'_i is based on the original valuations of CAP and defined by

$$v'_i(C) = \begin{cases} v_i(C \setminus \{a_i\}) & \text{if } C \cap A = \{a_i\}; \\ 0 & \text{otherwise.} \end{cases}$$

This is well-defined as $C \setminus \{a_i\} \subseteq U$ for $C \cap A = \{a_i\}$, and all other coalitions get a value of 0. The coalitions with (potentially) nonzero value in coalition structures for the game Γ'_i have the form $U' \cup \{a_i\}$ for $U' \subseteq U$ and their value is $v_i(U')$, so they correspond to the U_i in the CAP.

Now we have to model the fact that the U_i are a partition of U and that the solutions of the CAP correspond exactly to the solutions of the SCFG. We do this by choosing an appropriate relation $\mathcal{R}$: $(\text{CS}_i, \text{CS}_{i+1}) \in \mathcal{R}$ if, by definition,

1. $\text{CS}_i = \{U_i \cup \{a_i\}, A \setminus \{a_i\} \cup U \setminus U_i\}$ and $\text{CS}_{i+1} = \{U_{i+1} \cup \{a_{i+1}\}, A \setminus \{a_{i+1}\} \cup U \setminus U_{i+1}\}$, where $U_i, U_{i+1} \subseteq U$, and
2. $U_i \cap U_j = \emptyset$ for $i \neq j$, and
3. $\bigcup_i U_i = U$.

The function $\mathcal{V}(\boldsymbol{CS}) = \sum_{CS \in \boldsymbol{CS}} \sum_{C \in CS} v'_i(C)$ ensures that an optimal FCSS will have the same value as an optimal partition of items for the CAP. □

Our final step is to study Partition-Function Games (PFG) (Chalkiadakis et al., 2011, Section 5.2). We refer the reader to Section 2.2.2 for more information regarding that game. This is an interesting game as the value of a coalition C is influenced by the CS to which it belongs; that is, the other coalitions might influence the value of C. This is called *externalities*. This leads to the definition of an *embedded coalition* EC which is a pair (C, CS), where $C \subseteq A$, $CS \in \boldsymbol{\mathcal{CS}}^A$, and $C \in CS$. The set of all embedded coalitions is denoted as $\boldsymbol{\mathcal{EC}}^A$. A PFG Γ_P is given by a tuple $\Gamma_P = \langle A, u \rangle$, where $A = \{a_1, \ldots, a_n\}$ is a set of agents and $u : \boldsymbol{\mathcal{EC}}^A \to \mathbb{R}$ is a mapping that assigns a real number to an embedded coalition (C, CS). We show that PFGs are also special cases of SCFGs.

Lemma 5. Partition-Function Games PFGs, as defined in (Chalkiadakis et al., 2011), are special cases of SCFGs.

Proof. Given a PFG $\Gamma_P = \langle A, u \rangle$, we need to represent all embedded coalitions that u assigns a value to. We do so by creating a corresponding SCFG instance $\mathcal{G} = \langle A, \mathcal{H}, \mathcal{R} \rangle$ and inserting in $\mathcal{H}$ as many games $\Gamma = \langle A, v \rangle$ as exists embedded coalitions $EC \in \boldsymbol{\mathcal{EC}}^A$. Let $\boldsymbol{\mathcal{EC}}^A$ be arranged in any order. Therefore, $\Gamma_i = \langle A, v_i \rangle \in \mathcal{H}$ and $h = |\boldsymbol{\mathcal{EC}}^A|$. We let the valuation $v_i(C)$ correspond to $u(EC_i)$ when the coalitions match. Precisely,

$$v_i(C) \coloneqq \begin{cases} u((C', CS)) & \text{if } C = C' : (C', CS)_i \in \boldsymbol{\mathcal{EC}}^A; \\ 0 & \text{otherwise.} \end{cases}$$

Now, we let $\mathcal{R}$ be the set containing pairs (CS, CS') where $CS = CS'$. We construct $\mathcal{R}$ while building the sequence of games in such a way that for every embedded coalition (C, CS) we have $CS\ \mathcal{R}\ CS$; that is, $\mathcal{R}$ is reflexive.

Clearly, given an optimal CS CS^* that is a solution of game Γ_P, there exists a corresponding FCSS $\boldsymbol{CS}^*$ of the same value as only the valuation v_i evaluates an EC $EC_i \in CS^*$: $EC_i \in \boldsymbol{\mathcal{EC}}^A$ with a value different from 0. □

In this section we have compared SCFG with existing frameworks in the literature. Our next step is to investigate fine-grained constraints that could be applied to $\mathcal{R}$.

3.4 Level-dependent Constraints

So far we have considered a sequence of CFGs in which the feasibility is given by pairs of coalition structures in $\mathcal{R}$. This sort of constraint is applied to every game in the sequence. However, in some applications, coalition structures at particular positions might have their own peculiarities. For instance, at position 3 the sizes of the coalitions are restricted: only sizes smaller than 3 are allowed. Let CS_s be a coalition structure that follows this constraint and assume a three-game sequence: $h = 3$. For the above constraint to hold, we must have: $CS\ \mathcal{R}\ CS_s$, where CS can be any coalition structure. However, if $CS_1\ \mathcal{R}\ CS_2$ and $CS_2\ \mathcal{R}\ CS_3$ where CS_3 does not follow the size constraint above, but is a solution of games Γ_2 and Γ_3, then $\langle CS_1, CS_2, CS_3\rangle$ must be a solution, which we might not want.

3.4.1 Constraints Applied to Transitions

In order to avoid situations like the one above, one might model constraints in the transition from one level (i.e., position) to the next one. Consider again the example above. We can determine which CSs are feasible in the transition from level 2 to level 3 as a binary relation on its own. Let $R_2 \subseteq \boldsymbol{\mathcal{CS}}^A \times \boldsymbol{\mathcal{CS}}^A$ be such a relation. Then, we let $(CS, CS_s) \in R_2$ be the only pairs in R_2 such that CS can be any coalition structure. Clearly, the sequence $\langle CS_1, CS_2, CS_3\rangle$ is no longer feasible as $(CS_2, CS_3) \notin R_2$. We formally define this sort of game.

Definition 12 (SCFG with multiple relations $\mathcal{G}^{mult}$). An SCFG with ***multiple relations*** $\mathcal{G}^{mult}$ is a tuple $\mathcal{G}^{mult} = \langle A, \mathcal{H}, \mathcal{R}\rangle$ where:

- A is a set of agents $A = \{a_1, \ldots, a_n\}$;
- $\mathcal{H}$ is a totally ordered set of CFGs $\Gamma_i = \langle A, v_i\rangle$, $1 \leq i \leq h$; and
- $\mathcal{R}$ is an ordered set of binary relations $\mathcal{R} = \langle R_1, \ldots, R_{h-1}\rangle$ such that $R_i \subseteq \boldsymbol{\mathcal{CS}}^A \times \boldsymbol{\mathcal{CS}}^A$: $1 \leq i < h$.

Doing so, we can express constraints on the transition between subsequent positions in the sequence. Regarding the solution concept of an SCFG, we slightly modify it.

Definition 13 ($\boldsymbol{CS}$). A *solution* for an SCFG with *multiple relations* $\mathcal{G}^{mult}$ is a Feasible Coalition-Structure Sequence (FCSS) $\boldsymbol{CS} = \langle CS_1, \ldots, CS_h \rangle$ respecting the relations in $\mathcal{R}$:

$$CS_1 \; R_1 \; CS_2, \ldots, CS_{h-1} \; R_{h-1} \; CS_h$$

such that $R_i \in \mathcal{R} : 1 \leq i < h$.

Quite interestingly, it turns out this new form of game is in essence an SCFG modelled in a particular way to take those transitions into account. We show that the solutions of SCFGs with multiple relations correspond, in fact, to solutions in SCFGs.

Lemma 6. Any SCFG with multiple relations $\mathcal{G}^{mult}$ can be modelled as an ordinary SCFG $\mathcal{G}$ over a superset $A' \supseteq A$ of agents.

Feasible solutions are in one-to-one correspondence and have identical values.

Proof. Given a SCFG with multiple relations $\mathcal{G}^{mult}$ over the set A, we define an ordinary $\mathcal{G}$ over A' with a relation $\mathcal{R}$ over $\boldsymbol{\mathcal{CS}}^{A'}$. We need to let $\mathcal{R}$ distinguish between the levels a coalition structure belongs.

We let $A' \coloneqq A$ and add in it new ***distinguished*** agents a'_1, a'_2, …, a'_h; one for each level. Although we have now many more coalition structures, we only need to consider very few of the new ones. Precisely, we consider only those coalition structures that contain exactly one of the new agents as a singleton coalition: $\{a'_i\}$ must be contained in a CS CS and no other $\{a'_j\}$. Then we interpret this CS as being on level i.

The values of the CFGs are taken from the original game by forgetting all new agents a'_i in any coalition and assigning all coalitions consisting of only the distinguished agents the value of 0. $\mathcal{R}$ is chosen such that R_i is selected when the CSs have the required form explained above and simulates $\langle R_1, \ldots, R_{h-1} \rangle$.

Then any feasible solution of the extended game $\mathcal{G}^{mult}$ corresponds to a feasible solution in the ordinary game over the larger set of agents, and the values are the same. □

3.4.2 Constraints on Games

Another way to address the problem previously mentioned is to consider constraints applied to a CFG itself. Considering a single CFG, constraints have long been studied in the coalition structure generation problem (Rahwan et al., 2015) and a number of approaches have been proposed, for instance, graph-based games (Voice et al., 2012b; Bistaffa et al., 2014) and rules-based games (Rahwan et al., 2011). We discussed this topic in details in Section 2.2.3.

Example 6. Consider again the training session setting in Example 4. Suppose some of the company's employees have already worked together in past projects and the company has kept track of this. To form an initial trust relationship, the human resource department has established that all members of a team must be connected (directly/indirectly) through the social network recorded by the company. In each project, to avoid homogeneous team, participants should possess different skills.

In the example above, one can see that the social network among employees can be easily modelled by a graph in which each node corresponds to an employee. As the teams are meant to have a variety of skills that correspond to the expectation of each project, we can drop in the graph those edges connecting employees of similar skills. For the training session problem at hand, three different social networks (i.e., graphs) are created: one for each project. Topological-oriented constraints provide a good starting point to model constraints that come up from each CFG in the sequence. An interesting approach in the literature that uses a graph to model a CFG problem is based on the concept of Valuation Structure (VS) (Greco and Guzzo, 2017). The authors show that there exists problems that are solved either by rules (prohibition/permission) or by graph-oriented means. To address that limitation, a VS combines both worlds: prohibition constraints and an interaction graph. Therefore, in this section, we investigate how to integrate Valuation Structures (VS) to further express constraints on each CFG in an SCFG instance. The resulting integration is published in (Krausburg et al., 2021a).

Recall that in a VS setting $\sigma = \langle G, S, \alpha, \beta, x, y\rangle$ (Definition 6), the graph G and the set of pivotal agents S determine the set of *allowed* coalitions $\boldsymbol{\mathcal{C}}^\sigma$: a coalition $C \subseteq A$ is allowed to form if and 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 (Equation 2.1).

In this work, we are interested in constraints applied to the coalitions and therefore, we consider only G and S and leave out the remaining variables that modify a valuation. Without loss of generality, one may 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 14 (Sequential CFGs induced by VSs (SEQVS)). A sequential CFG induced by a sequence of VSs 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 CS CS_{i+1} may follow CS CS_i in a sequence 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 15 (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 $V_i(CS_i) = \sum_{C \in CS_i} v_i(C)$, $1 \leq i \leq h$.

A solution for a SEQVS game instance is an optimal FCSS

$$\boldsymbol{CS}^* = \arg\max_{\boldsymbol{CS}} \mathcal{V}(\boldsymbol{CS}).$$

By adding Valuation Structures to the SCFG framework, 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 3.1 in which an optimal solution in an SCFG instance is no longer feasible in a SEQVS setting.

However, constraints on the number of agents a coalition may have are left out of the VS framework, hence of SEQVS. One should note that it is not possible to model this sort of constraint using only an interaction graph and prohibition rules. In fact, Greco and Guzzo (2017) pointed it out as a future direction to extend the VS framework. To better understand the importance of size constraints, consider Example 7.

Example 7. Consider again the wildfire incident in Example 5. Suppose the fire brigade has at its disposal nine wildland fire trucks (type 3 Fire Engine)[2]. Each wildland fire truck can carry 3 (minimum required to operate the truck) to 5 personnel.

[2]https://www.piercemfg.com/pierce/blog/types-of-fire-trucks. Accessed on 20 December 2021.

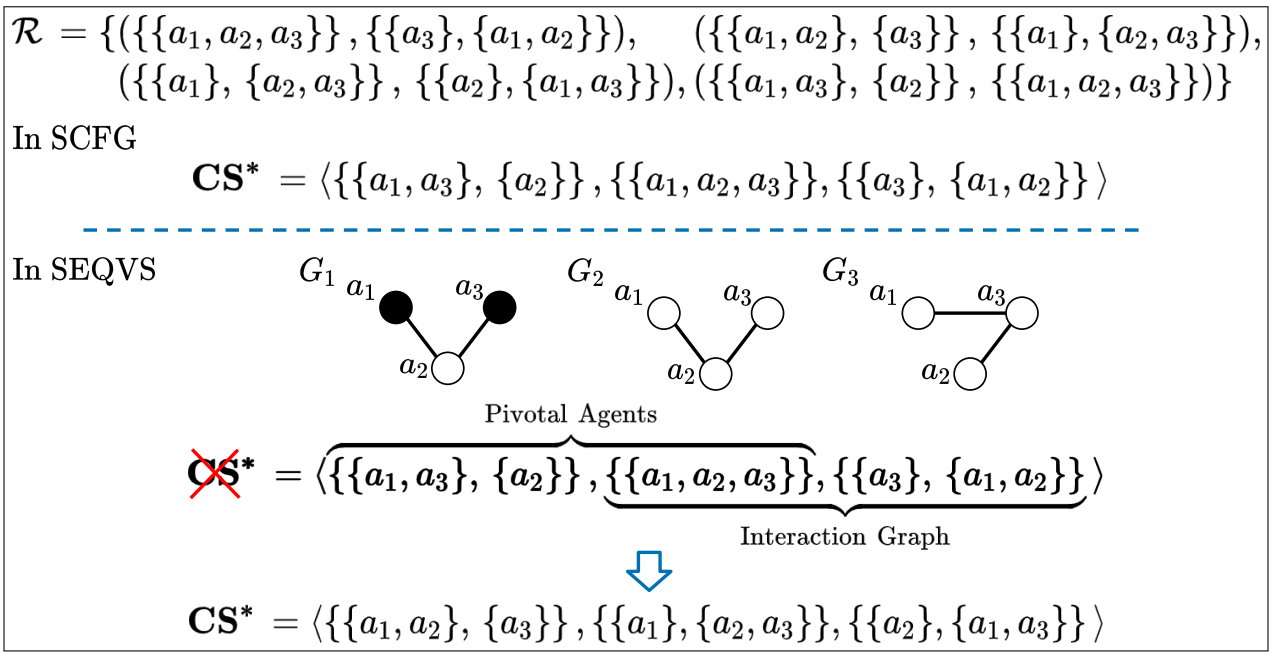


Figure 3.1: SCFG and SEQVS instances for $|A| = h = 3$ that share the same $\mathcal{R}$. An optimal solution in the SCFG instance is no longer feasible in the SEQVS one due to the pivotal agents in the first game (solid black nodes in G_1) and to the graph G_3 in the third game. For readability, we omit in $\mathcal{R}$ the pairs that start with $\emptyset$; assume that for all CSs in it such pair exists.

In the example above we note a physical constraint imposed by the number of seats available in a truck. Let d be the maximal number of personnel such a fire truck can carry. Then, for each game $\Gamma_i \in \mathcal{H} : 1 \leq i \leq 6$ in our sequence, we can compute the maximal number of personnel each coalition can have $\bar{d}_i = d \times ((h+1) - i)$. Then, we require any coalition to have size $|C| \in \{3, \ldots, \bar{d}_i\}$. Any CS that does not comply with the constraint above will eventually make a CSS unfeasible. For instance, assume a coalition C formed in game Γ_5 that has size $|C| = 4$. In the next game Γ_6, it cannot be split in a CS CS_6 into two new coalitions as the outcome would not comply with the size constraint determined above.

Size constraints are also important in ride-sharing problems (Bistaffa et al., 2017b). Similarly, a coalition is bounded by the number of seats available in a given vehicle. Moreover, in Chapter 6, we shall introduce an application in which coalitions are also bounded in size. Motivated by the examples above, we propose an extension to VS by adding a set of allowed sizes. In particular, we follow the definition of size constraints in the CCF framework (Rahwan et al., 2011).

Definition 16 (Sized Valuation Structure (SVS)). A *sized valuation structure* π on a CFG game $\Gamma = \langle A, v \rangle$ is a tuple $\langle G, S, Z \rangle$, where:

- $G = \langle A, E \rangle$ is an undirected graph on the set of agents A (called the *interaction graph*);
- $S \subseteq A$ is a set of *pivotal agents*; and
- $Z \subseteq \mathbb{N}$ is a set of allowed sizes that every coalition C must follow. That is, $|C| \in Z$.

The notation for Γ^π, $\boldsymbol{\mathcal{C}}^\pi$ and $\boldsymbol{\mathcal{CS}}^\pi$ relate now to SVS constraints. The set of allowed coalitions $\boldsymbol{\mathcal{C}}^\pi$ contains a coalition C such that: (i) the induced sub-graph of C over G is connected; (ii) $|C \cap S| \leq 1$; and (iii) $|C| \in Z$. This induces the set of all allowed coalition structures $\boldsymbol{\mathcal{CS}}^\pi$. We shall extend the definition of SCFG induced by VSs (Krausburg et al., 2021a) to take into account SVSs. This is done by replacing a VS σ by an SVS π.

Given an ordinary CFG $\Gamma = \langle A, v \rangle$, we let Γ^π be the game $\langle A, v \rangle$ induced by a sized valuation structure $\pi = \langle G, S, Z \rangle$. Our main notion below is an ordered sequence of games $\langle \Gamma_1^{\pi_1}, \ldots, \Gamma_h^{\pi_h} \rangle$ which requires a corresponding sequence of coalition structures $\langle CS_1, \ldots, CS_h \rangle$ as solution.

Definition 17 (Sequential CFGs induced by SVSs (SEQSVS))**.** A sequential CFG induced by a sequence of SVSs 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 $\pi_i = \langle G_i, S_i, Z_i \rangle$ with the SVS parameters described above, one for each game $i = 1, \ldots, h$;
- $\mathcal{R}$ is a binary relation on $\boldsymbol{\mathcal{CS}}^A \cup \{\emptyset\}$.

This tuple determines the sequence $\boldsymbol{\Gamma} = \langle \Gamma_1^{\pi_1}, \ldots, \Gamma_h^{\pi_h} \rangle$ of CFGs induced by SVSs, with $\Gamma_i = \langle A, v_i \rangle$ and $\pi_i = \langle G_i, S_i, Z_i \rangle$, where v_i are characteristic functions and $\pi_i \in \Pi$, for $i = 1, \ldots, h$.

The solution concept for a SEQSVS as well as the optimisation problem follow the SEQVS definitions.

Chapter Summary

In this chapter, we investigated the interdependence between solutions for Characteristic-Function Games (CFGs). We noted a lack of work in this topic and proposed the first general framework to address it, namely Sequential Characteristic-Function Game (SCFG). Quite interesting, by modelling the interdependence of solutions, one is able to model different frameworks available in the literature. Among them, we have investigated a general constrained coalition formation, a task-based setting, discrete overlapping coalition formation, partition-function games, as well as combinatorial auctions. This concludes Contribution 1. Our new setting offers many challenges in terms of computing solutions and modelling such problems, and we showed that variations of the proposed game (i.e., level-specific constraints) can be in fact modelled in the original framework. To facilitate the modelling regarding level-specific constraints, we integrated SCFG with valuation structures; a setting in which constraints are induced over a CFG. This concludes Contribution 2. Moreover, motivated by the fact that constraints on the size

of coalitions may have are also an important feature in many real-world applications, we proposed a sized-based version of valuation structure and integrated it with SCFG. This concludes Contribution 3. Our next step is to investigate how difficult it is to solve problems related to SCFG.

4. THE COST OF SOLVING SCFG PROBLEMS

To solve a set of games in which their solutions affect one another, no game can be addressed in isolation. Otherwise, a single-game optimal outcome might drive the overall solution (in our case, a sequence of coalition structures) away from an optimal result; or even worse, end up with an unfeasible solution. The coalition formation literature has already pointed out that to solve a single coalition structure generation problem is $\boldsymbol{F\Delta_2^P}$-complete (Greco and Guzzo, 2017). Exact algorithms (e.g., (Michalak et al., 2015)) were proposed to solve this problem, but due to its complexity, only small instances can be solved by them (in general, around 30 agents). Given this discouraging prospect, we seek to understand how bad it can be to compute optimal solutions to SCFG instances. This is an important step to further comprehend the problem and come up with alternatives, e.g., heuristic approaches to solve reasonably large instances, or to discover tractable subclasses of the problem.

In this chapter, we analyse both the SCFG and the SEQVS problems. We propose in Section 4.1 two exact algorithms to solve SEQVS instances. The first one is a trivial, but necessary, brute-force algorithm. It allows us to establish a baseline for comparisons in this challenging problem. Then, we apply a dynamic-programming technique on the search for an optimal FCSS. This approach results in an algorithm named SDP. Our next step in Section 4.2 is to experiment with both algorithms in general settings in order to compare them regarding running time and memory consumption. In Section 4.3, we discuss the challenges that arise while computing solutions for such problems. We then, in Section 4.4, embark on the computational complexity of solving SCFG. We show that this problem lays in the class of **PSPACE**-complete problems.

4.1 An Exact Approach to Find Sequences of Coalition Structures

An optimal FCSS can be computed by searching exhaustively the search space containing sequences of coalition structures. The question is how to perform that search, and whether there are ways to improve it. In the sections below we shall address these topics.

4.1.1 Preliminaries

We note that both SCFG and SEQVS are built upon a given binary relation on the set of all coalition structures; that is, $\mathcal{R} \subseteq \boldsymbol{\mathcal{CS}}^A \times \boldsymbol{\mathcal{CS}}^A$. Therefore, we say that both SEQVS and SEQSVS are SCFG-based problems. This means they share $\mathcal{R}$-related properties and we can exploit them to design mechanisms that work for all of them. For instance, an algorithm that solves SCFG problems can be adapted to take into account also VSs of a SEQVS game.

In Chapter 3, we introduced the idea of starting a sequence of CSs with $\emptyset$ (i.e., it is always the case that $CS_0 = \emptyset$). This simple modelling allow us to constrain the initial position of a sequence to potentially few options (depending on the application domain). One can easily identify them by looking up in $\mathcal{R}$ pairs of the form $(\emptyset, CS) : CS \in \boldsymbol{\mathcal{CS}}^A$.

Moreover, we note that to enlarge a sequence of CSs $\boldsymbol{CS}$, one needs to find the pairs of CSs in $\mathcal{R}$ that match the last element of $\boldsymbol{CS}$. This allows us to construct a set of CSs that are feasible regarding a preceding CS CS (we use the words feasible and compatible interchangeably).

Definition 18 (X^{CS}). Given a coalition structure CS and a binary relation $\mathcal{R} \subseteq \boldsymbol{\mathcal{CS}}^A \times \boldsymbol{\mathcal{CS}}^A$, we construct the set X^{CS} containing all coalition structures that may follow CS according to $\mathcal{R}$. That is, $X^{CS} = \{CS' \in \boldsymbol{\mathcal{CS}}^A \mid CS \; \mathcal{R} \; CS'\}$.

In case additional constraints per level are given, then we need to take into account the particular level l of the constraints in Definition 18. We remind the reader that in a SEQVS game, the allowed coalition structures at level l are given by $\boldsymbol{\mathcal{CS}}^{\sigma_l}$; similarly, $\boldsymbol{\mathcal{CS}}^{\pi_l}$ in SEQSVS context.

Definition 19 (X_l^{CS}). Given a coalition structure CS and a particular level l of the sequence, $1 \leq l \leq h$, we construct the set X_l^{CS} containing all coalition structures CS' that:

1. $CS \; \mathcal{R} \; CS'$; and
2. $CS' \in \boldsymbol{\mathcal{CS}}^{\sigma_l}$; similarly, $CS' \in \boldsymbol{\mathcal{CS}}^{\pi_l}$.

It will be clear from the context whether we are considering a SEQVS or a SEQSVS problem. Condition (2) means, in a SEQVS game, that 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 (see Definition 14). For a SEQSVS game, the above conditions hold plus the fact that the size of C must be allowed (see Definition 17).

Again, we note that all three games define $\mathcal{R}$ on $\boldsymbol{\mathcal{CS}}^A$. Therefore, the properties shown to hold in SCFG also hold for SEQVS and SEQSVS. In the remainder of this section we shall consider only SEQVS problems.

4.1.2 A Brute-force Algorithm

To search for an FCSS, we need to consider: (i) the number of agents and games; (ii) the valuation structures; and (iii) the binary relation $\mathcal{R}$. The idea of our brute-force algorithm is to construct an FCSS iteratively, and at each iteration, to check the feasibility of introducing a new CS CS' at level l in the sequence. The feasibility here is given in Definition 19. To search through the search space, we use the procedure in Algorithm 4.1. Once a sequence of CSs is complete, that is, the length of the candidate FCSS matches the number of games in $\mathcal{H}$, we compare its value with the best FCSS found so far and keep the one with the greatest value.

Algorithm 4.1 A brute-force algorithm.

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

```
1: for all CS ∈ X_1^∅ do
2:     BRUTEFORCE_SEQUENCE(⟨CS⟩)
3: return CS*
4: procedure BRUTEFORCE_SEQUENCE(CS)
5:     if |CS| = h then                                  ▶ sequence is complete
6:         if V(CS) > V(CS*) then
7:             CS* ← CS
8:         return
9:     l ← |CS|
10:    CS ← CS[l]
11:    for all CS' ∈ X_{l+1}^{CS} do
12:        BRUTEFORCE_SEQUENCE(CS · ⟨CS'⟩)               ▶ concatenation operation
13:    return
```

4.1.3 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 $\mathcal{R}$). Therefore, our problem displays ***optimal substructure*** regarding CS CS, because we can determine an optimal subsequence of coalition structures to CS, then use memoisation to keep such partial results. Doing so, we avoid recomputing the optimal subsequences when we need to try the various possible next CS CS$'$ that may follow each of those subsequence. Consider as an example Figure 4.1. Once we reach level i and evaluate coalition structure $CS' = \{\{a_1\}, \{a_2, a_3\}\}$, the CSS $\boldsymbol{CS}'$ does not change regarding CS'. Thus, the best we can do to compute a subsequence ending at CS CS', is to get a subsequence $\boldsymbol{CS}'$ of length $i - 1$ that maximises the subsequence value and keep it in memory.

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. Consider again the example in Figure 4.1. This intuition is applied to the remaining sequences until we reach level h. Once we do, the number of FCSS will be bounded by the number of CSs in $\boldsymbol{\mathcal{CS}}^{\sigma_h}$; each containing a single optimal subsequence in memory. To return an optimal solution, we just pick the one maximising the value among the options. We give a recursive definition of the value of an optimal FCSS below.

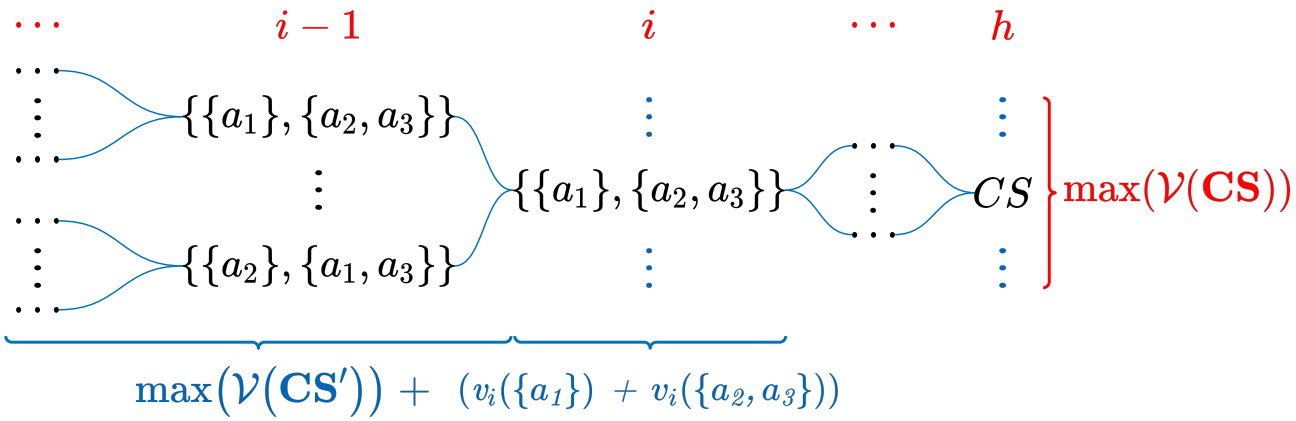


Figure 4.1: An example of optimal substructure regarding coalition structure $\{\{a_1\},\{a_2,a_3\}\}$.

Lemma 7. 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 7, 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 4.2 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.

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 4.1.2). 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.

Algorithm 4.2 The SDP algorithm.

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)]
```

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 $\mathrm{f}'[CS]$ and $\mathrm{t}'[CS]$. 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 7, 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 $\arg\max \mathrm{f}$. □

We note that the optimal substructure property discussed above is inherent to SCFG instances because its is built on the top of $\mathcal{R}$. Therefore, SDP is also a suitable tool to solve any problem related to SCFG.

4.2 Experiments

In this section, we experiment with the two algorithms introduced in Section 4.1. We implemented them using Python (version 3.8) and conducted all experiments in a virtual machine with 32 GB of RAM and a CPU with four single cores of 2095 MHz each. The results below were originally reported in (Krausburg et al., 2021a). All material is available at https://github.com/smart-pucrs/SCFG.

4.2.1 Preliminaries

We experiment with SEQVS instances containing a small number of agents and games. Precisely, we range from $2, \ldots, 10$ agents and also compute solutions for sequences containing 2, 4, 8, and 10 games. Even for those small instances, we set a ***timeout*** of one hour for the algorithms to output a solution. Regarding the metrics for our comparison, both algorithms output a solution of same quality, therefore, we compare the running time and memory consumption.

For each CFG $\Gamma \in \mathcal{H}$, we draw a value for any coalition $C \subseteq A$ from $v(C) \sim N(\mu, \sigma^2)$ where $\mu = |C|$ and $\sigma = \sqrt{|C|}$. This distribution is called NDCS and was introduced by Rahwan et al. (2009b). We use NDCS because it has the property of letting all coalition structures in $\boldsymbol{\mathcal{CS}}^A$ to have the same probability of being an optimal one. This way, the search space is not biased towards a particular subspace (e.g., the more agents in a coalition, the greater the value). In all experiments we use a table to store in advance the values for all coalitions. Each game assumes a different valuation drawn from the distribution. That means, we are keeping in memory h tables which store the value of every single coalition.

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.

We consider four binary relations as given below.

Definition 20 ($\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$). 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 $CS = CS'$.

$\mathcal{R}_3$ models a hierarchical group structure, which is important for various application scenarios. As an example, consider Figure 6.1. The first two relations (representing constraints that are quite general) are used in our empirical evaluation as worst-case scenarios. $\mathcal{R}_1$ forces us to compare a CS with the maximum number of compatible 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. All four relations allow every CS to begin an FCSS: for every $CS \in \boldsymbol{\mathcal{CS}}^A$ 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* algorithms for all four relations. Those algorithms compute each set X^{CS} (without taking into account the valuation structures) for both brute force (short bf in the charts) and SDP algorithms.

4.2.2 Time Analysis

We depict the running time results in Figure 4.2. We see that SDP is faster by several orders of magnitude in $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ if compared with the brute force approach for any h. This shows that the memoisation technique applied in SDP plays an important role while computing FCSSs. In $\mathcal{R}_4$, as only one candidate is possible for the next position in the sequence, both algorithms had a similar running time. That relation means for SDP that only a single subsequence is available for every CS in every position i. Similar behaviour 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.

4.2.3 Memory Analysis

In this experiment, we record the peak of memory used by each algorithm. It stands for the amount of memory we would have to have available to run them. For that purpose, we use the Python package `malloc-tracer` (version 1.7.0)[1] to capture the memory consumption of the process that is running the algorithm instance. We depict the results in Figure 4.3.

Note that brute-force algorithm and SDP swap their positions in comparison with the running-time experiments. As expected, SDP consumes more memory, by several orders of magnitude, than the brute-force approach. We also note that the amount of memory needed

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

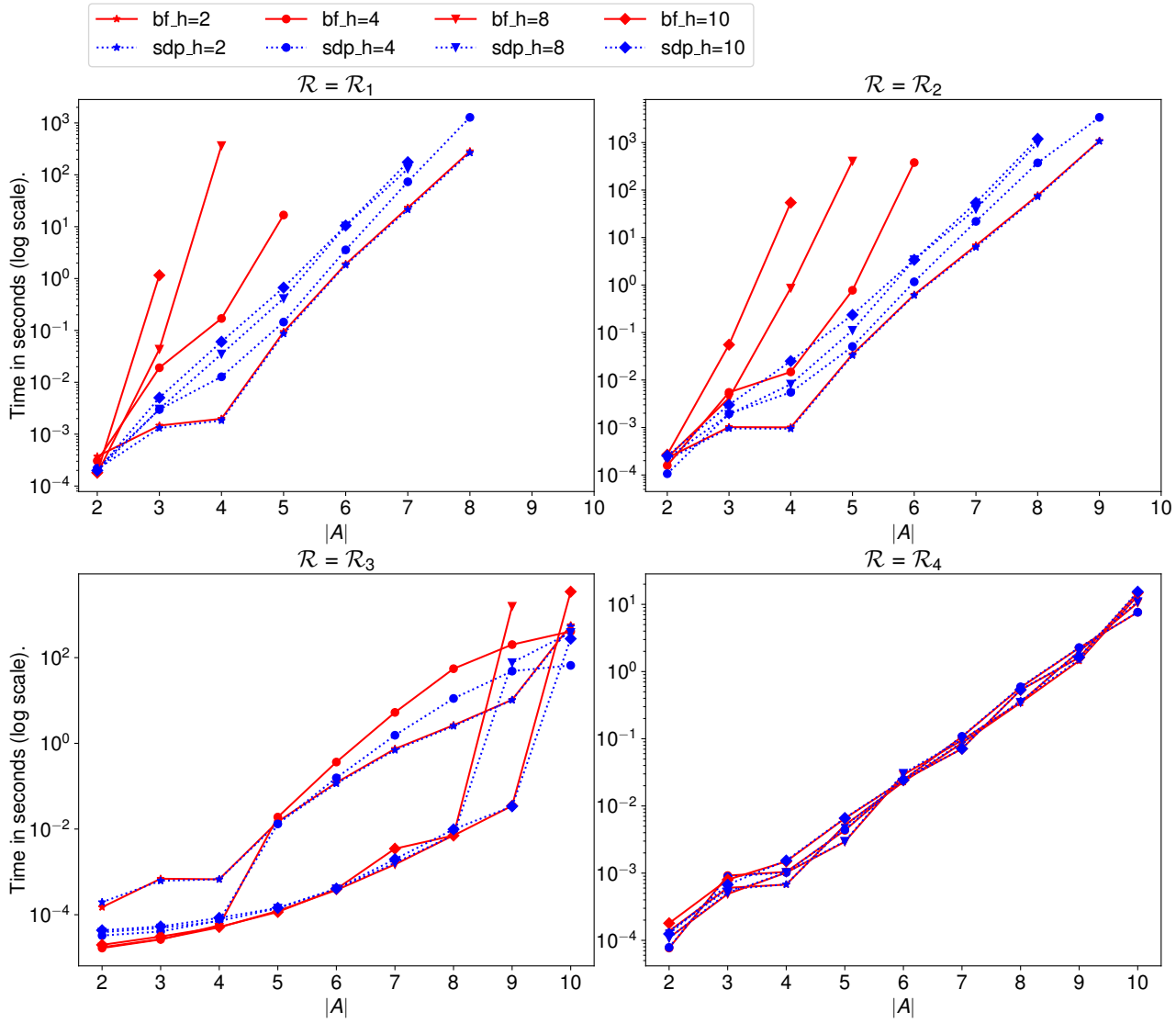


Figure 4.2: Comparison of running time in log scale between the brute force (bf) and SDP algorithms. In case a timeout is fired, we do not depict the corresponding point in the $|A|$ axis for that particular h. For instance, line `bf_h=8` when $\mathcal{R} = \mathcal{R}_1$ had timeouts from $|A| = 5$ to $|A| = 10$.

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.

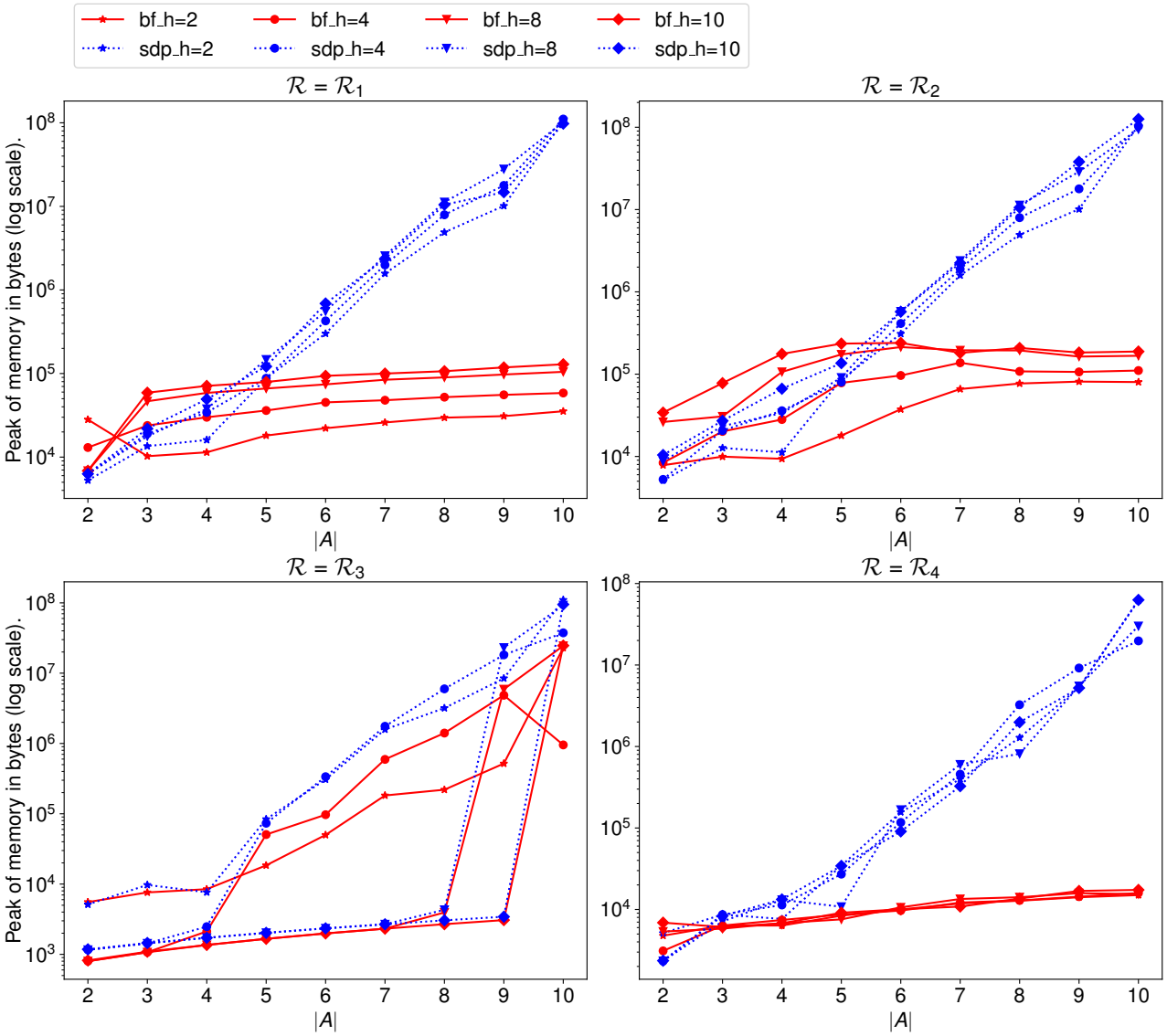


Figure 4.3: Comparison of peak of memory in log scale between the brute force (bf) and SDP algorithms.

4.3 Discussion

In the experiments reported above, we can clearly see that SDP is the right choice over the brute-force approach to be used as bottom line for comparisons in experiments, despite its use of memory. We do not expect SDP (and possibly any exact algorithm for SCFG problems) to be employed in a wide range of applications due to the fact it has to go through the whole space of coalition structures, in particular if there is no narrow number of CSs that may start a sequence. However, if the problem at hand is small enough and the first games are supposed to produce a small number of CSs, then it might be used in practice.

In future work, another interesting exact approach to investigate for SCFG problems regards branch and bound techniques. In this paradigm, one partitions the search space into subspaces and bounds them in order to prune (i.e., avoid) unpromising subspaces. An example

of this technique being applied to an algorithm that solves the CSG problem is found in (Rahwan et al., 2009b). In case the search space is biased, for instance, small coalitions are evaluated to greater values, a branch and bound mechanism may prune many CSs. A main challenge is then to come up with a good strategy to partition the search space.

In both Algorithms 4.1 and 4.2, we assume generative algorithms that can precisely generate all CSs that may follow a given one. That is, they build efficiently the set $\mathcal{R}$. This is a strong assumption given the fact that generating this set is a difficult part of the overall problem. Moreover, efficient generators might mean application-dependent generators being applied to a specific problem. Further work on problems related to SCFG should design generators that are application-independent by exploiting the structure of the framework itself. In the next chapter, we take the first step towards solving this particular problem for SEQSVS settings.

Both SCFG and VS pose constraints in the (sequence of) coalition structures that we can be formed. A SCFG addresses the sequence dimension and VSs (or SVSs) the position-specific dimension. Note that whether the VSs or the relation $\mathcal{R}$ of an SCFG is more restrictive depends on the specific domain. For some classes of applications, $\mathcal{R}$ could potentially be more restrictive than the interaction graphs plus pivotal agents and size constraints. In the experiments reported in this chapter, we considered the approach to generate the pairs of coalition structures in $\mathcal{R}$ and, after that, to check the constraints of the corresponding VS. In the next chapter, we shall consider the other way around as well: VS being used to generate CSs that are then filtered out by $\mathcal{R}$. However, no in-depth comparison between those two approaches is carried out within this thesis. We leave it as future work further experimentation and to assess the implications that choice.

4.4 Complexity

Given a SCFG $\mathcal{G} = \langle A, \mathcal{H}, \mathcal{R} \rangle$ and a number k, we can ask whether the SCFG instance has an FCSS solution $\boldsymbol{CS}$ with value at least k. In this section, we show that this decision problem as formalised below is **PSPACE**-complete.

Definition 21 (SEQCFG). We define the decision problem SEQCFG as the following language:

$$\{\langle \mathcal{G}, k \rangle \quad | \quad \mathcal{G} \text{ is an SCFG for which there exists an FCSS } \boldsymbol{CS} \text{ with } \mathcal{V}(\boldsymbol{CS}) \geq k\}.$$

Theorem 2. SEQCFG is **PSPACE**-complete.

Proof. To show that SEQCFG $\in$ **PSPACE**, consider a depth-first algorithm that systematically checks all feasible sequences until one is found with value at least k. Using $\mathcal{R}$ we can expand all sequences systematically and search the whole tree exhaustively. Consider Algorithm 4.1, for instance. We only need to store the currently being checked sequence of feasible coalition

structures and some constant space to check the relation $\mathcal{R}$ for the next element (if $\mathcal{R}$ is compactly represented and needs unfolding as previously discussed). So the space complexity is $\mathcal{O}(|A| \lg |A| \times |\mathcal{H}|)$.

To show **PSPACE**-hardness, we give a polynomial-time reduction from *Generalised Geography* GG to SCFG (GG is **PSPACE**-complete (Lichtenstein and Sipser, 1980)).

Given a GG instance $\langle G^{\text{GG}}, u_1 \rangle$ where $G^{\text{GG}} = (V, E)$ is a directed graph and u_1 the initial vertex, we generate an SCFG $\mathcal{G} = \langle A, \mathcal{H}, \mathcal{R} \rangle$ as follows. Let the set A be defined as $A = V \cup \{p_1, p_2, p_3, p_4, \otimes\}$, where $p_1, \ldots, p_4$ and $\otimes$ are special agents with the specific purpose explained below. The agents p_1 and p_2 represent Players I and II respectively. Agent p_3 is the agent used to mark the coalition used to keep track of vertices that have not yet been visited by any player, while p_4 is the agent of the coalition for the vertices that have already been visited. Agent $\otimes$, when in a coalition with agents p_1 or p_2, is used with the specific purpose of denoting that a player has no further moves allowed. We use C_I, C_{II}, C_O (open nodes), and C_C (closed nodes) as shorthand for sets of agents to which, respectively, the agents p_1, p_2, p_3, and p_4 belong.

Each turn for any of the two players is represented by a game. Therefore, we have at least $|V|$ games in $\mathcal{H}$ to represent the worst-case scenario in which all vertices are chosen. However, we use $\otimes$ to denote that a player has no further moves thus, we require an additional game in $\mathcal{H}$. This last game represents the fact that one of the players ***must*** choose $\otimes$; that is, both players have no more moves available. Note that in the worst case there will be as many turns as vertices in G^{GG} plus one; shorter GG games will have the last move copied over repeatedly until the last game in the SCFG game. Therefore, $h = |V| + 1$.

Now, consider the binary relation $\mathcal{R}$. We cannot assume it to be the set of exhaustively generated pairs of coalition structures that follow the rules of the GG problem. This would require exponential time to construct such a set for a given GG instance. Instead, we assume $\mathcal{R}$ to be given as a generative algorithm (see Section 4.3), in which given a CS, it generates all CSs that may follow it. To do so, we just need to keep a copy of the GG instance $\langle G^{\text{GG}}, u_1 \rangle$ in memory and follow a few rules. Below we give such an algorithm, but first, consider Figure 4.4. It contains a simple example of a GG instance reduced to an SCFG instance. We also show the subsequent CSs generated by $\mathcal{R}$.

To make it general, initially (i.e., at the first SCFG game), the only coalition structure in the domain of $\mathcal{R}$ is $\{\{p_1\}, \{p_2, u_1\}, \{p_3, u_2, u_3, \ldots, u_k, \otimes\}, \{p_4\}\}$; note u_1 is the vertex where Player I makes the initial move from. We use $\emptyset$ as the first element of the pair containing that CS to enforce it. The next coalition structure in the allowed sequences must represents a move from Player I, choosing for instance u_2, hence $\{\{p_1, u_2\}, \{p_2\}, \{p_3, u_3, u_4, \ldots, u_k, \otimes\}, \{p_4, u_1\}\}$, and so forth. To denote that no further moves are allowed to player j, the coalition $\{p_j, \otimes\}$ appears in the codomain of $\mathcal{R}$. In case a winning strategy does not require all nodes in V to be used, the last game's outcome is replicated until game h (e.g., the CS in blue in Figure 4.4).

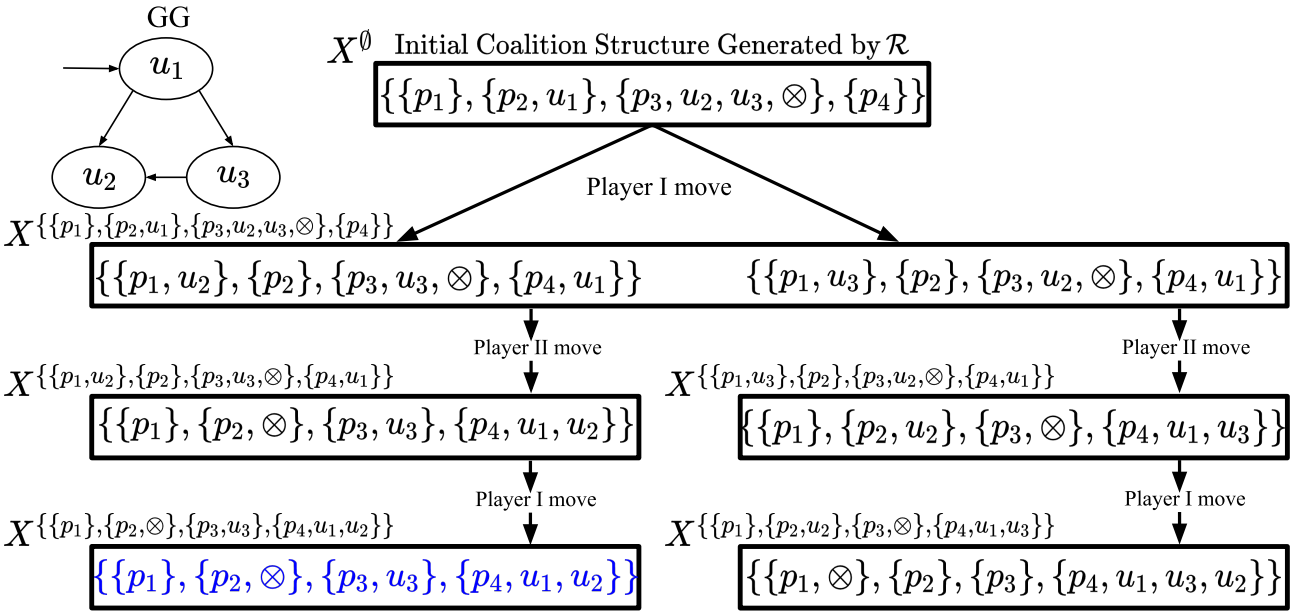


Figure 4.4: Generating a SCFG from a GG instance.

The following rules determine $\mathcal{R}$ by generating the feasible coalition structures at the successor level[2], that is, build the set X^{CS}. In the first game only one solution is feasible: the initial vertex where Player I makes its first move move from. Then, for each CS in $\boldsymbol{\mathcal{CS}}^A$:

1. if $\otimes \in C_I, C_I \in CS$, then for every $CS' \in X^{CS}$ we have $\otimes \in C'_I : C'_I \in CS'$. Similarly, for $\otimes \in C_{II}$.

2. if $CS = \{\{p_1, \hat{u}\}, \{p_2\}, C_O, C_C\}$ (i.e., one possible move was for Player I to move to vertex $\hat{u}$), then for each u_k such that $(\hat{u}, u_k) \in E$, $\{\{p_1\}, \{p_2, u_k\}, S_O \setminus \{u_k\}, S_C \cup \{\hat{u}\}\} \in X^{CS}$ (that is, each u_k is one possible response movement by Player II);

3. for the same CS pattern as in the item above, if there is no u_k such that $(\hat{u}, u_k) \in E$ then $\{\{p_1\}, \{p_2, \otimes\}, C_O, C_C \cup \{\hat{u}\}\} \in X^{CS}$ (that is, Player II got stuck);

4. for $CS = \{\{p_1\}, \{p_2, \hat{u}\}, C_O, C_C\}$, there are two more rules that are very similar to the two items above except that it is Player I rather than II that is moving.

Regarding the characteristic functions for the $1, \ldots, h-1$ games: they simply return 0 for all coalitions. In game h, v_h returns value 0 for all coalitions except $\{p_2, \otimes\}$, which is valued 1. Doing so, at game h we properly evaluate a winning strategy: if Player II has no further moves, coalition $\{p_2, \otimes\}$ is in the CS at level h, then Player I has a winning strategy. Otherwise, Player II has a winning strategy.

The SCFG instance generated with the help of the above procedure for a GG instance $\langle G^{\text{GG}}, u_1 \rangle$ is $\langle \mathcal{G}, k \rangle$ where $\mathcal{G}$ is an SCFG as described above and $k = 1$ is the target.

The reduction can be clearly done in polynomial time given that:

[2]Note that we cannot generate $\mathcal{R}$ exhaustively otherwise the reduction would not take polynomial time.

- $|\mathcal{H}|$ is at most the number of vertices in G^{GG} plus one;
- $|A|$ is the number of vertices plus the five special agents;
- the characteristic functions can be compactly represented as stated above; and
- $\mathcal{R}$ can also be compactly represented as an algorithm that follows the 4 rules exactly as stated above on the representation of the graph G^{GG} (which is kept in memory). Although this does not affect the reduction, note that determining $\mathcal{R}$ on-the-fly is also quite efficient in this particular case (linear in the degree of G^{GG}).

□

Note that, in a similar reduction but between the optimisation versions of the problems we could easily change the characteristic functions so that an optimal FCSS provides us the means to determine the winning strategy for Player I with the least number of moves.

Chapter Remarks

In this chapter, we investigated the challenges in solving optimally SCFG-based problems. We started out by proposing two exact algorithms to output an optimal Feasible Coalition-Structure Sequence (FCSS). A brute-force algorithm provides the baseline for comparisons on this problem. Our main contribution here is an algorithm based on dynamic programming called SDP. We showed empirically that SDP is faster by orders of magnitude compared with a trivial brute-force algorithm. However, the trade-off comes in terms of memory consumption in which SDP requires much more memory than the brute-force approach. Both algorithms in our experiments computed solutions for small instances of the problem only (up to 10 agents and 10 games), which narrows down their use in practical applications. This concludes Contribution 4. We noted that to solve an SCFG-based problem, an efficient method is to design an application-dependent generator of pairs of CSs. Even in those cases the complexity of solving SCFG lays in the class of **PSPACE**-complete problems, hence concluding Contribution 5. Given those findings, in the next chapter we investigate heuristic approaches to solving SCFG-based problems.

5. HEURISTIC APPROACHES TO SCFG

Our main goal in this chapter is to propose heuristic approaches to solve the problem of finding a sequence of feasible coalition structures. This problem is already intractable for small instances (as discussed in Section 4.3) and therefore, we need to find algorithms that can produce solutions quickly. Moreover, the challenge is also how to find those solutions as depending on $\mathcal{R}$ and the level constraints only a few might exist. The algorithms we propose here are designed to solve SEQVS and SEQSVS instances. However, as discussed in Chapter 4, both SCFG and its variations (i.e., SEQVS and SEQSVS) share the main underlying concept: the relation $\mathcal{R}$. Therefore, one can easily adapt the proposed algorithms to deal with any SCFG-based problem.

This chapter is structured as follows. In Section 5.1, we propose a hierarchical-clustering algorithm, inspired by C-Link, for solving SCFG instances. We refer to Section 2.3.3 for the description of how C-Link works. Our next step is to investigate how to apply Monte Carlo Tree Search to the SCFG problem. Our main strategy explained in Section 5.2 is to assign to each node in the tree a coalition structure, and then find a path corresponding to an FCSS. Doing so, we avoid local optima as we eventually explore different paths in the tree. Finally, in Section 5.3, we extensively evaluate both algorithms under general conditions. We compare their solutions with the one computed by SDP, although only for small instances. Then, we scale up the size of the instances to compare their performance in more challenging scenarios.

5.1 Hierarchical Clustering Algorithm to Compute an FCSS

We take as inspiration the clustering-based algorithm named C-Link (Farinelli et al., 2016), and construct a new algorithm called MC-Link (Krausburg et al., 2021b). MC-Link was originally proposed to address SCFG instances only. However, in (Krausburg et al., 2021a) we extended it to deal with SEQVS instances as well.

5.1.1 The Multiple Coalition Linkage Algorithm

The MC-Link algorithm follows the same general idea adopted for C-Link (Section 2.3.3); it starts off with the coalition structure of singletons and merges two coalitions based on a function that measures the suitability of such a merger. Nonetheless, in our context we have a sequence of CSs of singletons. The underlying intuition is that each game Γ_i has a corresponding table PL_i. We perform the most suitable movement for a table PL_i (i.e., a merger), then advance to the next table PL_{i+1} and repeat the process (see Figure 5.1). Doing so, we are able to construct the tables PL (one at time) whilst enforcing the relation $\mathcal{R}$, which is also given

PL_1^t	a_1	a_2	a_3	a_4
a_1	-∞	5	-5	5
a_2	--	-∞	-15	-10
a_3	--	--	-∞	10
a_4	--	--	--	-∞

PL_2^t	a_1	a_2	a_3	a_4
a_1	-∞	5	-5	5
a_2	--	-∞	-15	-10
a_3	--	--	-∞	-∞
a_4	--	--	--	-∞

PL_1^{t+1}	a_1	a_2	a_3,a_4
a_1	-∞	-∞	-20
a_2	--	-∞	-15
a_3,a_4	--	--	-∞

PL_2^{t+1}	a_3	a_4	a_1,a_2
a_3	-∞	-∞	-15
a_4	--	-∞	0
a_1,a_2	--	--	-∞

Figure 5.1: An example run with four agents for MC-Link. The red ovals represent the selected column and row to be merged, and t represents an iteration. MC-Link received as input $\mathcal{R}_H$ (Definition 9), and a sequence of two CFGs; the games use the same characteristic function.

as input. As we construct CSs based on mergers, we only need an algorithm that checks the feasibility of a pair of CSs in $\mathcal{R}$.

An algorithm for SCFG needs to take into account that:

1. A CSS $\boldsymbol{CS}$ may not be feasible right from the beginning. In MC-Link, all CSs start off with singleton coalitions which may not be acceptable by $\mathcal{R}$ (e.g., $\mathcal{R}_3$ in Definition 20). In fact, a CSS might become feasible only after several iterations of MC-Link.

2. To calculate a value for an FCSS, we must consider the sequence as a whole. For instance, consider $\mathcal{R}_3$ and $\mathcal{R}_H$. $\mathcal{R}_3$ generates constraints being applied in only one direction (top to bottom). On the other hand, $\mathcal{R}_H$ leads to constraints being imposed in both directions.

Condition 1 suggests that we need to look for an FCSS even if that would mean choosing a non-suitable merger (i.e., choosing zero or negative values). Condition 2 indicates that the search is conducted in rounds, and at each round only *one merger per table* is carried out.

We show in Algorithm 5.3 the pseudocode for MC-Link[1]. It starts by initialising a CSS $\boldsymbol{CS}$ with as many coalition structures of singletons as there are games in $\mathcal{H}$. Then, we start looking for a CSS that is feasible according to $\mathcal{R}$ (lines 3-16).

We check the feasibility condition using the Boolean function FEASIBLE (e.g., line 3). It checks whether all pairs of subsequent CSs in a candidate solution are in $\mathcal{R}$. In case an index is provided (e.g., line 34), we check if CS $\widehat{CS}$ at that index is feasible in relation to both the preceding and succeeding positions in the sequence.

[1]Note that in (Krausburg et al., 2021b) we did not introduce the possibility of stating coalition structures that may begin an FCSS (i.e., $(\emptyset, CS) \in \mathcal{R} : CS \in \boldsymbol{CS}^A$). To keep compatibility with the original work, we do not change MC-Link algorithm for SCFG problems, although it can be easily done.

Algorithm 5.3 The multiple coalition linkage algorithm.

Input:
A, a set of agents
$\mathcal{H}$, a sequence of CFGs
$\mathcal{R}$, a binary relation on $\boldsymbol{\mathcal{CS}}^A$

Output: $\boldsymbol{CS}$, an FCSS

$\boldsymbol{CS} \leftarrow \langle \{\{a_1\}, \ldots, \{a_n\}\}_1, \ldots, \{\{a_1\}, \ldots, \{a_n\}\}_h \rangle$
$\hat{M} \leftarrow \langle \infty_1, \ldots, \infty_h \rangle$
if $\neg$ FEASIBLE($\boldsymbol{CS}$, $\mathcal{R}$) **then**
  $i \leftarrow 1$
  while $i \leq h - 1$ **do**
    $PL \leftarrow$ FILLTABLE$(i, \boldsymbol{CS}, v_i)$
    $\hat{M}[i] \leftarrow \max_{j,k} PL(j,k)$
    if $\hat{M}[i] > -\infty$ **then** ▶ merger is available
      $\hat{j}, \hat{k} \leftarrow \arg\max_{j,k} PL(j,k)$
      $\boldsymbol{CS}[i] \leftarrow (\boldsymbol{CS}[i] \setminus \{C_{\hat{j}}\} \setminus \{C_{\hat{k}}\}) \cup \{C_{\hat{j}} \cup C_{\hat{k}}\}$
      $i \leftarrow i + 1$
    else
      if $i = 1$ **then**
        return $\emptyset$ ▶ No FCSS could be found for $\mathcal{R}$
      else
        $i \leftarrow i - 1$
$i \leftarrow 1$
while $\max_{\hat{M}} > 0$ **do**
  $PL \leftarrow$ FILLTABLE$(i, \boldsymbol{CS}, v_i)$
  $\hat{M}[i] \leftarrow \max_{j,k} PL(j,k)$
  if $\hat{M}[i] > 0$ **then** ▶ a suitable merger can be carried out
    $\hat{j}, \hat{k} \leftarrow \arg\max_{j,k} PL(j,k)$
    $\boldsymbol{CS}[i] \leftarrow (\boldsymbol{CS}[i] \setminus \{C_{\hat{j}}\} \setminus \{C_{\hat{k}}\}) \cup \{C_{\hat{j}} \cup C_{\hat{k}}\}$
  $i \leftarrow i + 1$
  if $i > h$ **then**
    $i \leftarrow 1$
return $\boldsymbol{CS}$
procedure FILLTABLE$(i, \boldsymbol{CS}, v)$
  $s \leftarrow |\boldsymbol{CS}[i]|$
  let PL be a $s \times s$ matrix initialised with $-\infty$
  for $j \leftarrow 1$ **to** s **do**
    for $k \leftarrow j + 1$ **to** s **do**
      $\widehat{CS} \leftarrow (\boldsymbol{CS}[i] \setminus \{C_j\} \setminus \{C_k\}) \cup \{C_j \cup C_k\}$
      if FEASIBLE$(i, \widehat{CS}, \boldsymbol{CS}, \mathcal{R})$ **then**
        $PL(j,k) \leftarrow slf(v, C_j, C_k)$ ▶ Equation 5.1
  return PL

In the beginning, if the sequence is not feasible, then we go through the tables PL merging two coalitions even if there is no suitable merger to carry out (e.g., all have a negative value). In this phase, a merger is considered impossible if it is constrained by relation $\mathcal{R}$ (i.e., the respective entry in table PL receives $-\infty$). In that case, we go back to the prior position in the sequence, construct the table, and repeat the procedure. If a merger is possible, then the new entry value is computed using Equation 5.1. Note that MC-Link selects the correct characteristic function based on the table it is evaluating at the moment.

$$slf(v, C_i, C_j) = v(C_i \cup C_j) - v(C_i) - v(C_j) \tag{5.1}$$

Once a first FCSS is found, we go to the next phase, improving it (lines 18-26). We iterate over the sequence, performing one merger per table, but now, we consider only suitable mergers (i.e., values greater than zero). If no such value is available in any table, then the algorithm returns the current FCSS up to that point (line 27).

5.1.2 MC-Link Analysis

By Condition 1, we know a CSS $\boldsymbol{CS}$ may not be feasible from the beginning of an execution, and its feasibility will depend on $\mathcal{R}$. However, we show that, if such FCSS $\boldsymbol{CS}$ exists, MC-Link will be able to find it for a particular set of relations, namely $\mathcal{R}_H$, $\mathcal{R}_O$, and $\mathcal{R}_3$. We pick those three relations as they are inspired by real-world applications.

Theorem 3. Given an SCFG $\mathcal{G} = \langle A, \mathcal{H}, \mathcal{R} \rangle$, in which $\mathcal{R} \in \{\mathcal{R}_H, \mathcal{R}_O, \mathcal{R}_3\}$, MC-Link eventually outputs an FCSS $\boldsymbol{CS}$, if one exists.

Proof. For relation $\mathcal{R}_H$ this result immediately follows: the CSS containing only coalition structures of singletons is feasible. For $\mathcal{R}_3$, we start off from a CSS $\boldsymbol{CS}$ containing only singleton coalitions. The CS $CS_1 \in \boldsymbol{CS}$ will dictate how CS_2 is to be constructed. That is, it is not constrained by any other coalition structure in $\boldsymbol{CS}$ and hence for its table PL_1 the condition $\max_{j,k} PL_1(j,k) > -\infty$ holds; thus, a merger is carried out. In fact, for each table PL_i, $1 \leq i \leq h$, at most $n - i$ mergers can be carried out (recall mergers of negative or zero gain are allowed). As $h \leq n$, under $\mathcal{R}_3$ an FCSS $\boldsymbol{CS}$ is eventually found. For $\mathcal{R}_O$, we have $\mathcal{R}_O \subset \mathcal{R}_3$. □

Based on the theorem above (and hence the relations above), we can claim that MC-Link has two interesting properties:

Convergence: MC-Link converges to a solution. This is due to the fact that at most $n-1$ mergers can be done per table PL. As we need to evaluate h tables, our algorithm provides an outcome after at most $n \times h$ mergers. This applies to any binary relation on $\boldsymbol{\mathcal{CS}}^A$.

Anytime: As soon as MC-Link finds a CSS that satisfies relation $\mathcal{R}$, it becomes an anytime algorithm, since its solution from that moment on will only be improved at each iteration until no more mergers between coalitions are feasible.

Regarding the complexity of MC-Link to compute a solution for an SCFG problem, we shall assume any $\mathcal{R}$ to be given in an algorithmic form. Moreover, let α and β, in the result below, denote its time and space complexity, respectively.

Theorem 4. The time complexity of MC-Link is $\mathcal{O}(h^2n^3\alpha)$, and the space complexity is $\mathcal{O}(n^2 + hn + \beta)$.

Proof. MC-Link has two phases that share the same CSS $\boldsymbol{CS}$. Each $CS \in \boldsymbol{CS}$ has a corresponding table PL that can be constructed, when required, in $n^2 \times \alpha$ steps. To make $\boldsymbol{CS}$ feasible, MC-Link needs at most time $\mathcal{O}(hn^3\alpha)$, as we need at most $(h-1) \times (n-1)$ steps to find an FCSS. To improve an FCSS $\boldsymbol{CS}$, as the worst-case scenario, we assume that the CSS of singletons is feasible, so no merger was carried out. Then, $\mathcal{R}$ makes it possible only one merger per iteration (from 1 to h). That is, we reach the grand coalition in a single table in $h \times (n-1)$ iterations. As we have h tables, the time complexity of MC-Link is $\mathcal{O}(h^2n^3\alpha)$. However, we only need space $\mathcal{O}(n^2 + hn + \beta)$, as we evaluate one table at time and can construct it directly from a $CS \in \boldsymbol{CS}$. Although the final complexity depends on α and β, it is expected that MC-Link is typically polynomial. □

We also empirically evaluate MC-Link to assess how our algorithm performs when we vary the number of games and scale up the number of agents. As storing the value of every coalition in a table is not feasible, we set every game to be evaluated according to $v(C) = |C|^2$. We pick this valuation v based on the properties shown above. We know MC-Link converges and we want to evaluate it in the worst-case scenario, which is when all feasible movements of each table are performed. Hence, we need v to be super-additive. We scale the number of agents up to $n = 100$, and pick relation $\mathcal{R}_3$ for this particular analysis. We show the results in Figure 5.2.

Each line represents a different number of games and appears in the figure when the number of agents is sufficient (if $h > n$ there exists no hierarchy for $\mathcal{R}_3$). We can see that even with a heuristic approach, the running time increases significantly. However, even real-world problems may not require a large number of different games (e.g., the problems described in Section 3.2). In addition, the characteristic function may limit positive gains of mergers to a few coalitions, thus less operations would be required.

The reader should bear in mind that MC-Link might not compute a solution for a given problem, even though such a solution exists. MC-Link adopts a greedy strategy based on the valuation functions, and those values might not be compatible with the constraints at hand. That is, mergers that produce a great gain might lead to sequences of coalition structures that are unfeasible. Recall that MC-Link cannot undo a merger. Therefore, in problems that require a great number of constraints, one must choose carefully the valuation functions.

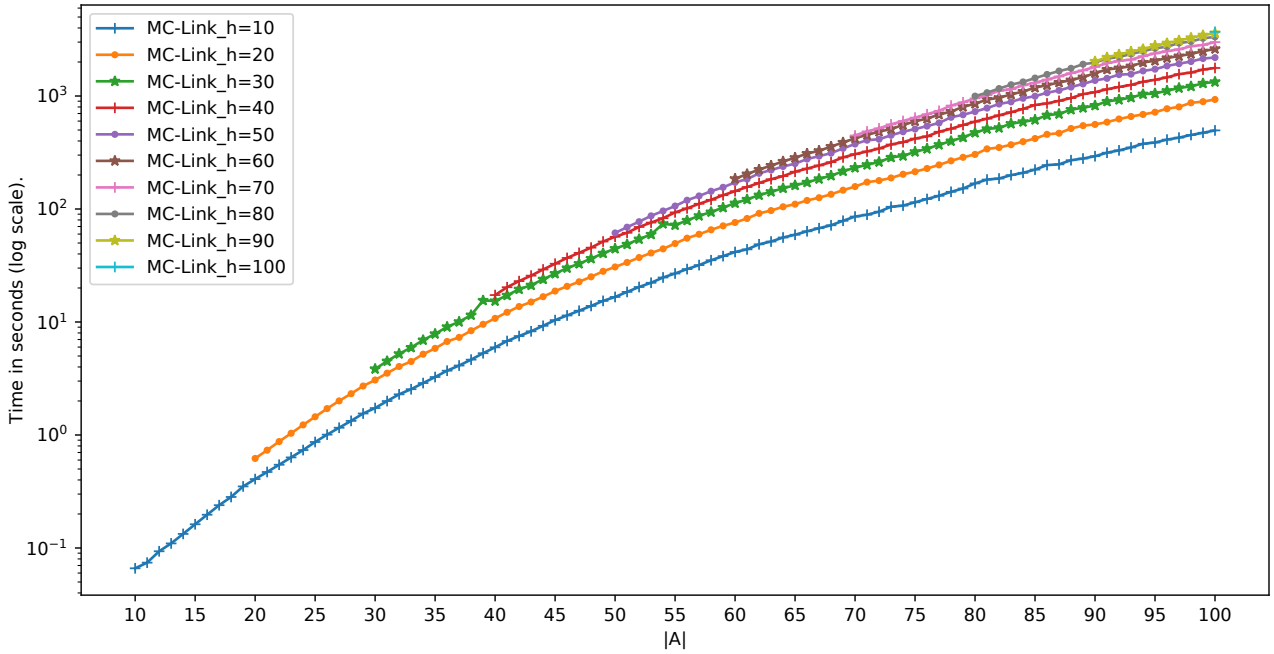


Figure 5.2: MC-Link running time for different lengths of $\mathcal{H}$ and varying the number of agents.

5.1.3 MC-Link for SEQVS instances

As originally proposed in (Krausburg et al., 2021b), MC-Link is designed to cope with SCFG instances. However, we extend it to deal with SEQVS instances as well, so as to enable comparisons with algorithms not restricted only to SCFG problems. This extension was first introduced in (Krausburg et al., 2021a) to further analyse the experiments aimed to compare SDP with the brute-force algorithm.

Apart from checking each game's valuation structure, we insert a further check to make sure a sequence of CSs holds the feasibility property. This is required to make MC-Link compatible with a wider range of binary relations. Therefore, we modify the improving phase in the original MC-Link. To understand the reason why this is the case, consider the example below.

Example 8. Consider $\mathcal{R}_2$ from Definition 20: all CSs in a sequence must have the same number of coalitions. Suppose MC-Link is improving on the FCSS containing only CSs of singletons. It carries out a merger in each CS in the sequence up to position i. In that game, no merger is allowed because the combination of pivotal agents and interaction graph makes it unfeasible; that is, for all $CS \in \boldsymbol{\mathcal{CS}}^{\sigma_i}$ it holds that $|CS| = n$. As a carried out merger in MC-Link cannot be undone, it is not the case that the current sequence of CSs will eventually become feasible (CSs $CS_1, \ldots, CS_{i-1}$ already have size $n-1$).

To cope with that requirement, we distinguish between the CSS we are modifying during a particular iteration, and the best FCSS $\boldsymbol{CS}^*$ found so far. Initially, the best FCSS is the one computed during MC-Link's first phase. Also, we insert an additional check in case

Algorithm 5.4 Modifications to MC-Link's improvement phase.

```
1: if i > 0 and not (CS[i] ∈ 𝒞𝒮^{σ_i} and (CS[i − 1], CS[i]) ∈ ℛ) then
2:     if M̂[i − 1] ≤ 0 then
3:         return CS*                          ▶ the best FCSS found so far
4:     else
5:         i ← i − 1
6: else
7:     i ← i + 1
```

no merger is allowed in the current table. Precisely, we add an else condition for Line 21 in Algorithm 5.3. The new steps are introduced in Algorithm 5.4. In case no merger is allowed at the current position i, we check if the corresponding CS is still compatible with the preceding one in the sequence. If not so, and there are no longer mergers available at position $i-1$, the CSS will never become feasible, and we then return the best FCSS found so far. Otherwise, we return to the previous CS and carry out another merger continuing with the ordinary procedure. The FCSS $\boldsymbol{CS}^*$ is updated every time we reach position h. In the example above, MC-Link would stop to iterate and return the FCSS containing only singleton coalitions, even though it had performed a merger on each $i-1$ CSs.

5.2 A Monte Carlo Approach to SCFGs

As noted in the section above, MC-Link is not complete because under certain constraints it might not be able to find an FCSS even though one exists. Thus, we investigate alternatives to that problem and decided to design and evaluate an algorithm based on Monte Carlo Tree Search (MCTS) for the SCFG problem. MCTS has received much attention in recent research due to its achievements in particular applications. For instance, MCTS is used in AlphaGo (Silver et al., 2016) in order to select an action to be played in a game called Go. It has also been applied to various problems such as intention scheduling (Dann et al., 2020), and even the coalition structure generation problem (Wu and Ramchurn, 2020).

Two MCTS steps are particularly important for an algorithm that solves SCFG instances. They are:

(i) the generation of pairs of coalition structures; and

(ii) the simulation of a path in the tree.

For a discussion on the MCTS method, we refer to Section 2.4. The main problem in Step (i) was already introduced in Section 4.3, and here we shall focus on domain-independent approaches. Step (ii) refers to quickly reaching a terminal state in the tree, which is also not trivial for problems containing constraints. In the sections below, we propose the UCT-Seq algorithm and discuss how we address in it the two steps mentioned above.

5.2.1 A General MCTS Approach to SCFG

In this section, we introduce a general MCTS algorithm based on UTC (Kocsis and Szepesvári, 2006) to solve the problem of finding a sequence of coalition structures (i.e., the underlying SCFG problem). In particular, we consider here the SEQSVS framework as it fits well the application we shall introduce in Chapter 6. In our formulation, a single node is not seen as a solution to the SCFG problem. Instead, each node represents a single CS. Thus, we are interested in finding a path in the tree that reaches level h. This path corresponds to an FCSS.

Let $\mathit{Tree}(\mathsf{root})$ be a tree with root root iteratively built using UCT-Seq. The root root corresponds to $\emptyset$. We do so to point out which coalition structures may start a sequence. This procedure is the same as the one used in Chapter 4 to design the exact algorithms. Given any node x of $\mathit{Tree}(\mathsf{root})$, we use $\mathit{Tree}(\mathsf{x})$ to denote the subtree rooted at x (i.e., the tree induced by the descendants of x). Let $\mathit{path} = \langle \mathsf{root}, \mathsf{y}_1, \ldots, \mathsf{y}_k \rangle$ be a unique sequence of nodes in the tree. We use $\mathit{length}(\mathit{path})$ to denote the length of a path. Note that the maximum height of the tree $\mathit{Tree}(\mathsf{root})$ is equal to the length of the sequence of games; that is, h. Once UCT-Seq finds a path path such that $\mathit{length}(\mathit{path}) = h + 1$, it will have found a corresponding FCSS. Each node in the tree is defined to have the data structure below.

Definition 22 (Node). A node x is a tuple

$$\mathsf{x} = \langle \mathit{parent}, \mathit{CS}, l, \mathit{children}, N, \mathit{val}, \mathit{expanded}, \mathit{terminal}, i \rangle,$$

where:

- $\mathsf{x}.\mathit{parent}$ is a pointer to the parent node;
- $\mathsf{x}.\mathit{CS}$ is a coalition structure from $\boldsymbol{\mathcal{CS}}^A$;
- $\mathsf{x}.l$ is the level where x is placed in the tree, that is, $\mathit{length}(\langle \mathsf{root}, \ldots, \mathsf{x} \rangle) - 1$;
- $\mathsf{x}.\mathit{children}$ is a finite list of child nodes;
- $\mathsf{x}.N \in \mathbb{Z}^+$ is a counter of visits to the node;
- $\mathsf{x}.\mathit{val} \in \mathbb{R}$ is the cumulative reward of the node;
- $\mathsf{x}.\mathit{expanded}$ is a Boolean variable stating whether the node has been fully expanded;
- $\mathsf{x}.\mathit{terminal}$ is a Boolean variable stating whether the node is terminal; and
- $\mathsf{x}.i$ is an index with the specific purpose explained below.

Alternatively, we use y and z to refer to nodes as well.

Algorithm 5.5 A Monte Carlo tree-search approach to the SCFG problem.

Input:
A, a set of agents
$\mathcal{H}$, a totally ordered sequence of CFGs
Π, a totally ordered sequence of SVSs
$\mathcal{R}$, a binary relation on $\boldsymbol{\mathcal{CS}}^A$
t, a time budget to run the algorithm
Output:
$\boldsymbol{CS}$, an FCSS

procedure MCTS($A, \mathcal{H}, \mathcal{R}, t$)
  $\boldsymbol{CS} \leftarrow \emptyset$
  root $\leftarrow$ CreateNode(NIL, $\emptyset$) ▶ root.$parent$, root.CS
  while not ($timeout(t)$ **or** root.$terminal$) **do**
    x $\leftarrow$ TreePolicy(root)
    $\Delta \leftarrow$ DefaultPolicy(x)
    Update(x, Δ)
    if x.$terminal$ **then**
      if x.$l = h$ **then**
        $\boldsymbol{CS}' \leftarrow$ RetrieveFCSS(x)
        if $\mathcal{V}(\boldsymbol{CS}') > \mathcal{V}(\boldsymbol{CS})$ **then**
          $\boldsymbol{CS} \leftarrow \boldsymbol{CS}'$
      Remove(x)
  return $\boldsymbol{CS}$

A node is only distinguishable given the attributes x.$parent$, x.CS, and x.l. The Boolean variable x.$expanded$ is true when the whole set $X^{\text{x}.CS}_{\text{x}.l}$ (Definition 19) has been added to the x.$children$. The Boolean variable x.$terminal$ is set to true either if: (i) x.$l = h$; or (ii) x.$expanded \wedge |\text{x}.children| = 0$. The last attribute x.$i$ is used to retrieve a CS from a list that corresponds to $X^{\text{x}.CS}$ (Definition 18). That is, during the search we construct the set X^{CS}; it is based only on $\mathcal{R}$. Essentially, each node at a different level filters out incompatible CSs given the VS constraints of the subsequent level. Note this list is shared among all nodes, therefore, each node keeps an index pointing out to the next CS to evaluate.

At the beginning of the execution, the tree contains only node root where root.$CS = \emptyset$. This is the same mechanism introduced for the exact algorithms in Section 4.1.1. To select a node, we apply the tree policy, which returns a node. We consider in this procedure both ***selection*** and ***expansion*** steps. First, let us discuss the expansion step. Consider the root node. We need to select an action (i.e., a transition from one CS to another) which can be applied at that particular state. We can construct the set of actions that are compatible with it using the set $X^{\emptyset}$ (see Definition 18 and 19). To do so, we assume a generative algorithm $gen(CS)$ to generate compatible CSs. In case it is efficient, then $\{CS' \in gen(\text{x}.CS)\} = X^{\text{x}.CS}$. However, we investigate procedures that can deal with any $\mathcal{R}$ by generating potentially many more CSs than needed, and therefore $\{CS' \in gen(\text{x}.CS)\} \supseteq X^{\text{x}.CS}$. Such a procedure, for instance, can generate the complete set of coalition structures $\boldsymbol{\mathcal{CS}}^A$. We discuss the generation of coalition

Algorithm 5.6 Selection of a node to expand.

```
1: procedure TreePolicy(x)
2:     if not x.terminal then
3:         if not x.expanded and ⌊x.N^α⌋ ≥ |x.children| then
4:             ProgressiveWidening(x)                ▶ tries to expand x.children
5:         if |x.children| > 0 then
6:             y ← arg max_{z∈x.children} UCB1(z)
7:             return TreePolicy(y)
8:     return x
```

structures in Section 5.2.2. Thus, we construct $X^{\mathsf{x}.CS}$ on the fly and use a list $Act[\mathsf{x}.CS]$ to store the compatible CSs.

Despite the choice of a generative $\mathcal{R}$, the set $X^{\mathsf{x}.CS}$ might still be very large. In fact, depending on $\mathcal{R}$, a CS CS can relate to every other CS $CS' \in \boldsymbol{\mathcal{CS}}^A$. Therefore, we apply a progressive widening mechanism, in particular, we follow (Lee et al., 2020) and use the heuristic in Equation 5.2 in order to expand the list of child nodes of a non-terminal node x:

$$\lfloor \mathsf{x}.N^{\alpha} \rfloor \geq |\mathsf{x}.children| \tag{5.2}$$

where $\alpha \in [0, 1]$ is the expansion factor. If the above condition holds, then we retrieve a CS from $Act[\mathsf{x}.CS][\mathsf{x}.i]$. In case a node x has added all CSs from that list to $\mathsf{x}.children$, then we call the generative algorithm to produce a new CS.

To select a node from the current node x, we execute Equation 5.3 on its list of child nodes.

$$\mathsf{y} = \arg \max_{\mathsf{z} \in \mathsf{x}.children} UCB1(\mathsf{z}) \tag{5.3}$$

We use heuristic UCB1 (Auer et al., 2002) to establish a priority among the current child nodes. This heuristic is given in Equation 5.4.

$$UCB1(\mathsf{z}) = \frac{\mathsf{z}.val}{\mathsf{z}.N} + c\sqrt{\frac{\log \mathsf{z}.parent.N}{\mathsf{z}.N}} \tag{5.4}$$

It states that less frequently visited child nodes progressively increase their priority. This is given by the second component of the equation which is influenced by a ***exploration factor*** c which is chosen prior to the execution of the algorithm. In our case, we use the standard value $c = \sqrt{2}$ (Dann et al., 2020). The first component of Equation 5.4 is the exploitation term. In case a child node y has not been visited yet, that is, $\mathsf{y}.N = 0$, then we use $V_{\mathsf{y}.l}(\mathsf{y}.CS) + c\sqrt{\log \mathsf{z}.parent.N}$. Doing so, the exploration term eventually makes y be chosen even if the exploitation term is low. In case Equation 5.3 does not have a single solution, we randomly select a node among the options. We introduce the tree policy in Algorithm 5.6.

Once a node has been selected to be simulated, we estimate how good its subtree can be by carrying out the ***default policy***. This is another challenging part of the problem as we

Algorithm 5.7 Simulation on a given node.

Input:
$\bar{b}$, the maximum number of branching attempts
$\bar{d}$, the deepest relative level to achieve

```
1: procedure DefaultPolicy(x)
2:     if x.l = h then
3:         return V_l(x.CS)
4:     if x.N > 0 then
5:         return 0
6:     return Rollout(x.CS, x.l, min(x.l + d̄, h))
7: procedure Rollout(CS, l, depth)
8:     if l ≥ depth then
9:         return V_l(CS)
10:    Δ ← −∞
11:    CS' ← Split(CS, l + 1)
12:    if (CS, CS') ∈ R then
13:        Act[CS].add(CS')
14:        if CS' ∈ 𝒞𝒮^{π_{l+1}} then
15:            Δ ← Rollout(CS', l + 1, depth)
16:    for i ← 0, b̄ do
17:        CS' ← Merge(CS, CS', l + 1)          ▶ CS' ∈ 𝒞𝒮^{π_{l+1}} or CS' ← NIL
18:        if CS' = NIL then
19:            break
20:        Δ' ← Rollout(CS', l + 1, depth)
21:        Δ ← max(Δ, Δ')
22:    if Δ = −∞ then
23:        Δ ← 0
24:    return V_l(CS) + Δ
```

do not know which actions can be carried out at the selected node. We perform a depth-first search to try to reach an end of a sequence (that is, a coalition structure at level h). Our approach consists in generating a baseline coalition structure by performing *split* operations on the simulated CS. A split operation divides a coalition into two or more subsets. From this baseline CS, we start carrying out *merger* operations. A single merger operation merges two coalitions. We discuss how we apply those operations in Section 5.2.3. However, the above heuristic is neither guaranteed to find a subsequent CS nor to reach the end of a sequence. Therefore, we assume as input:

- a maximum number of attempts $\bar{b}$ to generate CSs at each level; and
- a maximum relative depth $\bar{d}$ to simulate in the subtree.

We introduce the default policy in Algorithm 5.7.

The last step is to update the tree statistics according to the new simulated node (and its subtree). Usually, $\mathsf{x}.val$ is a cumulative value of all rewards collected up to that point averaged by the number of visits on the given node (Browne et al., 2012). We follow the same approach

Algorithm 5.8 Statistics update regarding a selected path.

```
1: procedure UPDATE(x, Δ)
2:     while x.parent ≠ NIL do
3:         x.N ← x.N + 1
4:         x.val ← x.val + Δ
5:         x ← x.parent
```

updating the path with the reward generated by a roll-out. In case a selected node has already been simulated (e.g., no action available in the selected path), we return a reward of 0. Doing so, nodes in which it is difficult to find actions (i.e., compatible coalition structures) gradually decrease the priority in Equation 5.4. We are left to update properly the visited nodes' counters. We introduce those steps in Algorithm 5.8.

In case the selected node is a terminal node at level h, we are in a position to evaluate an FCSS. We backtrack the path from the terminal node up to the root, and return the FCSS $\boldsymbol{CS}'$. Recall that each path $path$ in the tree is in one-to-one correspondence to a CSS (a path may not reach level h). At this point, if it leads to the best solution found so far, then we store it, otherwise we continue the search. As a terminal node will not provide any further information for the search, we remove it from the tree so as not to bias the search towards it. This is addressed differently in some works in the MCTS literature in which an MCTS solver (Winands et al., 2008) back-propagates the game-theoretical values of ∞ or $-\infty$. No further search is conducted on nodes having value of $-\infty$.

Given the overall MCTS method employed by UCT-Seq, one can show an interesting property as follows.

Theorem 5. UCT-Seq is complete whenever the generative algorithm generates all compatible coalition structures given a preceding one.

Proof. We just need to show that for every node in the tree all possible child nodes are added to it. In the worst-case scenario, only the generator will find CSs to expand the UCT-Seq tree. We note that, given a preceding CS CS, for all levels $1 \leq l \leq h$, it holds that $X_l^{CS} \subseteq X^{CS}$ (if the problem of interest does not require constraints per level, then one should consider $X_l^{CS} = X^{CS}$). If a generator gen computes all compatible CSs, then we have at least $\{CS' \in gen(CS)\} \supseteq X_l^{CS}$ for any given level l. As UCT-Seq starts off from the node root corresponding to $\emptyset$, every CS in $X_1^{\emptyset}$ will correspond to a child node of root. The search proceeds until UCT-Seq reaches level $h-1$, where given a node x at that level, every CS in $X_h^{\mathsf{x}.CS}$ will be added to the tree. Once a terminal node is evaluated, it is removed from the tree. Recall that a node y is terminal if (i) it is placed at the tree level h; or (ii) it has added all $X_{\mathsf{y}.l+1}^{\mathsf{y}.CS}$ to its list of child nodes and all of those nodes have already been removed from the tree. Therefore, UCT-Seq eventually builds the complete search space of an SCFG-based instance and finds a solution for the problem. □

One can easily note as well that UCT-Seq not only computes a solution for the problem (provided the assumption above), but it also finds eventually an optimal solution. However, for

most real-world applications, one cannot afford the time required by UCT-Seq to search the entire search space. Such applications usually require large instances to be solved. Therefore, we think of UCT-Seq from a heuristic perspective, where we only run it for as long as we can reasonably wait for a result.

5.2.2 The Generation of Coalition Structures

An important part of the MCTS method is to select an action to apply on a state and hence expand the tree. If the set of actions is small, one can select an action based on some distribution. For instance, in Go, one has approximately 250 actions available at each state (Silver et al., 2016). Once a node is expanded, for each action a prior probability of selecting it is calculated. In a continuous-action space, for instance in (Lee et al., 2020), an action is sampled from a computed policy network (the same policy is used during a roll-out). Although our set of actions is finite, it is intractably large. In fact, it is of size $\Omega(n^{\frac{n}{2}})$, where n is the number of agents. To address this problem, we propose an algorithm inspired by CFSS (Bistaffa et al., 2014), which is graph-based and anytime, to generate the CSs that may follow a given CS. CFSS uses 2-coloured graphs to keep track of the coalition structures that have already been evaluated. For more details on this algorithm we refer to Section 2.3.5.

Definition 23 (2-coloured Graph G^c induced by Γ^π). Given a CFG $\Gamma = \langle A, v\rangle$ induced by an SVS $\pi = \langle G, S, Z\rangle$, where $G = \langle A, E'\rangle$, a 2-coloured graph G^{Γ^π} is a tuple $\langle V, E, c, w\rangle$ where:

- $V = \{\{a\} \mid a \in A\}$, that is, each vertex represents a coalition $C \subseteq A$;
- $E = \{(\{u\}, \{r\}) \mid (u, r) \in E'\}$;
- c is a function $E \to \{green, red\}$; and
- w is a function $E \to \mathbb{R}$.

In particular, we let $w((u, r)) = gain(u, r)$ (see Equation 2.2).

Moreover, regarding the constraints imposed by an SVS π. Let $\bar{\pi}$ be a relaxed version of the constraints in $\pi = \langle G, S, Z\rangle$ such that $\bar{\pi} = \langle G, S, \{1, 2, \ldots, \max(Z)\}\rangle$. Also, let $\boldsymbol{\mathcal{C}}^{\bar{\pi}}$ be the set of coalitions that are feasible regarding $\bar{\pi}$. That is, for all $C \in \boldsymbol{\mathcal{C}}^{\bar{\pi}}$: (i) $|C \cap S| \leq 1$; (ii) the sub-graph of C induced over G is connected; and (iii) $|C| \leq \max(Z)$. Note that $\boldsymbol{\mathcal{C}}^{\bar{\pi}} \supseteq \boldsymbol{\mathcal{C}}^{\pi}$. To colour an edge, we first check the VS constraints. Given an edge $e = (u, r)$, if $u \cup r \notin \boldsymbol{\mathcal{C}}^{\bar{\pi}}$, then $c : e \mapsto red$. That is, the resulting coalition has more than one pivotal agent or its size is bigger than the maximum size allowed. Note that the sub-graph of $u \cup r$ induced over G is connected given Definition 23, as the coloured graph is build on the top of the interaction graph. The remaining edges are initially coloured green. We shall use $e \in G^{\Gamma^\pi} : e \in E$ when it is clear from the context.

We follow a similar procedure introduced by Bistaffa et al. (2014, Definition 3) to contract a green edge. Their definition is adapted to our context and introduced below.

Definition 24 (Green Edge Contraction (adapted from (Bistaffa et al., 2014))). Given a 2-coloured graph $G^{\Gamma^\pi} = \langle V, E, c, w \rangle$ and a green edge $e = (u, r)$, where $e \in E$, the result of the contraction of edge e is a graph $G^{\Gamma^{\pi\prime}}$ obtained by performing the contraction of the edge e in graph G^{Γ^π}. Whenever two parallel edges are merged into a single one, the resulting edge is coloured red either if at least one of them is red-coloured or the resulting coalition $u \cup r \notin \boldsymbol{\mathcal{C}}^{\bar{\pi}}$. It is coloured green otherwise.

Before introducing our algorithm to generate CSs, let us introduce some additional notation. Let CS^e be the coalition structure represented by graph $G^{\Gamma^{\pi\prime}}$ after the contraction of edge e in G^{Γ^π}. Moreover, given the preceding CS CS in the sequence, we construct $\bar{E} = \{e \in E \mid c(e) = green, CS^e \in \boldsymbol{\mathcal{CS}}^\pi, (CS, CS^e) \in \mathcal{R}\}$, the set of all edges that result in a CS compatible with CS. Algorithm 5.9 introduces the steps to generate a CS. Note we need to generate a CS (i.e., an action in the context of MCTS) and suspend the execution until a new CS is requested. We use the keyword **yield** to denote that the algorithm returns a CS and is suspended. We use the variable $counter$ to enforce a number of attempts to return a CS. Doing so, UCT-Seq will then decide whether it continues searching for a feasible CS in the generator or explores a different path in the tree.

As an example, consider Figure 5.3. We aim to generate the set of CSs compatible with $CS = \{\{a_1, a_2, a_3\}, \{a_4\}\}$. We draw a green-coloured edge in Figure 5.3 using a dashed line when the edge is not in $\bar{E}$. The feasibility of pairs in $\mathcal{R}$ is given by the same number of coalitions in the coalition structures; that is, $\mathcal{R} = \mathcal{R}_2$. The set X_l^{CS} contains only a single CS, namely $CS' = \{\{a_1, a_3\}, \{a_2, a_4\}\}$. However, 7 additional coalition structures are checked by the generator. Edges not in $\bar{E}$ represent that a resulting CS (i.e., the CS resulting from an edge contraction) is not feasible yet. For instance, consider a G^{Γ^π} where every vertex is a singleton coalition. Let $Z = \{1, 3\}$, then no contraction of any edge results in a coalition that follows the size constraint. In fact, at least two edge contractions must be carried out to reach a coalition of size 3.

We show that the generator explained above cope with the requirement in Theorem 5. That is, the procedure generates all coalition structures that may follow a given one in an FCSS.

Theorem 6. Given a preceding coalition structure CS and a level l, for all coalition structures $CS' \in X_l^{CS}$ there exists a corresponding node in the tree rooted at node $G^{\Gamma^{\pi_l}}$.

Proof. Assume by contradiction that there exists a coalition structure $CS' \in X_l^{CS}$ that does not have a corresponding node in the tree rooted at node $G^{\Gamma^{\pi_l}}$. Recall that a CS $CS' \in X_l^{CS}$ iff $CS \; \mathcal{R} \; CS'$ and $CS' \in \boldsymbol{\mathcal{CS}}^{\pi_l}$. Therefore, it must be the case that for all coalitions $C' \in CS'$, their induced sub-graphs over the interaction graph G_l are connected. The trivial case is when CS' corresponds to $G^{\Gamma^{\pi_l}}$. If not so, the corresponding edges in $G^{\Gamma^{\pi_l}}$ of the sub-graphs induced by the coalitions $C' \in CS'$ over G_l must be green coloured. Otherwise, CS' would not follow

Algorithm 5.9 A coalition structure generator for SEQSVS.

Input:
CS, a coalition structure representing the first element in a pair in $\mathcal{R}$
t, number of attempts to generate a CS CS'

```
1: counter ← 0
2: G^Γπ ← INIT(G)                                   ▶ interaction graph G
3: return GENCS(G^Γπ, counter)
4: procedure GENCS(G^Γπ, counter)
5:     while Ē ≠ ∅ do
6:         counter ← 0
7:         e ← arg max_{e'∈Ē} w(e')
8:         G^Γπ' ← GREENEDGECONTRACTION(G^Γπ, e)          ▶ Definition 24
9:         Mark e with colour red in G^Γπ
10:        yield CS^e        ▶ return a CS compatible with CS and suspend the execution
11:        GENCS(G^Γπ', counter)
12:    while there exists green edges in G^Γπ do
13:        counter ← counter + 1
14:        if counter > t then
15:            counter ← 0
16:            yield NIL          ▶ no solution found so far; suspend the execution
17:        e ← arg max_{e'∈G^Γπ} w(e') : c(e') = green
18:        G^Γπ' ← GREENEDGECONTRACTION(G^Γπ, e)
19:        Mark e with colour red in G^Γπ
20:        GENCS(G^Γπ', counter)
21:    return
```

the pivotal or maximum size constraints. We know that every node in the tree will have as many child nodes as it has green edges. Any green edge in a node is only coloured red iff: (i) it has been tried before; or (ii) the resulting coalition is not feasible regarding the pivotal and maximum size constraints. All green edges in the tree are eventually contracted. Therefore, CS' has a corresponding node in the tree. □

Given this result, the one below immediately follows.

Corollary 1. UCT-Seq is complete given the generator above.

Proof. The proof follows from Theorems 5 and 6. The generator yields all corresponding CSs in its tree that are compatible with a preceding one CS and follow the SVS constraints of the current level l. As $X_l^{CS} \subseteq X^{CS}$ and for a SEQSVS problem the coalition structures of interest are in X_l^{CS}, UCT-Seq eventually finds a solution for the problem. □

To speed up the search, we keep in memory a list of CSs compatible to x.CS (i.e., the CS represented by node x) regardless of the level in which x is in the tree. Let $Act[CS]$ be such list. Once the $gen(CS)$ is run to completion, then $Act[CS] \supseteq X_l^{CS}$. We note that more than one node may represent the same CS but at different parts of the tree. Moreover, one should note that UCT-Seq and MC-Link share the same greedy principle while selecting coalitions to

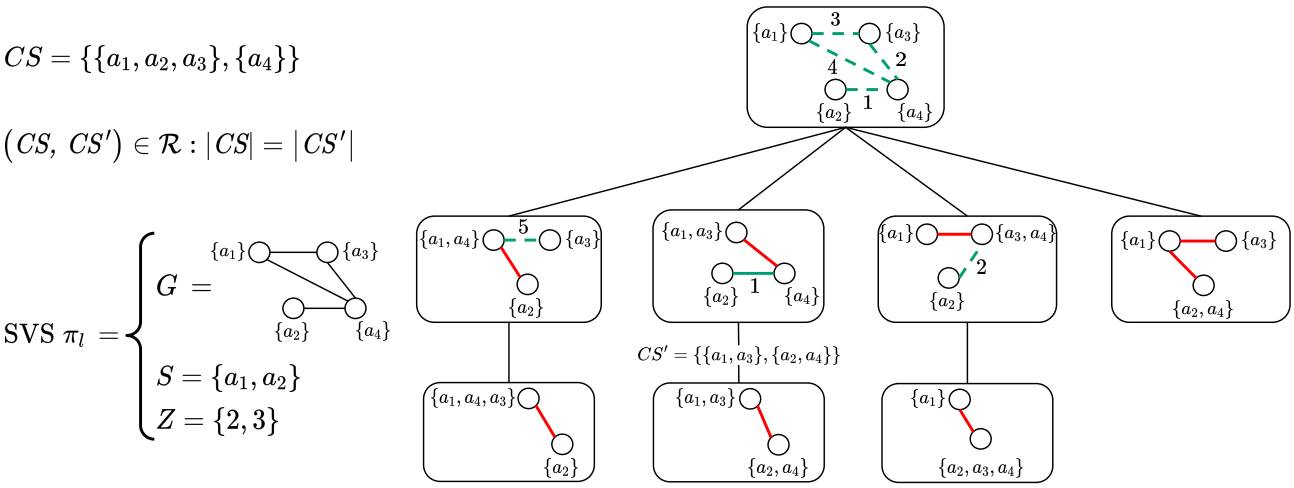


Figure 5.3: A running example to generate the set X_l^{CS} where $CS = \{\{a_1, a_2, a_3\}, \{a_4\}\}$.

merge. The main difference between the generation procedure described above and MC-Link is that UCT-Seq eventually tries different CSs and thus, it can potentially avoid local optimum.

5.2.3 Simulation of a Sequence of Coalition Structures

The generator discussed above addresses a single level of the sequence. In a simulation, one is interested in quickly reaching a terminal state, which, for an SCFG problem, is a CS at level h. This step is referred as either *roll-out* or *play-out* and a standard approach for this is to select actions randomly at each level of the simulated tree (Browne et al., 2012). However, this is not always the case. For instance, Wu and Ramchurn (2020), in the context of the coalition structure generation problem, use an informed search to reach a terminal state, which ends up to be the grand coalition. In our case, we do not know beforehand which actions are applicable at the node of interest. By employing an informed search (e.g., using the characteristic functions to guide the search), it is not guaranteed that the procedure will reach a terminal state. In fact, the values assigned to coalitions might not be related to the constraints posed by both $\mathcal{R}$ and the VSs. That means, well-evaluated coalitions might lead to dead ends in a sequence of CSs.

In a simulation, we conduct a series of split and merger operations on the CS CS of the newly expanded node. Other operators have been used in the CSG literature. For instance, Mauro et al. (2010) used split, merge, shift, exchange, and extract operations. However, they noted that one needs to be careful about applying different operators on coalitions in order to avoid repeating CSs.

The split operations are first based on: (i) the VS constraints of the subsequent level $l+1$; then on (ii) random operations. Given a coalition C, if $|C \cap S_{l+1}| > 1$, then we split the pivotal agents into singleton coalitions. Let $\widehat{CS}$ be a set of coalitions containing initially only singletons of pivotal agents. For each remaining agent $a \in C \setminus S_{l+1}$, we decide whether to include it in a coalition $C \in \widehat{CS}$ (if $C \cup \{a\} \in \boldsymbol{C}^{\bar{\pi}_{l+1}}$) or keep it in a singleton $\{a\} \in \widehat{CS}$ with a

Algorithm 5.10 A procedure to split coalitions in a coalition structure.

procedure SPLIT(CS, l)
  $CS' \leftarrow \emptyset$
  for all $C \in CS$ **do**
    if $|C \cap S_l| > 1$ **or** $|C| > \max(Z_l)$ **then**
      $CS' \leftarrow CS' \cup$ BREAKUPCONSTRAINTS(C, l)
    else
      for all $C' \in$ COMPONENT(C, G_l) **do**
        $CS' \leftarrow CS' \cup$ BREAKUPCONTRIBUTION(C')
  return CS'
procedure BREAKUPCONSTRAINTS(C, l)
  $CS' \leftarrow \emptyset$
  for all $a \in C \cap S_l$ **do**
    $CS' \leftarrow CS' \cup \{\{a\}\}$
  for all $a \in C \setminus S_l$ **do**
    $CS' \leftarrow CS' \cup \{\{a\}\}$
    for all $C' \in CS' : C' \neq \{a\}$ **do**
      if $C' \cup \{a\} \in \boldsymbol{\mathcal{C}}^{\bar{\pi}_l}$ **and** $U(0,1) \leq 0.5$ **then**
        $CS' \leftarrow (CS' \cup \{C' \cup \{a\}\}) \setminus \{C'\} \setminus \{\{a\}\}$
        break
  return CS'
procedure BREAKUPCONTRIBUTION(C)
  $CS' \leftarrow \emptyset$
  $C' \leftarrow C$
  for all $a \in C$ **do**
    if $U(0,1) \leq 0.5$ **then**
      $C' \leftarrow C' \setminus \{a\}$
      $CS' \leftarrow CS' \cup \{\{a\}\}$
  if $C' \neq \emptyset$ **then**
    $CS' \leftarrow CS' \cup \{C'\}$
  return CS'

probability of 50%. The size procedure occurs if $|C| > \max(Z_{l+1})$. In case both constraints are satisfied, then the disconnected components of C induced over G_{l+1} become the new coalitions and we let any agent $a \in C$ form a singleton coalition with a probability of 50%. The overall procedure to split a coalition structure for the subsequent level l is introduced in Algorithm 5.10.

The procedure described above produces a single coalition structure. Our next step is to merge coalitions. The procedure is introduced in Algorithm 5.11. We merge any two coalitions $C, C' \in CS'$ iff $C \cup C' \in \boldsymbol{\mathcal{CS}}^{\bar{\pi}_{l+1}}$ with a probability of 50%. In case a merger is feasible, i.e., allowed by $\mathcal{R}$ and the SVS, $(CS_l, CS') \in \mathcal{R}$ and $CS' \in \boldsymbol{\mathcal{CS}}^{\pi_{l+1}}$, then we return CS' and repeat the procedure until we reach the maximum depth in a simulation; otherwise we try other mergers until no further mergers are possible.

To potentially contribute to the expansion of the tree (i.e., selecting new coalition structures as child nodes), once a CS compatible with CS CS according to $\mathcal{R}$ is found, we add

Algorithm 5.11 A procedure to merge coalitions in a coalition structure.

1: **procedure** MERGE(CS, CS', l)
2: $\quad \widehat{C}, \widehat{C'} \leftarrow$ randomly select $C, C' \in CS'$ s.t. $C \neq C', C \cup C' \in \boldsymbol{\mathcal{C}}^{\bar{\pi}_l}$
3: $\quad \widehat{CS} \leftarrow (CS' \setminus \{\widehat{C}\} \setminus \{\widehat{C'}\}) \cup \{\widehat{C} \cup \widehat{C'}\}$ ▶ return NIL if none is found
4: $\quad$ **if** $(CS, \widehat{CS}) \in \mathcal{R}$ **then**
5: $\quad\quad Act[CS].add(\widehat{CS})$ ▶ construction of set X^{CS}
6: $\quad\quad$ **if** $\widehat{CS} \in \boldsymbol{\mathcal{CS}}^{\pi_l}$ **then**
7: $\quad\quad\quad$ **return** $\widehat{CS}$
8: $\quad$ **return** MERGE(CS, $\widehat{CS}$, l)

it to the list $Act[CS]$. Doing so, two distinct local searches contribute to the construction of the set X^{CS}, namely the generator and the simulation.

5.2.4 Discussion

MC-Link and SDP differ fundamentally (besides the fact that SDP is exact) in how they address the SCFG problem. MC-Link relies only on the valuation functions to apply a greedy strategy and come up with a good FCSS (in terms of value). On the other hand, the exact algorithms introduced in Chapter 4 rely on a procedure to generate pairs of coalition structures. UCT-Seq combines both worlds by employing the MCTS method. In a simulation, it performs random moves to modify a given CS (following the SVS constraints) to try to reach a terminal state in the tree. It also relies on a generator to compute compatible CSs in case the simulation procedure fails to find them. The latter strategy employed by UCT-Seq provides an important property for this sort of algorithm: the completeness property (Corollary 1).

Many components of the UCT-Seq algorithm might influence the outcome produced after the time budget is finished. Of special concern is how to deal with constraints in this approach. In case both generation and simulation steps fail to produce child nodes (i.e., placing new CSs in the list Act) in a given number of iterations, the UCB1 heuristic can converge easily to the same value for all nodes and from that moment on, only exploration is carried out (i.e., breadth-first search). This might increase the running time when the only way out is provided by the generator. That is, one needs to run the generator a few times until it computes a CS of interest. As future work for those cases, one might investigate the possibility of running other instances of UCT-Seq for the same problem but with different parameters. Alternatively, UCT-Seq could adjust its MCTS parameters as soon as it detects a convergence in the UCB1 heuristic.

Regarding the simulation, it plays an important role in the MCTS method. It might be interesting to investigate and compare in future work other approaches:

Informed search: use the valuation functions to guide the search. This is the strategy adopted in CSG-UCT (Wu and Ramchurn, 2020), in which the authors, in a simulation step, merge

continuously the two coalitions that lead to the greatest gain until the grand coalition is formed. In this particular case, the simulation running time increases significantly $\mathcal{O}(m^3)$ where m is the size of the coalition structure. Another drawback is in cases the valuation function is not aligned with the constraints. For instance, the coalitions of greatest values are unfeasible.

Unfeasible coalitions: use the incompatible coalitions in a CS to guide the search. Recall that a relation $\mathcal{R}$ states which coalitions structures are compatible. This compatibility can be derived from the coalitions that belong to the CS of interest. In that case, instead of developing algorithms that check whether a pair of CSs are in $\mathcal{R}$, it could also return what coalitions do no follow the constraints required by $\mathcal{R}$. This new information can be used by general-purpose simulation algorithms to guide the search. In case the constraints follow from the CS itself (e.g., the CS requires an exact size s) the whole CS is returned.

Also, we aimed at providing general algorithms that could be applied to any domain of interest. However, if the time performance is a critical issue, one should consider both generation and simulation that are domain dependent.

One should also bear in mind that UCT-Seq cannot tell quickly whether an instance is *hard* to solve or *impossible* to solve, unless the infeasibility of the sequence of games is given at games in the initial positions. In case the infeasibility comes from games in the last positions of the sequence, UCT-Seq will continue expanding CSs in the upper levels hoping that a CS which will make the whole sequence feasible is still to be found.

5.3 Experiments

In this section we report on the experiments carried out to compare MC-Link with UCT-Seq. Those are the only available heuristic approaches to compute approximate solutions for SCFG-based problems. To assess the quality of their solutions, we use SDP as a reference. We discuss this procedure in detail in Section 5.3.2. Then, in Section 5.3.3, we increase the set of agents and compare the performance of UCT-Seq with MC-Link in more challenging instances.

5.3.1 Preliminaries

To evaluate the proposed algorithms, we pick different characteristics functions to understand how well they perform. The valuations follow from the CSG literature (Michalak et al., 2015). We remind the reader that a characteristic function v calculates a value for every coalition C that can be formed by the agents in A. We experiment with characteristic functions that draw a value from the distributions below.

Uniform for all $C \in 2^A$, $v(C) \sim U(0, |C|)$.

Modified Uniform a first value for a coalition follows $U(0, 10 \times |C|)$, then it is increased by a random number $r \sim U(0, 50)$ with probability of 20%.

Agent-based Uniform each agent $a_i \in A$ is assigned a random number $p_i \sim U(0, 10)$. Then, for each coalition $C : a_i \in C$, agent a_i contributes to C a value p_i^C generated based on $p_i^C \sim U(0, 2p_i)$. Hence, $v(C) = \sum_{a_i \in C} p_i^C$.

Normal for all $C \in 2^A$, $v(C) \sim N(10 \times |C|, 0.01)$.

Modified Normal a coalition receives a value drawn from $N(10 \times |C|, 0.01)$, then it is increased by a random number $r \sim U(0, 50)$ with probability of 20%.

Agent-based Normal similar to Agent-based Uniform. For each $a_i \in A$, $p_i \sim N(10, 0.01)$ and for each coalition $C : a_i \in C$ we have $p_i^C \sim N(p_i, 0.01)$. Hence, $v(C) = \sum_{a_i \in C} p_i^C$.

NDCS for all $C \in 2^A$, $v(C) \sim N(|C|, |C|)$.

Exponential for all $C \in 2^A$, $v(C) \sim |C| \times Exp(1)$.

Beta for all $C \in 2^A$, $v(C) \sim |C| \times Beta(0.5, 0.5)$.

Gamma for all $C \in 2^A$, $v(C) \sim |C| \times Gamma(2, 2)$.

We compare the algorithms in terms of quality of the outcome, running time and memory consumption.

Quality of the outcome: this metric refers to the value computed by $\mathcal{V}(\boldsymbol{CS})$. In particular, we are interested in the distance between two computed solutions. In small instances, when it is possible to compare solutions with the optimal value, we shall use the averaged optimal ratio (Definition 25 below). In instances in which an optimal outcome is not feasible, we use the quality improvement ratio (Definition 26 below). We set MC-Link as the baseline and evaluate whether the solution computed by UCT-Seq is of a better quality ratio (> 0) or poorer quality ration (< 0).

Running time: refers to the required time to compute a solution. For SDP and MC-Link measuring the running time is straightforward: we run the algorithm and record the precise time it took to output a solution. For UCT-Seq, we slightly modify this procedure. Recall that UCT-Seq requires us to set a time budget to compute a solution. Hence, this metric would always result in the same running time[2]. Instead, we record for UCT-Seq the precise moment that the best solution so far is found and use it to compare with the solutions computed by the other algorithms. That means, as an example, that the best solution found by UCT-Seq after the time budget of 60 seconds might have been found after only 20 seconds.

[2]In case the instance of interest is small enough, then it would indeed finish before the time budget is finished.

Memory consumption: refers to how much memory an algorithm requires to compute a solution for the problem. We compare the peak of memory in that regard. To keep track of this metric, we use the Python package `malloc-tracer` (version 1.7.0)[3].

Definition 25 (Averaged Optimal Ratio (AOR)). Given an optimal FCSS $\boldsymbol{CS}^*$ computed by SDP and an FCSS $\boldsymbol{CS}$ computed by either MC-Link or UCT-Seq, an averaged optimal ratio metric is calculated as follows:

$$\sqrt{\frac{\mathcal{V}(\boldsymbol{CS})^2}{\mathcal{V}(\boldsymbol{CS}^*)^2}}.$$

Definition 26 (Quality Improvement Ratio (QIR)). Given an FCSS $\boldsymbol{CS}$ computed by MC-Link and an FCSS $\boldsymbol{CS}'$ computed by UCT-Seq, the quality improvement ratio is computed as follows:

$$\sqrt{\frac{\mathcal{V}(\boldsymbol{CS}')^2}{\mathcal{V}(\boldsymbol{CS})^2}} - 1.$$

Regarding the algorithms, we consider only SEQVS problems. This is due to the fact that size constraints are particularly difficult for MC-Link as the transition from one size to the next one must be contained in that set of constraints. For instance, in SEQSVS instances in which all coalitions must follow $|C| > 3$ MC-Link will not be able to compute a solution as it starts off from the CS of singletons and any initial merger will result in a coalition $|C| = 2$ which will force MC-Link to halt. To avoid that, we consider only corresponding SEQVS instances by setting $Z = \{1, \ldots, n\}$. Moreover, both MC-Link and UCT-Seq use an algorithm to check whether a pair of CSs is compatible. On the other hand, SDP uses a generative algorithm for each relation $\mathcal{R}$ of interest. Both MC-Link and UCT-Seq are implemented in Python 3.8.10.

5.3.2 Approximation of the Optimal Value

In this experiment, we evaluate how close the solutions computed by the heuristic approaches are to the optimal value. To do so, we run both MC-Link and UCT-Seq for small instances and compare the outcome of each run with the one outputted by SDP. SDP computes an optimal solution (Theorem 1) and is faster than a brute-force algorithm (see Section 4.2). This makes SDP the perfect baseline for the experiments that follow. Out of the four binary relations experimented with SDP, we chose to evaluate the worst-case relation and the one that enforces a hierarchy of agents. That is, we experiment with $\mathcal{R}_1$ and $\mathcal{R}_3$ (Definition 20). This implies that we can experiment with a set of agents containing $1 \ldots 7$ agents for $\mathcal{R}_1$, and $1 \ldots 10$ agents for $\mathcal{R}_3$. Moreover, for $\mathcal{R}_1$ a sequence containing 10 CFGs is feasible. On the other hand, for $\mathcal{R}_3$ that number depends on the size of the set of agents A, as we cannot have a hierarchy

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

that demands more agents than the quantity available in it. For instance, a set containing four agents can never form a five-level hierarchy.

Before conducting each experiment, we draw a value for each coalition $C \subseteq A$ from a distribution and store it in a table. For each game $\Gamma \in \mathcal{H}$, we always sample new values; this means, all valuations are different, but from the same distribution. Similarly regarding the constraints, we generate a new set of constraints for each experiment containing a different number of agents and games as well as a different sort of valuations. That means, a new set of constraints is generated even if we change only the valuation functions. Doing so, we evaluate the algorithms in many more different settings. 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 $q \sim U(0, \lceil n \times 0.2 \rceil)$. We use $n \times 0.2$ to avoid picking all agents from A as pivotal agents.

Regarding the MCTS parameters, we experiment with the ones that follow.

Simulation depth $\bar{d}$: it determines the maximum level in a simulated subtree and is bounded by h. For instance, given $h = 4$, $\bar{d} = 2$, and a node in the first level, a simulation tries to achieve coalition structures at the third position in a sequence. For nodes located at the second and third levels, it always tries to achieve a coalition structure at the fourth position in the sequence. We experiment with 2, 3 and 4, and 10.

Simulation degree $\bar{b}$: it determines the branching factor of the simulated subtree; that is, how many attempts to find compatible CSs the simulation makes. We experiment with the values of 3 and 7.

Exploration factor γ: it determines the minimum number of child nodes a given node should have based on its visiting counter. As an example, suppose that for a given node x, its visiting counter is $\mathsf{x}.N = 1000$ and $\gamma = 0.4$. If $|\mathsf{x}.children| < 16$, then UCT-Seq tries to expand node x's list of child nodes. In the same setting when $\gamma = 0.6$, $|\mathsf{x}.children| < 64$ should be true for the expansion procedure to take place. We experiment with the values of 0.4 and 0.6.

Moreover, we let UCT-Seq run for 60 seconds in each instance. We conduct the experiments described above in a machine with 32 GB of RAM and a CPU with four single cores of 3400 MHz each.

We report in Figure 5.4 the results when $h = 10$ and $\mathcal{R} = \mathcal{R}_1$. In general, UCT-Seq achieves results that are slightly inferior to the ones computed by MC-Link. Both algorithms alternate solutions of better quality when we experiment with coalition values drawn from different distributions. They also tend to a negative steepness as we increase the number of agents, which indicates they will a have poor performance in larger instances. Moreover, for small instances the UCT-Seq parameters seem to not interfere in the solution produced by the MCTS-based approach.

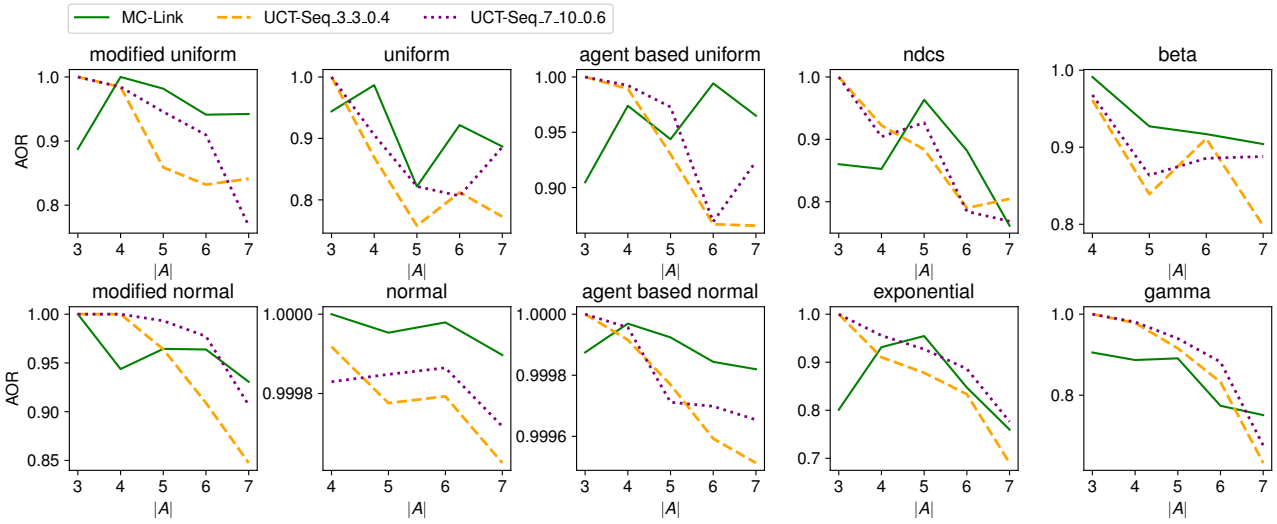


Figure 5.4: Comparison of MC-Link and UCT-Seq solution quality given an optimal solution when $h = 10$ and $\mathcal{R} = \mathcal{R}_1$. We depict the results of UCT-Seq for the settings $\bar{b} = 3$, $\bar{d} = 3$, $\gamma = 0.4$ (line UCT-Seq_3_3_0.4) and $\bar{b} = 7$, $\bar{d} = 10$, $\gamma = 0.6$ (line UCT-Seq_7_10_0.6).

Next, we turn our attention to the running time of each algorithm. To analyse the running time in UCT-Seq, we look at each instance individually and plot the amount of time required by MC-Link to output a solution (a solid circle in the charts). Then, we compare it with UCT-Seq varying the parameters listed above. In Figure 5.5, we depict the results when $h = 10$, $\mathcal{R} = \mathcal{R}_1$, and $n = 7$. We see that in general MC-Link is faster than UCT-Seq to compute a solution of better quality if compared with the ones produced by UCT-Seq after a similar amount of time. For instance, when the distribution is exponential, MC-Link achieves $\approx 75,98\%$ of the optimal value after $0,21$ second. On the other hand, line `UCT-Seq_3_3_0.6` after 0.23 second achieves a ratio of $\approx 60.81\%$. Similar running times for a similar solution quality occur only in instances in which the coalition values were drawn from the NDCS distribution. Recall that this distribution is known for producing an unbiased search space. For instance, MC-Link achieves $\approx 76,16\%$ of the optimal value after $0,19$ second. Nonetheless, line `UCT-Seq_3_10_0.6` after $\approx 0,11$ seconds had already achieved $\approx 78,84\%$. Regarding the different parameters of UCT-Seq, although there is no clear winner for all the settings, we see in some cases a difference in the running time even though the final solution quality is similar. For instance, for the modified uniform valuations, line `UCT-Seq_3_3_0.4` achieves its final quality value, $\approx 84,13\%$ of the optimal value, after $\approx 3,45$ seconds, whilst line `UCT-Seq_3_10_0.4` takes $\approx 27,97$ seconds to achieve $\approx 84,19\%$ (its final solution is computed after $\approx 55,89$ seconds with a ratio of $\approx 85,49\%$).

Next, we consider the narrow relation $\mathcal{R}_3$. It reduces significantly the search space, so it is interesting to evaluate the performance of both algorithms in such settings. We depict in Figure 5.6 the results when $h = 2$. For this number of games there should exist a solution for games containing $3 \ldots 10$ agents (considering the settings introduced above). One can clearly see that MC-Link has poorer performance than UCT-Seq. This is the case for all distributions. That means, the greedy approach of merging the two coalitions that provide the greatest gain

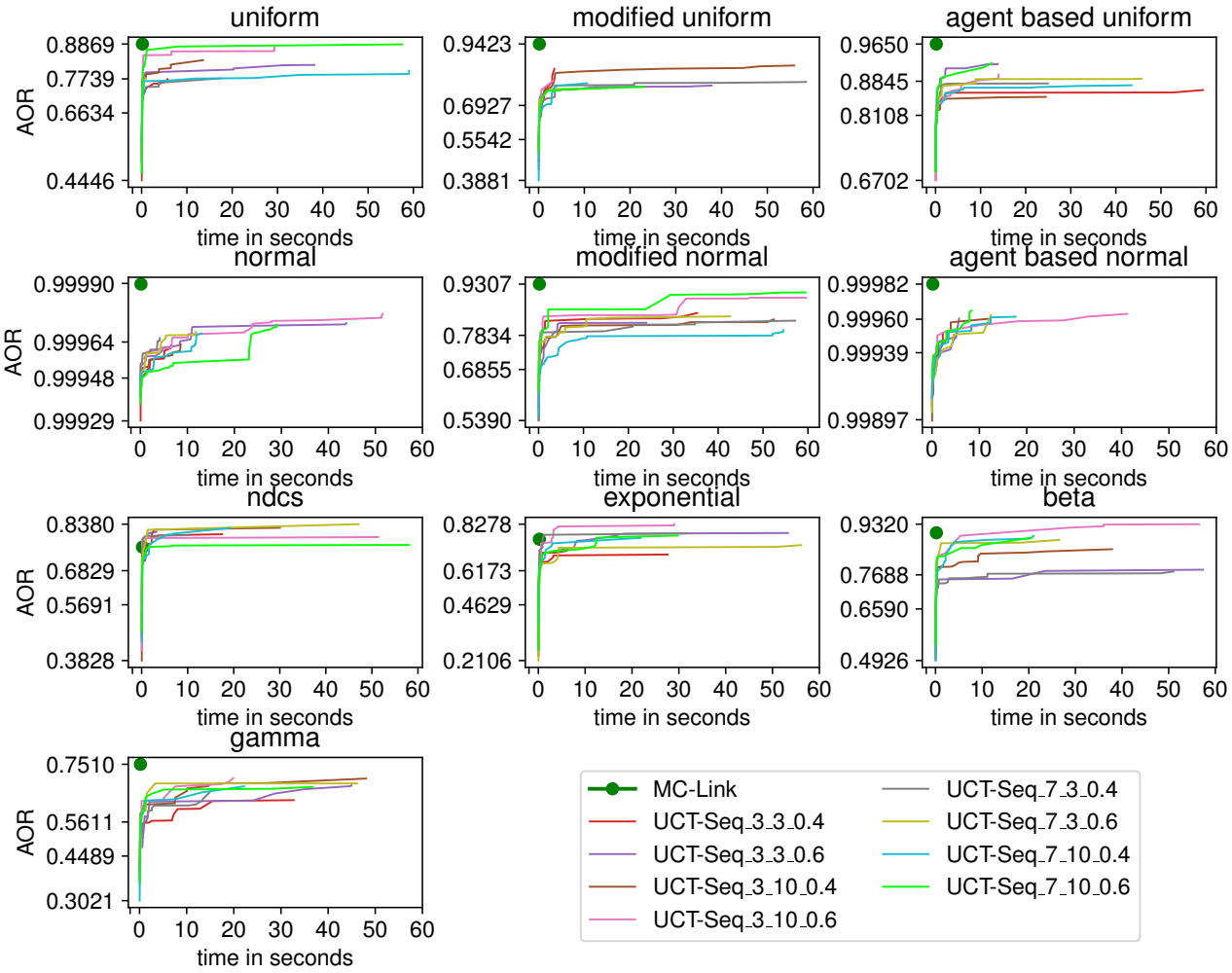


Figure 5.5: Comparison of MC-Link and UCT-Seq running time and solution quality given an optimal solution when $n = 7$, $h = 10$ and $\mathcal{R} = \mathcal{R}_1$. Each UCT-Seq line follows the same pattern UCT-SEQ_$\bar{b}$_$\bar{d}$_γ.

does not payoff in such a narrowed down search space (if no backtrack is allowed). This claim is supported by the experiments conducted for $h = 4$; that is, a sequence of four CFGs. In the results depicted in Figure 5.7, MC-Link falls behind UCT-Seq for all distributions as well. Moreover, in some cases (i.e., for the modified normal distribution when $n = 6$, and for the NDCS when $n = 5$) MC-Link cannot find a solution. This empirical observation is in accordance with the results in Section 5.1.2 where we show that MC-Link is not complete.

However, even though the performance of MC-Link decreases for more restrictive binary relations, one should recall that for $\mathcal{R}_1$ MC-Link is faster than UCT-Seq to compute a solution. It remains to be shown how fast it is for $\mathcal{R}_3$. Again, we evaluate the improvement made by UCT-Seq throughout a time budget of 60 seconds when $n = 10$ and $h = 4$. We report the results in Figure 5.8. One can see that the performance of UCT-Seq improved in comparison with the experiments conducted with $\mathcal{R}_1$. In fact, UCT-Seq, in general, outperforms MC-Link in the long term. Moreover, after similar running times, UCT-Seq computed solutions of similar quality or better than MC-Link. However, the quality in uniform-based valuations is greater than

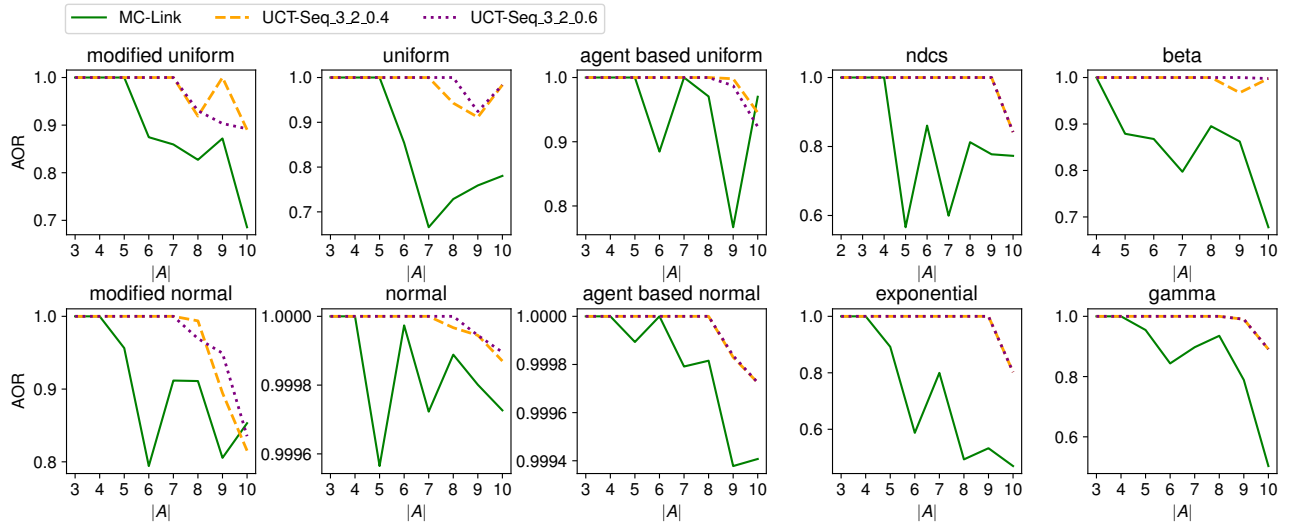


Figure 5.6: Comparison of MC-Link and UCT-Seq solution quality given an optimal solution when $h = 2$ and $\mathcal{R} = \mathcal{R}_3$. We depict the results of UCT-Seq for the settings $\bar{b} = 3$, $\bar{d} = 2$, $\gamma = 0.4$ (line UCT-SEQ_3_2_0.4) and $\bar{b} = 3$, $\bar{d} = 2$, $\gamma = 0.6$ (line UCT-SEQ_3_2_0.6).

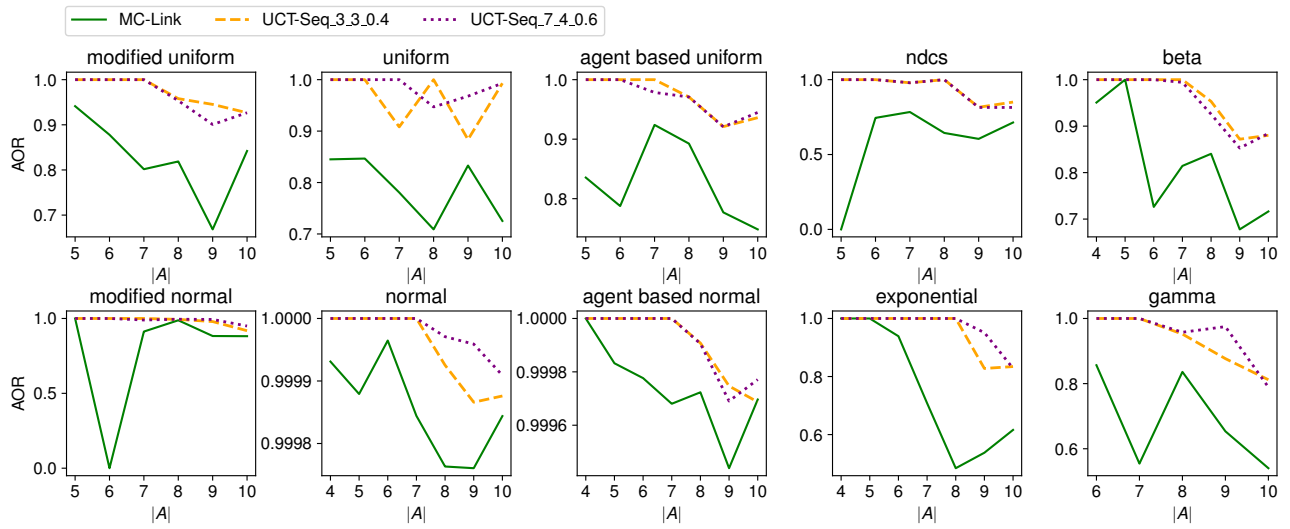


Figure 5.7: Comparison of MC-Link and UCT-Seq solution quality when $h = 4$ and $\mathcal{R} = \mathcal{R}_3$. We depict the results of UCT-Seq for the settings $\bar{b} = 3$, $\bar{d} = 3$, $\gamma = 0.4$ (line UCT-SEQ_3_3_0.4) and $\bar{b} = 7$, $\bar{d} = 4$, $\gamma = 0.6$ (line UCT-SEQ_7_4_0.6).

in normal-based distributions. Regarding the MCTS parameters, once again one can see no clear winner while comparing the quality of the solutions and the corresponding running time.

Another feature to consider is how much memory the proposed algorithms require. Based on the experiments in Section 4.2, we know SDP requires a considerable amount of memory to compute a solution. We expect the same to hold for UCT-Seq as we maintain in memory a tree of coalition structures, even though we remove from it the nodes that cannot improve the current result in any way. In our analysis, we record the peak of memory used by each algorithm for the same set of experiments as the one described above. We chose the NDCS-based valuations for this comparison.

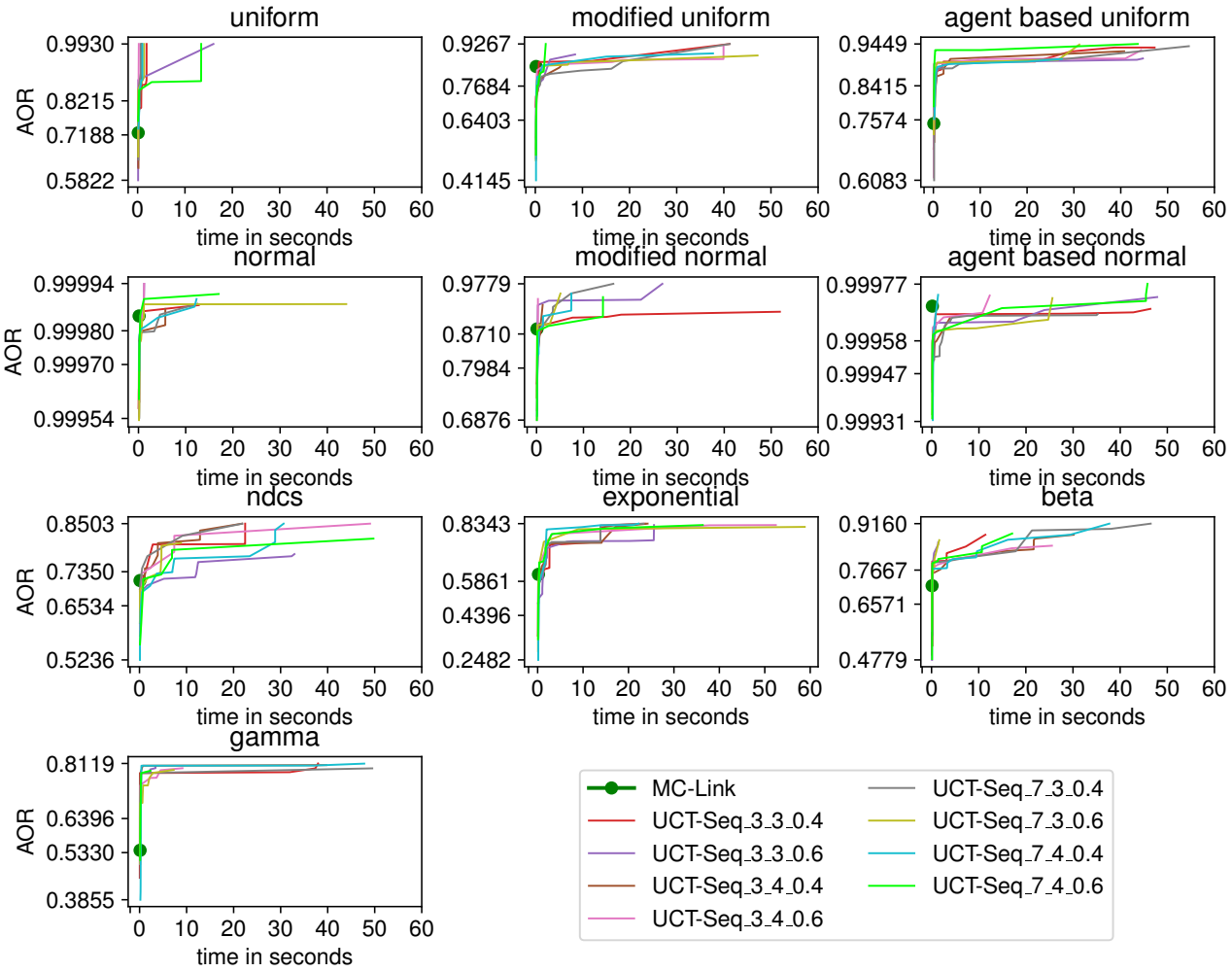
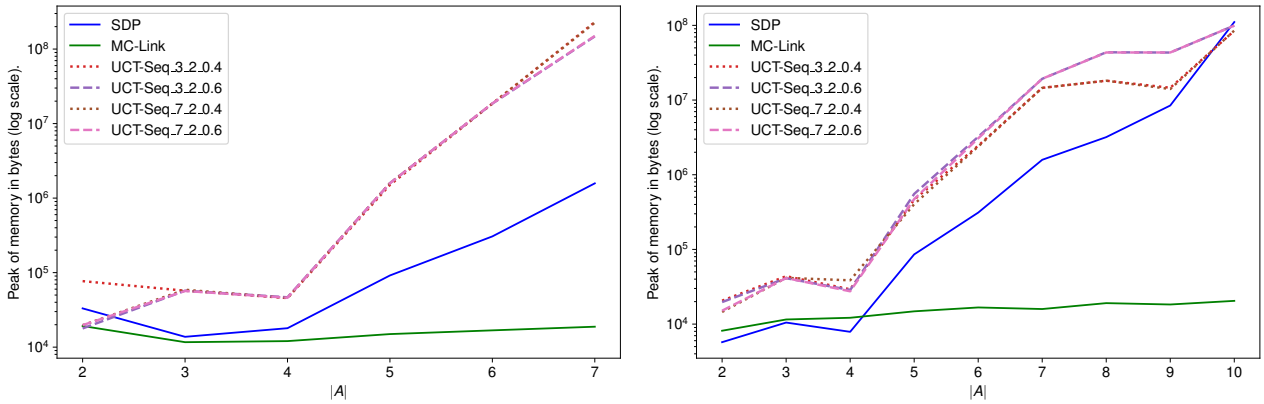


Figure 5.8: Comparison of MC-Link and UCT-Seq running time and solution quality given an optimal solution when $n = 10$, $h = 4$ and $\mathcal{R} = \mathcal{R}_3$. Each UCT-Seq line follows the same pattern UCT-SEQ_$\bar{b}$_$\bar{d}$_γ.

First, we analyse all algorithms for a sequence containing only two CFGs. In the results depicted in Figure 5.9, one can see that MC-Link consumes much less memory, by several orders of magnitude, than both SDP and UCT-Seq. This is expected as MC-Link keeps in memory only a sequence of tables (one per game) and the maximum size of a single table is n^2, which is used to store a coalition structure of singleton coalitions. Regarding UCT-Seq, the simulation parameters play almost no role in the peak of memory amount. In fact, the only parameter that influences it is the exploration factor. In general, the simulation parameters do not require anything to be kept in memory (apart from the list of compatible CSs regarding $\mathcal{R}$). When $\mathcal{R} = \mathcal{R}_1$ (Figure 5.9a) both factor values ($\gamma = 0.4$ and $\gamma = 0.6$) converge to the same amount of memory when the number of constraints in $\mathcal{R}$ is minimum. This suggests new nodes are always being added to the tree during the search. When it comes to $\mathcal{R}_3$ (Figure 5.9b), we note a difference in memory consumption regarding the exploration factor. For instances containing $7, \ldots, 9$ agents, the exploration factor of 0.6 consumes more memory than the one of 0.4. When $\gamma = 0.6$, a node requires less visits to it to expand its list of child nodes. It suggests that in $\mathcal{R}_3$,



(a) $\mathcal{R} = \mathcal{R}_1$. (b) $\mathcal{R} = \mathcal{R}_3$.

Figure 5.9: Comparison of the peak of memory used by SDP, MC-Link and UCT-Seq as we scale up the number of agents for 2 CFGs.

the tree gets wider as there exist more paths that are not reaching level h yet. Therefore, no node can be removed.

Interestingly, SDP matches the peak of memory consumption of UCT-Seq for 2 CFGs and 10 agents (i.e., when $\mathcal{R} = \mathcal{R}_3$). To better discuss this trend, we introduce in Figure 5.10 the peak consumption for 10 CFGs when $\mathcal{R} = \mathcal{R}_1$. UCT-Seq peak of memory converges to the same amount when the instances have more than 4 agents. Approximately, the same plateau is achieved by SDP in the instance containing 10 agents in Figure 5.9. It seems that the tables kept in memory to store the values of every coalition (each coalition has 10 different values) are limiting the amount of memory the algorithms can use. We can conclude that UCT-Seq is the algorithm that consumes the most memory among SDP, MC-Link, and also the brute-force algorithm.

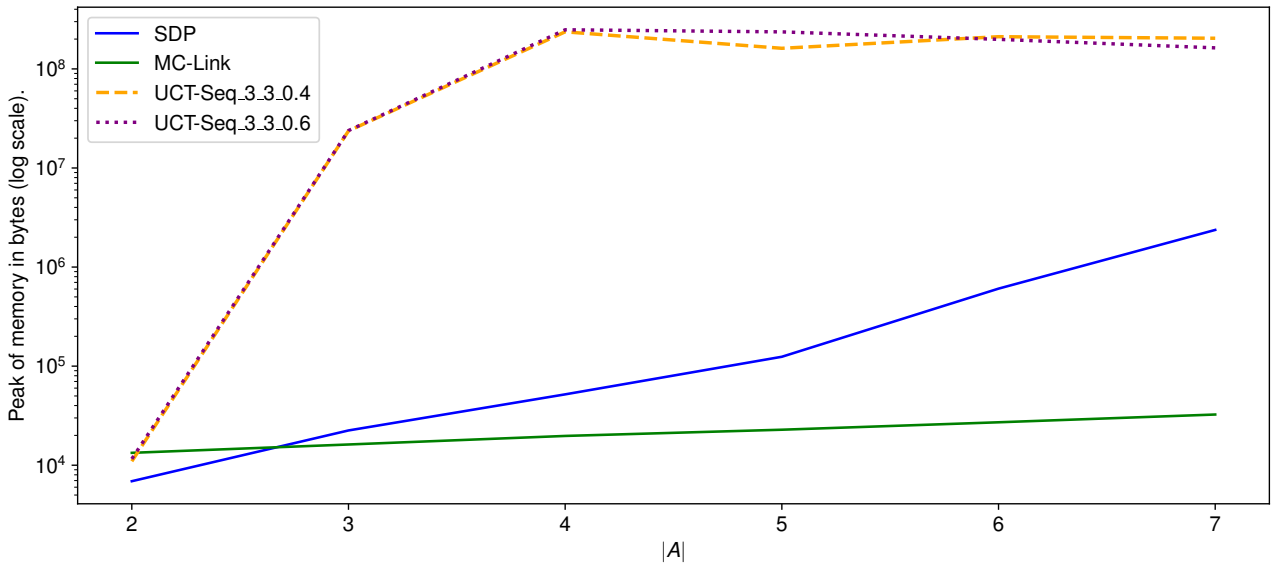


Figure 5.10: Comparison of the peak of memory used by SDP, MC-Link and UCT-Seq as we scale up the number of agents for 10 CFGs and $\mathcal{R}_1$.

In all experiments conducted in the small instances above, as the number of agents increases, the quality of the solutions computed by both algorithms decreases. We expect it to continue for larger instances. In our next set of experiments, we shall scale up the number of agents. However, it is not feasible anymore to compare MC-Link and UCT-Seq solutions with an optimal one. Instead, we evaluate the improvement in quality using MC-Link as baseline.

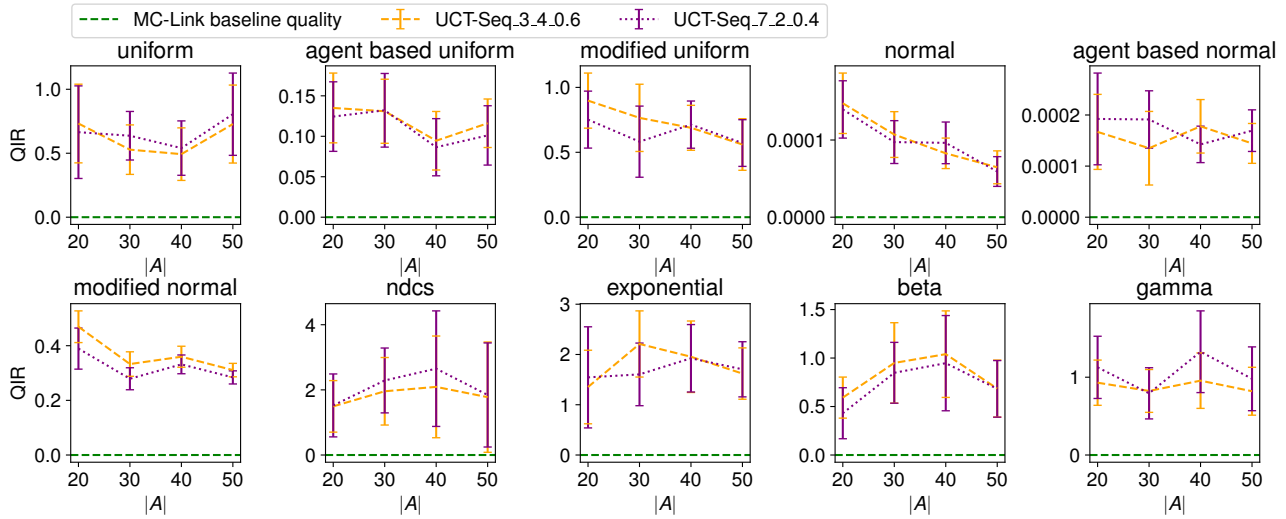
5.3.3 Scaling Up Comparison

In this section we scale up the number of agents in the experiments. We set the number of agents to 20, 30, 40, and 50, and the number of games to 2, 4, 6, and 8. We limit the experiments to 8 CFGs as we empirically noted that UCT-Seq demands a considerable amount of time to output a solution for this setting (we introduce the results below). However, we do not expect real-world applications to require several games.

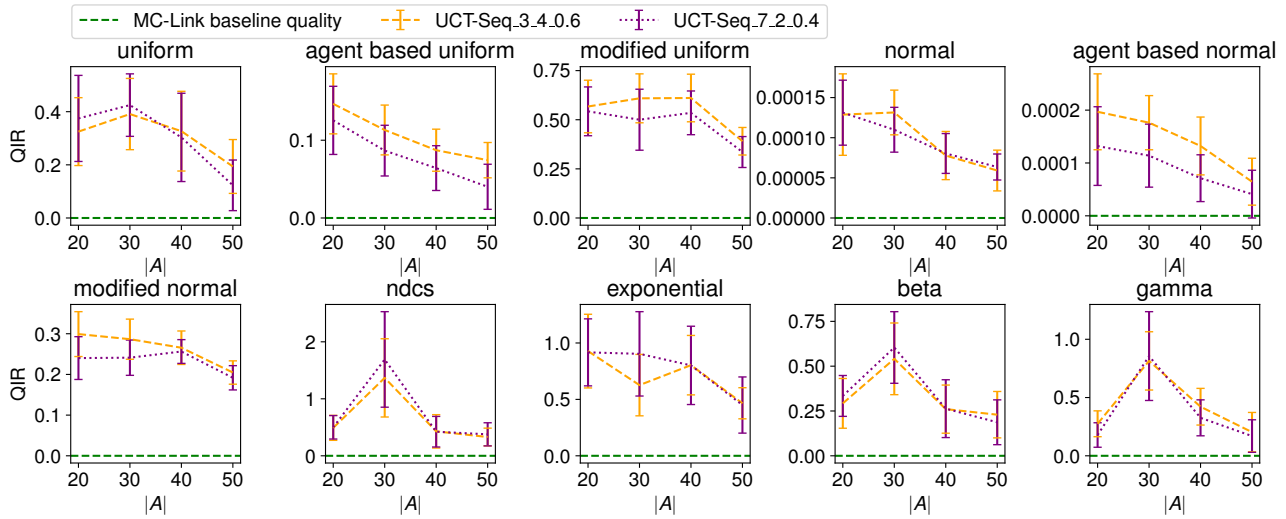
We follow our former experiments and consider only $\mathcal{R}_1$ and $\mathcal{R}_3$ (Definition 20). Moreover, the coalition values are drawn from the 10 distribution introduced in Section 5.3.1. All games in an instance draw values from the same distribution. However, storing the value of every coalition in a table is no longer feasible due to the number of agents and games. Instead, we draw a value from a given distribution on demand, that is, every time the valuation function is called. This procedure does not allow us to compare the precise quality of the solutions outputted by both algorithms, rather to estimate their values. To do so, we run each experiment many times and average their results. We set the number of repetitions to 30.

To generate the constraints, for each experiment containing a different number of agents and games a new set of constraints is computed. This differs from the former experiments because all instances with valuations from different distributions use the *same* set of constraints. That is, we only change how we compute a value for a coalition. We create all interaction graphs randomly with a probability of 60% of an edge to connect any two agents. Regarding the pivotal agents, we randomly pick, from A, q agents and insert them into the corresponding set of pivotal, where $q \sim U(0, \lceil n \times 0.2 \rceil)$.

In our first set of experiments we set the number of games to 4 and introduce the results in Figure 5.11. Recall that we use the QIR metric (Definition 26) to compare the quality of the outputted solutions. Moreover, we set the UCT-Seq time budget to 60 seconds. The error bars in the charts below stand for a confidence interval of 95% calculated on the 30 solutions based on the t distribution. One can see that as we increase the number of agents UCT-Seq maintains a better quality performance than MC-Link. In fact, that holds for all distributions; even for $\mathcal{R}_1$ (Figure 5.11a), in which UCT-Seq computed poor solutions in the experiments in Section 5.3.2. One can see as well that for $\mathcal{R}_3$ (Figure 5.11b) the general quality improvement ratio decreases when compared with $\mathcal{R}_1$. This is expected as the FCSSs of great values might



(a) $\mathcal{R} = \mathcal{R}_1$.

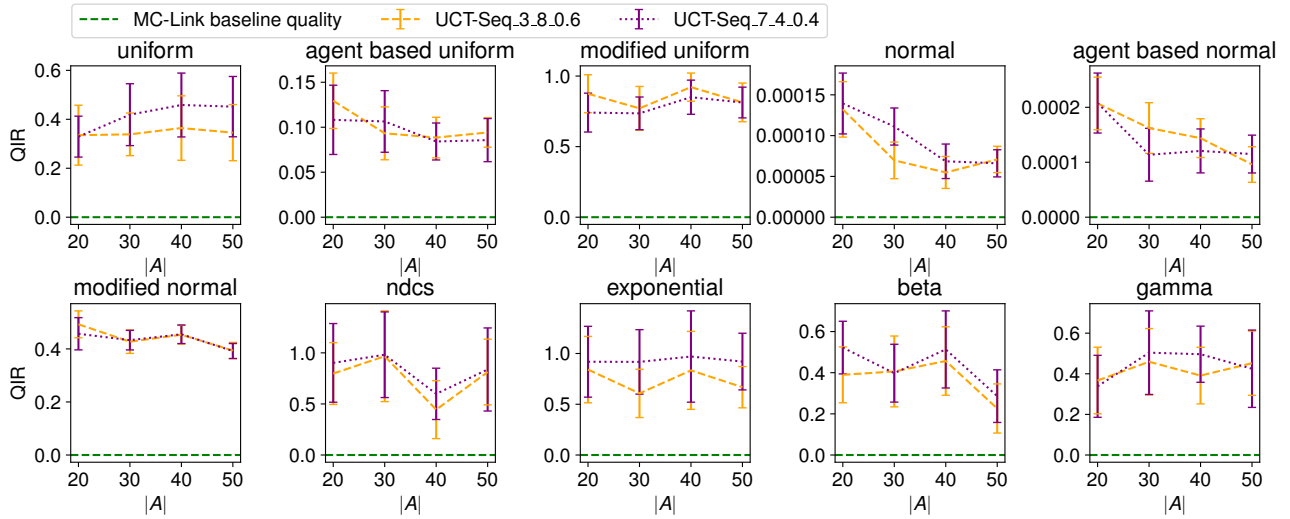


(b) $\mathcal{R} = \mathcal{R}_3$.

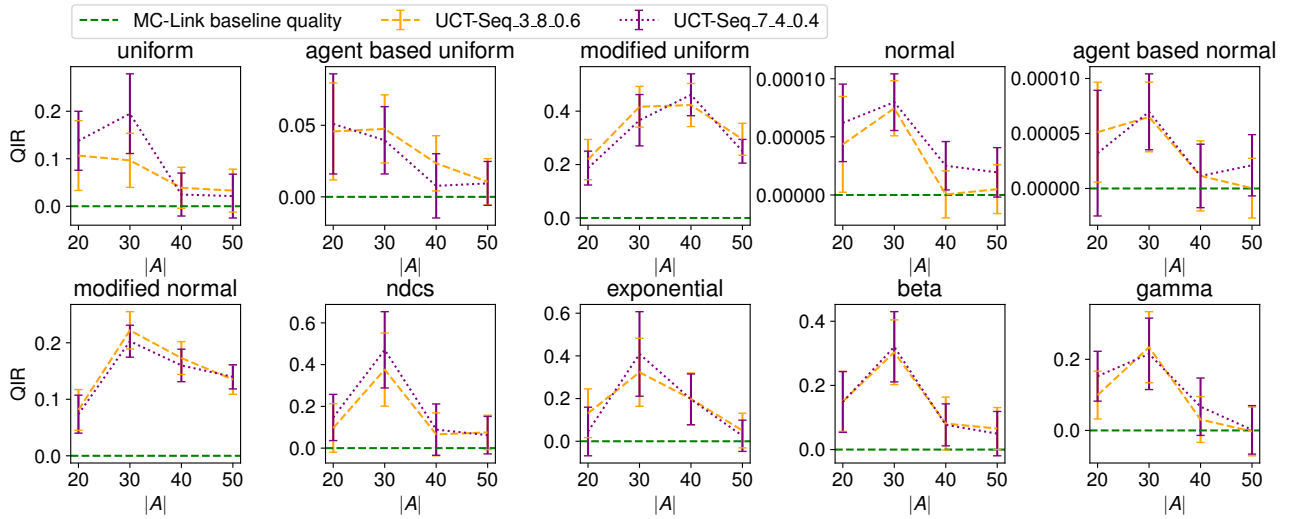
Figure 5.11: Comparison of MC-Link and UCT-Seq performance as we scale up the number of agents for 4 CFGs.

have been constrained by $\mathcal{R}_3$. Regarding the UCT-Seq parameters, one can see no clear difference to determine which configuration is the best. Depending on the distribution at hand, a configuration leads to an estimated solution of better quality than the other configuration. Moreover, the error bars in the charts overlap, showing solution qualities in both configurations may lay within the same range.

In our second set of experiments, we increase the number of games to 8 and depict the results in Figure 5.12. This time the time budget is set to 120 seconds. UCT-Seq remains computing solutions of better quality for $\mathcal{R}_1$ in all distributions (Figure 5.12a). However, when it comes to $\mathcal{R}_3$, one can clearly see that the UCT-Seq performance decreased. In fact, in all distribution but in the modified uniform and normal, the confidence interval dropped below the MC-Link baseline quality. In those two distributions, an additional value is summed up to the coalition value based on a probability of 20%. It suggests that UCT-Seq is more likely to detect good combinations of coalitions in valuations that show that property. Again, no clear



(a) $\mathcal{R} = \mathcal{R}_1$.



(b) $\mathcal{R} = \mathcal{R}_3$.

Figure 5.12: Comparison of MC-Link and UCT-Seq performance as we scale up the number of agents for 8 CFGs.

best combination of UCT-Seq parameters. As a matter of fact, all combination of parameters seem to lead to similar results in all experiments conducted.

Our next step is to evaluate the running time of both algorithms as we scale up the number of agents. To do so, we run each algorithm 30 times and average the required time to output a solution. For UCT-Seq, again, we set the time budget to 60 seconds for instances containing 2 and 4 CFGs, and 120 seconds for instances containing 6 and 8 games. We record the precise moment (in seconds) that a new FCSS was found and use that record during the comparison. For instance, suppose that for an instance containing 2 games (60 seconds time budget) the best solution was found after 30 seconds. Therefore, we store 30 seconds as the final record. Moreover, we keep UCT-Seq running until it finds at least one solution for the problem; even if the time budget is finished. In this experiment we consider only the NDCS distribution as it provide unbiased search spaces (Rahwan et al., 2009b). We depict the running time results in Figure 5.13.

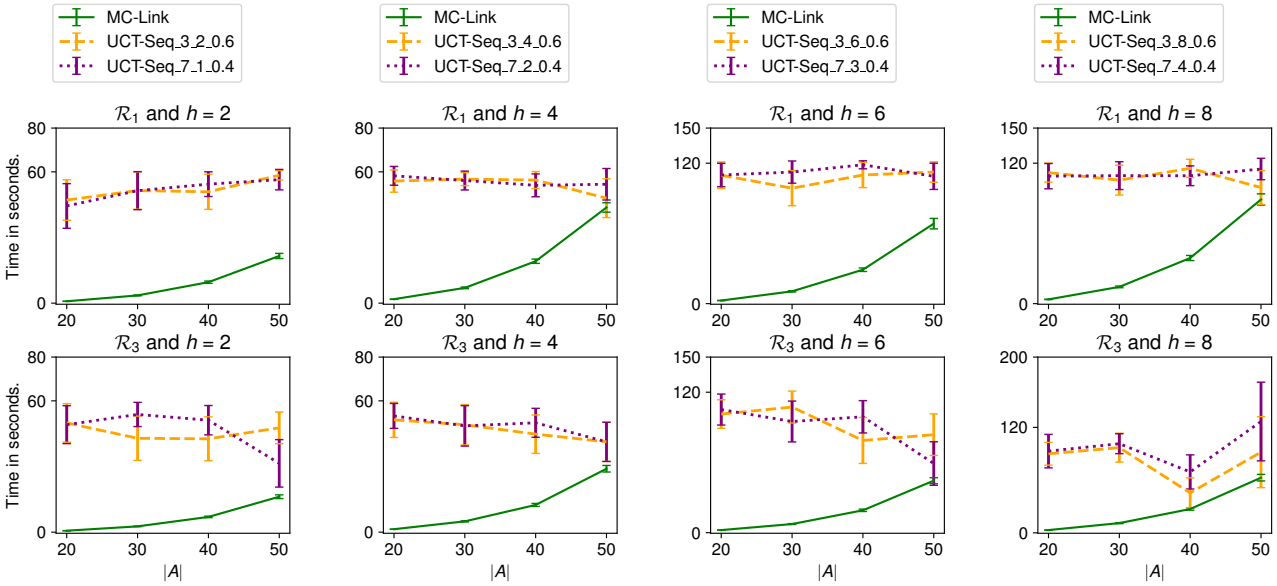


Figure 5.13: Comparison of MC-Link and UCT-Seq running time as we scale up both the number of agents and games.

One can see an increase in MC-Link running time to compute a solution as we vary the number of agents. Thus, the number of games is not the most significant factor that impacts the running time performance of MC-Link. When it comes to UCT-Seq, it requires more time than MC-Link to find a good solution. In fact, UCT-Seq keeps finding better solutions even when the time budget is almost out. This is particularly the case for $\mathcal{R}_1$ (first line of charts in Figure 5.13) where there exist many compatible coalition structures. On the other hand, for $\mathcal{R}_3$ (second line of charts in Figure 5.13) when $h = 8$, we notice that the time budget of 120 seconds is not enough for computing solutions in that instance because the confidence interval indicates expected running time above the time threshold. In fact, when $n = 50$, in 20 out of 60 repetitions (2 configurations of MCTS parameters) the UCT-Seq required more than 120 seconds to compute a solution. Regarding UCT-Seq parameters, one can see again no clear difference in performance regarding the configurations. However, this behaviour can also be related to the time budget. Increasing it reasonably could be determinant for both configurations to reach the same outcome.

5.3.4 Discussion

Having concluded the experiments with both heuristic algorithms, a number of conclusion can be stated. In general, given the same amount of time, MC-Link computes better solutions, in terms of the value of the sequence, than UCT-Seq. However, MC-Link can easily be caught in a local optimum and, more importantly, it is more sensible to constraints than UCT-Seq, in particular the ones regarding sizes. In MCTS-based algorithms, one continues

trying different solutions as long as there is time available for that and therefore, it can avoid local optima. However, to do so, UCT-Seq requires more time than MC-Link.

One should choose first MC-Link as an algorithm for solving SCFG-based problems whenever it can compute a solution for them. Although both algorithms eventually output a solution (in the UCT-Seq case, not only a solution, but an optimal solution as well), MC-Link requires less time to do so. In fact, at the time of writing, we lack mechanisms to estimate the UCT-Seq running time for any given application. The only alternative is to estimate it beforehand empirically.

As we have noticed throughout this section, there is no general configuration of MCTS parameters that determines a good result for every problem. This decision is domain dependent. Nonetheless, we also noted that different combinations of parameters generally share similar quality solutions as they fall within the same range, given a 95% confidence interval. This suggests that the parameters we have experimented with do not have a huge impact on the UCT-Seq performance. Further investigations are required to determine the reasons for this behaviour. For instance, it may be relevant to compare the number of compatible coalition structures that are being found in both simulation and generation procedures.

Chapter Remarks

In this chapter, we proposed two different heuristic algorithms to compute an FCSS. The first algorithm, named MC-Link, is based on a hierarchical clustering method in which it starts off from the sequence of CS of singletons and mergers any two coalitions if the resulting merger is compatible with the preceding CS. Then, it advances to the next CS in the sequence and continues carrying out the same procedure until no merger is allowed. Our second algorithm is based on Monte Carlo Tree Search (named UCT-Seq) and was developed to avoid local optima in which MC-Link might be caught on. Two main challenges come up when designing an MCTS approach for SCFG-based problems. The first one is to generate compatible actions with the current state (i.e., transition from one CS to another). The other challenge regards the simulation of a path in a subtree. We proposed domain independent methods to solve both challenges. We experimented extensively with both algorithms and concluded that usually MC-Link is faster than UCT-Seq to compute a solution. However, as UCT-Seq can explore different parts of the search space, given enough time, UCT-Seq eventually outperforms MC-Link in terms of the quality of the outcome. Those two algorithms conclude Contribution 6 and 7.

6. CASE STUDY

In this chapter, we show how one can use the SCFG framework to model a real-world application. To this end, we consider a disaster response problem in which coalitions of resources (e.g., firefighters, rescuers, and robots) must respond to a disaster incident. Disaster response is a sensitive topic due to its major social and economical impact on the affected communities. Usually, a team of responders is required to perform several tasks in the damaged environment (hot zone), for instance, save victims. In case there is no prior knowledge about the current situation (or specific tasks) that the responders will have to deal with, then they must be prepared for a large number of different tasks. In this context, we have to distribute the set of available resources into teams; each possessing the required expertise and abilities to comply with the designated objectives. In general, this sort of application can be modelled by a single CFG. However, our main goal is to model the complete chain of command of a given response operation following the constraints imposed by it, which makes the use of a single CFG no longer possible (details in the sections below).

To bring about our goal, we first introduce the ICS (Irwin, 1989), a management system designed for disaster response operations, in Section 6.1. We then, in Section 6.2, simplify the ICS problem to consider only a fraction of its overall hierarchy: the Operations Section. This section is responsible for acting upon the damaged zone and is organised into a three-level hierarchy. In Section 6.3, we introduce a concrete example of a disaster response operation: the Roaring River Scenario (RRF). It is used as part of the ICS training course (U.S. Department of Agriculture, 2021). Having introduced it, we show in Section 6.4 how to model this scenario and present two variations of the RRF problem. In the first variation, in Section 6.5, we assume that resources acting upon the damaged zone can form coalitions only with the superiors (i.e., a personnel resource responsible for a set of resources) determined by the Operations Section chief. In the second variation, we relax that constraint and allow the resources to form coalitions on their own (i.e., without the presence of superiors in it). This is the main topic in Section 6.6. Finally, in Section 6.7, we discuss the main findings and achievements in our approach to model the problem of assigning resources throughout a chain of command.

6.1 The Incident Command System Framework

The Incident Command System (ICS) (Irwin, 1989) is designed to deal with multiple organisations that respond to a disaster incident. Multiples incident managers are intended to work together seamlessly. It was initially developed during 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 (FEMA, 2017).

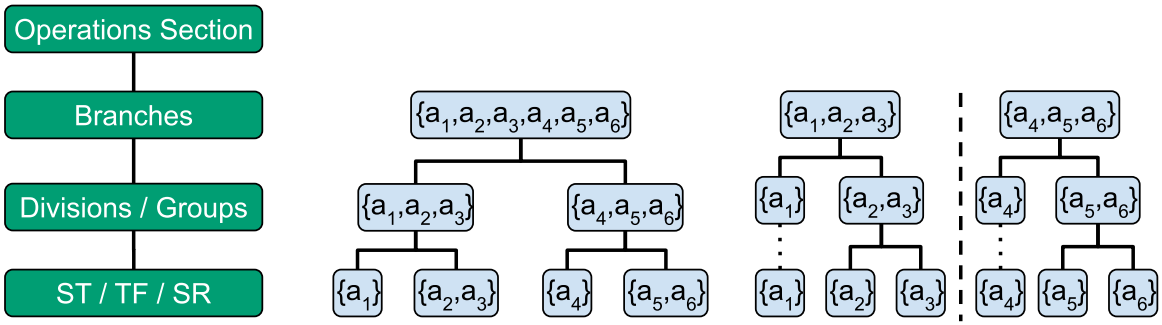


Figure 6.1: The Operation Sections hierarchical organisation on the left-hand side. On the right-hand side, two sequences of coalition structures of length three. Each of them represents a three-level hierarchy of agents starting at different CSs.

The ICS establishes several general guidelines to respond to disaster incidents. Among them, two are of particular importance to our work.

Span of Control: the ratio of command among superiors and subordinate units: $\frac{1}{\lambda}$; meaning that one superior is responsible for at most λ subordinate units.

Modular Organisation: the organisational structure is adjusted according to the complexity of the current event following the span-of-control guidance. For instance, if the event occurs over a large area, then the span of control for a supervisor may be reduced (e.g., from five to two or three subordinate units per supervisor). This ratio is closely related to the characteristics of a disaster event.

While the ICS's overall organisational structure is much richer (five main sections are defined), we focus on the Operations Section (OPS) as it already contains several key components. It acts upon the damaged area of the disaster event, and its responsibilities include achieving command objectives, tactical operations, contingency planning, among others. We depict in Figure 6.1 the OPS hierarchical organisation. Branches, at the upper level, are divided into divisions/groups. Divisions are allocated to geographical areas of the incident (e.g., because of different jurisdictions). Groups represent a functional aspect of the operation (e.g., rescue group). The number of branches, divisions, and groups depends on the span-of-control ratio (referred as λ here), which can make the units more manageable by their superiors. At the bottom level, resources are organised into Task Forces (TF), Strike Teams (ST), or even Single Resources (SR). Those are components of a partition of divisions and groups. A resource can be a person or an individual piece of equipment (Irwin, 1989). Heterogeneous resources constitute task forces, whilst strike teams are homogeneous. Usually the number of organisational levels follow a *bottom-up approach*, in which once a superior is overwhelmed a new level is created to make it fit into the span of control.

Applying the span of control feature to the coalition structure generation problem 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. The authority system within a section is as follows:

chief: a person responsible for a section;

director: a person responsible for a branch;

supervisor: a person responsible for either a group or division;

leader: a person responsible for either a task force, strike team, or single resource;

We shall use the term *superior* to capture the notion of leadership in a unity.

Figure 6.1 illustrates the problem we are facing: a series of coalition structures, further refined at each subsequent level. The main problem is to find the best assignment of agents into branches, groups, etc., respecting several constraints coming from the overall ICS process (size of groups, superiors in each group, etc.). The question is then, how to represent all constraints induced by an ICS instance over the sequence of coalition structures (one CS per hierarchical level). We return to this problem in the section below.

The disaster response operation is guided by Incident Objectives (IO). They lead to activities that must be carried out; for example, an IO to establish situational awareness. During the management cycle (FEMA, 2017), given the current IO, the superiors should identify the requirements for accomplishing them and then gather the relevant resources. To provide an efficient and effective resource identification (even across different jurisdictions), the ICS recommends to classify them according to their characteristics.

Capability: the core capabilities for which a resource is useful. Examples of core capabilities are critical transportation and situational assessment (FEMA, 2015).

Category: the function for which a resource would be most useful. Examples of functions are firefighting, health and medical (FEMA, 2017).

Kind: a broad characterisation of what a resource is. They are classified as either personnel, team, facility, equipment or supply.

Type: describes a resource's level of minimum capability to perform its function. One determines a resource's type based on the resource kind and the mission envisioned for it. To do so, the level of capability will follow from the resource's size, power, and capacity (for equipment) or experience and qualifications (for personnel and teams). Types are organised in a decreasing order Type $1 >$ Type $2 > \ldots >$ Type k in which Type i is a higher capability than Type $i + 1$ for $1 \leq i < k$ (FEMA, 2017).

To illustrate the core idea, consider Example 9. FEMA provides a set of predefined types in the Resource Typing Library Tool (RTLT)[1].

[1] https://rtlt.preptoolkit.fema.gov

Example 9 (Ambulance Ground Team). Consider an ambulance ground team as introduced in (FEMA, 2019, page 193). As *capability*, it provides basic life support and transports 2 non-ambulatory patients. It is tagged with *category* emergency medical services. It has resource *kind* team. Finally, the resource is tagged as Type 3, which means a crew of 2 personnel.

The success of the ICS framework in practice will depend on many factors as highlighted by Jensen and Waugh Jr (2014, Section 4). Out of those factors, we emphasise:

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 resources. To this end, we formulate the problem of forming the OPS hierarchy as a SEQSVS problem.

6.2 The Operations Section as a SEQSVS Problem

We intend to model the distribution of resources throughout the hierarchy in such a way that the span of control is enforced. Moreover, we model the problem as a hierarchy instead of a single level (a single CS that represents the last hierarchical level) for twofold reasons:

1. We need to distribute the superiors to the coalitions that they can extract the most. We note that a coalition acting upon the hot zone (i.e., a TF, ST, or SR in the last hierarchical level) is influenced not only by its corresponding leader, but also by the superiors of upper levels. The same reasoning applies to the leaders in which a good relationship is expected throughout the chain of command. This is motivated by Jensen and Waugh Jr (2014, Section 4) in which the authors discuss the importance of superiors.
2. We need to enforce the span of control between subsequent levels. Doing so, we form the chain of command.

We showed in (Krausburg et al., 2021b) that the hierarchical structure required by the Operations Sections, designed to enforce the span of control, can be modelled by the SCFG framework. Given this result, we embark on the practical modelling of this important section of the ICS. First, we establish what is expected from a single CFG. Then, given a sequence of games, we determine the interdependence among their solutions.

6.2.1 A Single CFG Problem for Disaster Response

Usually, disaster response operations inspire problems in which one assigns agents to accomplish a set of tasks announced by the command centre. Ayari et al. (2017) model a disaster response incident in which tasks have different priorities, deadlines and are composed of sub-tasks. To carry out a task, an agent needs a number of resources, and therefore the agents responding to the event form coalitions to meet a task requirement. Mouradian et al. (2017) focus specifically on robots. Each robot has two vectors of capabilities, one for sensing (e.g., cameras) and other for acting (e.g., arms). Similarly, to complete a task, the required sensing and acting capabilities must be met. To deal with tasks announced during a flood-inspired disaster event (e.g., to collect samples of water), Basegio and Bordini (2018) investigate composed tasks; each demanding particular roles to be carried out. Roles are mapped onto capabilities that are possessed by the robots acting upon the flood incident. Ramchurn et al. (2010) focus on the allocation of tasks given spatial and temporal constraints. This game is motivated by the RoboCup Rescue competition which is inspired by an earthquake incident in Japan.

The coalition structure generation problem has already been used to model a response to a disaster incident. Wu and Ramchurn (2020) model a scenario in which a satellite, powered by radioactive fuel, has crashed in a sub-urban area (this scenario was first introduced by Ramchurn et al. (2016) as an agent-based planning problem). The emergency services, that respond to this incident, are composed of medics, soldiers, transporters and fire-fighters. These are the roles. The scenario is modelled as a grid where responders must accomplish a set of rescue tasks by dropping off the *targets* at specific locations in the grid. A target can either be a victim, animal, fuel, or other resource; therefore, four targets are considered. Each task demands a set of roles in order to be carried out (a mapping targets onto roles). Each agent has a different different level of capabilities while playing each of the four roles. This information may represent its training for that role, past experience, etc. Doing so, one can estimate the performance p_k^C of a coalition C to rescue a specific target k by summing up the capabilities of each individual responder in C for each role required by target k. To determine a partition of the resources, the function $v(C) = \sum_{k=1}^{4} p_k^C$ is used. This way, Wu and Ramchurn (2020) form coalitions that can deal with as many targets as possible.

In our particular application, we consider a CFG $\Gamma = \langle A, v \rangle$, where $A = \{a_1, \ldots, a_n\}$ is the set of resources assigned to the Operations Section. Moreover, a set of roles R is used to easily identify resources, assigning them where they are needed. Each resource has a set of capabilities which allows them to adopt roles in a disaster response operation. We use the function $cap : A \cup R \to 2^{Cap}$ to map resources and roles onto a set of known capabilities Cap. Doing so, we can construct the set of roles that a resource $a_i \in A$ may adopt, $adopt(a_i) = \{r \in R \mid cap(r) \subseteq cap(a_i)\}$. Given a disaster incident, the Operations Section must achieve a set of incident objectives $IO = \{o_1, \ldots, o_m\}$. Each of them demands a set of roles, $demand : IO \to 2^R$.

It should be clear now that this application can indeed be modelled using MAS organisation frameworks as the ones discussed in Section 2.1.2.

A CFG Γ is induced by a Sized Valuation Structure $\pi = \langle G, S, Z \rangle$, where $G = \langle A, E \rangle$ is an interaction graph and $S \subseteq 2^A$ is a set of pivotal agents for that particular game. The set Z establishes how many resources may participate in any coalition $C \subseteq A$. Moreover, for each special agent in the game $a_i \in S$ (i.e., a superior), we assume he/she is responsible for accomplishing a set of incident objectives IO. We use the function $resp : S \to 2^{IO}$ to select the objectives assigned to a given superior. Note that superiors might share an incident objective. For instance, two task forces are deployed to rescue victims at different locations. Both superiors might share the same general goal of rescuing victims. Now, for a superior $a_i \in S$, we can construct the corresponding set of roles a_i expects to have at its disposal to achieve its assigned IOs $expect(a_i) = \{r \in R \mid o \in resp(a_i), r \in demand(o)\}$.

In this modelling we assume that a disaster response management system is available in which the information above can be queried. This system will provide the set of capabilities Cap as well as the set of roles R available for the disaster response operation. This assumption is not disregarded from real-world operations as agencies that undertake them use systems to help managing resources. For instance, consider the Interagency Resource Ordering Capability (IROC)[2] system used by agencies in the USA to manage resources as well as the incident itself. Unfortunately, anonymised information about the resources was not disclosed.

Moreover, the use of capabilities is motivated by the requirements that each role expects from a resource. For instance, consider the position ***Fire Behavior Analyst***[3] defined by the National Wildfire Coordinating Group in the USA. For this particular role, a resource is expected to have concluded a set of training courses, a certain level of fitness condition, and specific experiences[4]. As a second example, consider the position ***Wildland Firefighter*** defined in the RTLT[5]. This role demands a number of concluded training courses, a certain physical condition, and professional licenses. All of these requirements become capabilities in our model.

6.2.2 A Sequence of CFG Problems for Disaster Response

To connect each level of the hierarchy we use the SEQSVS model $\mathcal{G} = \langle A, \mathcal{H}, \Pi, \mathcal{R} \rangle$. The Operations Section contains three hierarchical levels: branch, group/division, and TF/ST/SR levels. As stated in Section 6.1, the intuition behind the ICS hierarchy is to build it following a bottom-up approach. Once leaders are overwhelmed, new hierarchical levels are created. We follow this intuition and model the hierarchy in a bottom-up way as well. That means, our first CFG corresponds to the bottom level of the hierarchy (i.e., TF/ST/SR level). Moreover, we use

[2] https://famit.nwcg.gov/applications/IROC
[3] https://www.nwcg.gov/positions/fban
[4] https://www.nwcg.gov/positions/fban/position-qualification-requirements
[5] https://rtlt.preptoolkit.fema.gov/Public/Position/View/4-509-1028

an additional CFG to keep track of the span of control of the OPS chief. That will be the fourth game in our sequence in which only the grand coalition will be allowed to form. Therefore, $\mathbf{\Gamma} = \langle \Gamma_1^{\pi_1}, \Gamma_2^{\pi_2}, \Gamma_3^{\pi_3}, \Gamma_4^{\pi_4} \rangle$.

The set of pivotal agents in each game are the corresponding superiors at each hierarchical level. We shall use throughout this chapter L to denote the set of all superiors. That is, $L = \bigcup_i^4 S_i$. Regarding the size constraints, we set $Z_2 = Z_3 = \{1, \ldots, n\}$. In the first game, a coalition can have at most $\lambda + 1$ members (a superior and λ resources). Therefore, $Z_1 = \{1, \ldots, \lambda + 1\}$. In the last game, only the grand coalition is allowed, $Z_4 = \{n\}$.

To connect the outcomes of the CFGs in the sequence of games, we design $\mathcal{R}$ as follows.

Definition 27 ($\mathcal{R}^{\text{BASIS}}$). Given $CS, CS' \in \boldsymbol{\mathcal{CS}}^A$, the pair $(CS, CS') \in \mathcal{R}^{\text{BASIS}}$ iff:

1. $|CS| > |CS'|$; and
2. for all $C' \in CS'$ there exist at most $\lambda + 1$ coalitions $C \in CS$ such that $\bigcup_j C = C' : 1 \leq j \leq \lambda + 1$;

Moreover, $(\emptyset, CS) \in \mathcal{R}^{\text{BASIS}}$ for all $CS \in \boldsymbol{\mathcal{CS}}^A$.

Note that, without the fourth game (i.e., the OPS level) the CS of the branch level CS_3 could have any number of coalitions. In fact, it could easily overwhelm the OPS chief. By adding the fourth game, in which the CS CS_4 contains only the grand coalition, we enforce that $|CS_3| \leq \lambda + 1$.

We delay on introducing the interaction graphs as they are specific to the problem at hand that we want to solve. In this chapter we consider two different problems. Thus, in Sections 6.5 and 6.6 we introduce how we model the remaining constraints for each variation of the main problem; which we introduce below.

6.3 The Roaring River Flood

As an application of SCFG, we aim to model a fraction of the hierarchy provided in the the Roaring River Flood (RRF) scenario (U.S. Department of Agriculture, 2021). It is a part of an ICS training course of the United States Department of Agriculture (USDA). In this scenario, due to heavy rains, a severe flooding is now inundating the roaring river valley. Of special concern are:

(i) the contamination of food processing plants;

(ii) the heavy damage to a USDA fruit fly research facility which determined the release of thousands of fruit flies; and

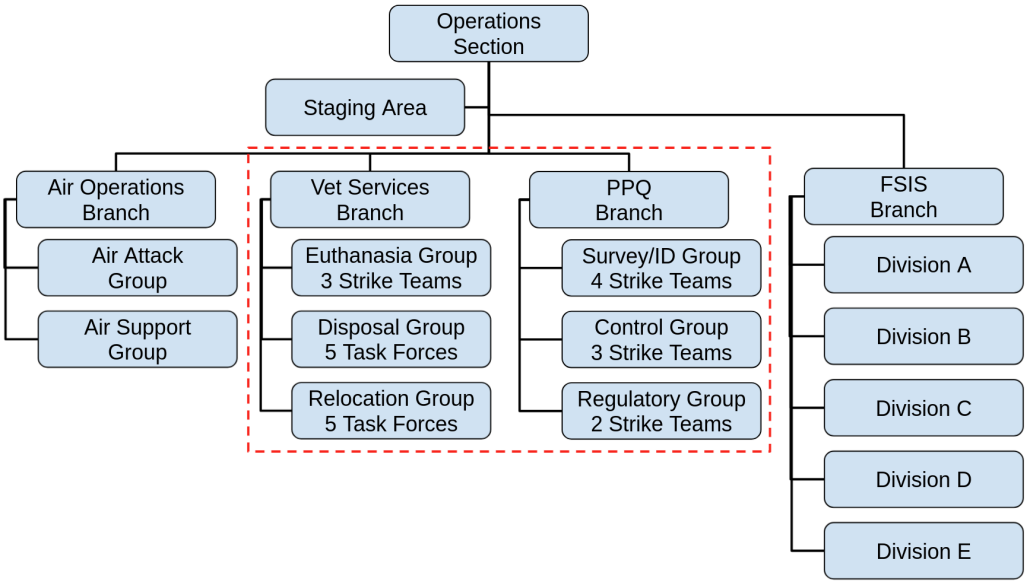


Figure 6.2: The hierarchical structure of the Operations Section for the Roaring River Valley scenario.

Source: U.S. Department of Agriculture (2021)

(iii) the widespread livestock losses.

The entire Operations Section for this particular scenario is depicted in Figure 6.2. However, we focus on modelling only the Vet Services and PPQ branches (the dashed red square). These two branches offer a good balance between complexity and easy explainability.

We extract the incident objectives directly from the ICS training course (U.S. Department of Agriculture, 2021); in total, six are listed out. Each objective relates to a particular group in the hierarchy in Figure 6.2. They are as follows:

- euthanize suffering animals;
- dispose of animal corpses;
- identify relocation sites and relocate animals;
- control the movement of host material;
- eradicate fruit flies; and
- survey and identify fruit flies.

The last goal contains two main objectives that are unified for the Survey/ID Group (see Figure 6.2). In fact, in the training course description (U.S. Department of Agriculture, 2021), 3 strike teams are responsible for surveying fruit flies and 1 strike team has as its main goal identifying fruit flies. Therefore, in the remainder of this chapter, we consider seven main IOs as introduced in Table B.3.

Although the RRF scenario provides a good explanation of the disaster response operation, it does not provide all the necessary information to model and solve the problem of distributing resources to act upon the damaged zone. For instance, it does not provide which resources are identified and available for that particular operation. By identified resources we mean resources that are classified according to the FEMA specification available in Section 6.1. To continue modelling this application, we are to assume the missing information.

The set of capabilities: given the above mentioned incident objectives, we define a list of capabilities to be available for the RRF disaster response operation. We introduce the complete set of capabilities in the appendix in Table B.1.

The set of roles: we looked up roles in the Resource Typing Library Tool[6] that are expected to provide similar capabilities as the ones determined above. In Table B.2, we introduce the list of roles along side with their respective set of capabilities Ids. Those roles are proposed based on the ICS concepts of *capability* and *type*. We note that both information can be merged into different roles in which each role is a combination of a capability and a type. For instance, the ICS resource typing establishes two types of Veterinary Assistant resource. One can introduce two different roles to represent that information; e.g., `vet-assist-1` and `vet-assist-2`. From the gathered set of roles, we extract the *expected capabilities* they are supposed to provide. This is a common procedure as the ICS specification enforces typing each resource into known skills (called *function* in RTLT). We consider only personnel-related *kinds* (i.e., *personnel* and *team*), although the ICS framework specifies also *equipment*, *facilities* and *supplies* as kinds (FEMA, 2017). In this application we let them out.

Mapping objectives onto roles: given the IOs above, we assume a set of roles that would be needed to have them accomplished. We introduce in Table B.3 the mapping of goals onto a set of roles.

Given the information provided in the ICS training course combined with the complementary one described above, we are ready to model and solve the RRF problem.

6.4 The RRF as a SEQSVS Problem

We intend to provide the full hierarchy for those two branches. Apart from the coalition structure at the last hierarchical level, we aim to model the relationship of each resource regarding its superiors. Note this is not limited to its direct superior, i.e., the leader of a particular task force or strike team. We aim to model the relationship regarding superiors of upper levels as well. This is due to the fact that their decisions affect the work developed in the hot

[6] https://rtlt.preptoolkit.fema.gov

zone. For instance, how the resources in a particular strike team in the Euthanasia Group get along with the Euthanasia Group's supervisor. Furthermore, we need to form the hierarchy keeping in mind the established span of control in order to not overwhelm any given superior.

We set the span of control to five (1 supervisor responsible for 5 subordinates units) following the maximum number of task forces in the last hierarchical level (see Figure 6.2). We assume that a set of resources (i.e., agents) has already been allocated to those branches and are to be further organised into a sub-hierarchy. Moreover, we assume that for each branch, group, strike team and task force, a superior has already been assigned by the Operations Section chief.

6.4.1 Preliminaries

To solve the RRF problem, we assume a set of 101 and 141 resources. We detail those instances in Section 6.4.2. The set of incident objectives and roles follow from Section 6.3. That is, nine roles are required in the two branches of interest and seven main incident objectives are established. We arrange the IOs in such a way that they form subsets as we move down in the hierarchical structure. For instance, the director of PPQ branch is responsible as well for achieving the goals from the Survey/ID, Control, and Regulatory groups.

Assigning Roles to the Agents

To assign a role to each agent, we do as follows.

- Each pivotal agent adopts a random *personnel* role.
- We select $|R|$ agents and assign to each of them a single different role in R. This is to make sure that all required roles are adopted by at least one agent in the RRF response operation.
- The remaining agents adopt a role based on a probability associated with each role in R.

To associate a probability value with each role $r \in R$, we compute a bound on the expected quantity of roles. Assume a pivotal agent $s \in S_1$ in game Γ_1. Recall that the first game models the TF and ST level. Given a span of control λ, let e^r be the number of expected agents adopting role r computed as follows:

$$e^r = \sum_{s \in S_1 : r \in expect(s)} \left\lceil \frac{\lambda}{|expect(s)|} \right\rceil.$$

The main idea is that we look up in the leaders (of both TF and ST) to figure out which roles they need. By summing up all their needs we can estimate the number of expected agents that

should assume each role. Note that we can do this in a bottom-up approach to find out which roles are required by supervisors in upper levels as well.

Example 10. Assume $S_1 = \{s_1, s_2\}$ and the span of control $\lambda = 2$. Leader s_1 expects roles $\{r_1, r_2\}$, and leader s_2 expects roles $\{r_3\}$. Then, in the overall operation we expect $e^{r_1} = 1$, $e^{r_2} = 1$ and $e^{r_3} = 2$ (leader s_2 expects a single role and the span of control is set to 2).

The probability of agent $a \in A$ adopting role r is then computed using Equation 6.1. Using this procedure, we first determine whether an agent is of kind *personnel* or *team* and then compute a probability for that particular subset of roles. A resource has 50% chance of being one of the two kinds.

$$P(a, r) = \frac{e^r}{\sum_{r' \in R} e^{r'}} \tag{6.1}$$

Example 11. Consider Example 10. The probabilities of agent a adopting a given role (assuming all roles to be of the same kind) are as follows $P(a, r_1) = 0.25$, $P(a, r_2) = 0.25$ and $P(a, r_3) = 0.5$.

Based on procedure described above, we assign roles to all resources in a given instance.

Valuation Functions

Prior to introducing the valuation functions for the four-game sequence, we discuss two important components of those functions. The first aspect is the relationship among resources. We assume a function that evaluates the relationship between any two agents, regardless of whether they are ordinary resources or superiors, $relationship : A \times A \to [0, 1]$. Note that the $relationship$ values does not change throughout the sequence, they remain constant. Then we can compute the relationship within any coalition using a function $relationship : 2^A \to [0, 1]$.

The second aspect addresses the roles available in a coalition. we seek to balance the available roles within a coalition. For instance, suppose a coalition C requires roles r_1 and r_2, where $|C| = 5$. It is desirable that C contains 2 agents of role r_1 and 2 agents of role r_2, instead of 1 agent of role r_1 and 3 agents of role r_2. Therefore, we define a function $disturbance : 2^A \to [0, 1]$ to calculate how unbalanced are the roles within any coalition.

We combine those two terms above and propose the valuation function in Equation 6.2.

$$v_i(C) = |C| \times (relationship_i(C) - disturbance_i(C)) \tag{6.2}$$

The first three games in the sequence $\Gamma_i \in \mathcal{H}$ calculate the coalition values based on the valuation defined above. In the last game, we set it to zero to not modify the overall value computed for the three main levels, $v_4 : C \mapsto 0$. We show the exact instantiation of both components in the sections that introduce each problem, i.e., Section 6.5 and 6.6.

The rationale behind the characteristic function in Equation 6.2 is: the more resources in a coalition the better, given that the coalition has a balanced number of roles. Usually in the literature the size of a coalition is given as a sub-additive function, i.e., the more agents in it the greater the loss in value (Bistaffa et al., 2017b). However, in our case, the span of control set for the disaster response operation already takes this matter into account. Therefore, we are only considering coalitions that are manageable by its corresponding superiors.

Metrics

To evaluate the coalitions that compose a hierarchy, we use two metrics that are related to the valuation function in Equation 6.2.

Relationship: we aim to form coalitions in which their members get along well with one another. To this end, the greater a relationship value the better the coalition.

Role Disturbance: aims to evaluate the distribution of roles within a coalition. The lesser the disturbance value the better the coalition.

How those metrics are calculated follows from the components of the valuation function which we introduce for each RRF problem at hand.

Modifications to UCT-Seq

From the results in Section 5.3.3, we noted the UCT-Seq running-time performance drops as we increase the number of agents. Indeed, in preliminary experiments, for the instances containing 101 agents and a timeout of one hour, UCT-Seq computed no solution regardless of the chosen parameters. To address this problem, we can adjust the generation process of CSs. Fortunately, a pair $(CS, CS') \in \mathcal{R}^{\text{BASIS}}$ (Definition 27) for the RRF problem requires a CS CS' in which all of its coalitions are a superset of some coalitions in CS (from the bottom level of the hierarchy up to the uppermost level). To speed up the search, we can just enforce this constraint in UCT-Seq. To do so, we modify Definition 23 to take into account the preceding CS in the sequence as follows.

Given a coalition $C \in CS$ and an interaction graph $G = (A, E')$, let F^C be the set of agents that are compatible with all agents in C according to G. That is, $F^C = \{a' \in A \backslash C \mid \forall a \in C, (a, a') \in E'\}$. Then, we modify how a SEQSVS game is induced over a 2-coloured graph.

Definition 28 (2-coloured Graph G^c induced by Γ^π). Given a CFG $\Gamma = \langle A, v \rangle$ induced by an SVS $\pi = \langle G, S, Z \rangle$ and a CS CS, where $G = \langle A, E' \rangle$, a 2-coloured graph G^{Γ^π} is a tuple $\langle V, E, c, w \rangle$ where:

- $V = \{C \in CS\}$;
- $E = \{(u, r) \mid |F^u \cap r| \neq \emptyset\}$;

- c is a function $E \to \{green, red\}$; and
- w is a function $E \to \mathbb{R}$.

We basically make the coalitions in CS the initial nodes and the execution of the generator remains the same. The mechanism above does not make the generator efficient; that is, given a level l and a CS CS it generates exactly the CSs in X_l^{CS}. However, it does avoid computing many CSs.

6.4.2 Roaring River Flood Instances

Our first step is to generate an instance containing 101 agents assigning to each resource a single role. That is, we assume that every resource's capabilities maps onto a single role. We followed the procedure described in Section 6.4.1 to assign a role to each agent. Moreover, the RRF problem, as introduced in Section 6.3, allows more agents than the 101 stated above to form a full hierarchy. That is, a hierarchy in which at the TF and ST level all leaders have λ-subordinate resources. In fact, a full hierarchy is formed with 141 resources: 31 superiors and 22×5 ordinary agents at the last hierarchical level. For this particular case, we are not interested in agents adopting randomly one of the available roles. Instead, we assign the roles required by the leaders in its exact quantity. We depict both generated distributions of roles for 101 and 141 resources in Figure 6.3.

The green bars stand for the number of roles required by the leaders at the TF and ST level of the hierarchy. That is, summing up the roles expected by each leader $s \in S_1$. The first two bars (left to right) show the number of superiors in the RRF instances (i.e., the superiors in all levels). The blue bars depict the number of agents adopting each role in an instance containing 101 resources. Similarly, the purple bars show the same information for an instance containing 141 resources though. Finally, the yellow bars stand for a bound on the quantity of roles required to solve the problem. This is calculated based on the requirement of each leader (in the bottom level) and the span of control. For instance, given $\lambda = 5$ and 2 leaders who require 5 resources of the same role r_1 each, then the bound for role r_1 will be equal to 10. The complete description regarding each role is introduced in Table B.2 and we shall use their acronyms to refer to particular roles.

Considering 101 resources, one can see in Figure 6.3 that role ABS is required by a single leader and only 5 agents are adopting it. This means, supposedly, they should form a single coalition with the corresponding leader. Quite interestingly, we can note that roles VMT and ASRTEC show a higher demand than availability. That means, some leaders will not get its desired subordinate resources. Moreover, for this particular instance, no role exceeds its demand. That is, there are enough leaders for each resource. The roles in which this index is pushed closer to the limit are ACHO, ASRT, and AADAT. For 141 resources, the available roles are supposed to match the leaders' expectations. The special case here is for the task forces

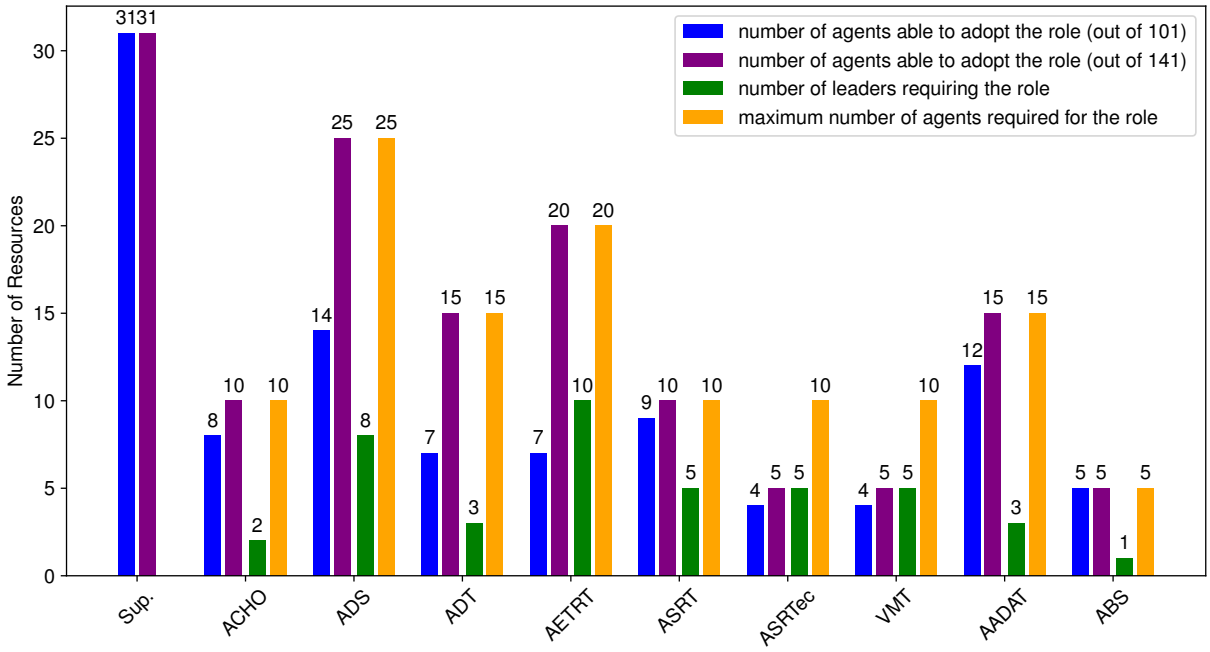


Figure 6.3: Comparison between required and available roles when resources may adopt at most one role. Refer to Table B.2 for a description of the roles.

in which the role ASRT takes on the exceeds to complete the required size. This is due to the fact that a task force in this problem requires three roles while the span of control is set to five. Therefore, a coalition at the TF and ST level cannot have two resources of each required role.

Our next step is to generate instances in which a *personnel* agent may adopt at most 3 roles. This can be the case, for instance, when a resource is of higher type (i.e., Type 1) and is skilled enough to assume any of the remaining types. We assume that resources of kind *team* must stick with a single-role-adoption rule. We understand that the corresponding team roles are too dissimilar in this application. However, it is completely feasible to consider a team that can play different types. Again, we assume instances containing 101 and 141 resources. We depict the resulting distributions of roles in Figure 6.4.

One can see a sharp increase in the available roles for the RRF response operation. In fact, the demand is exceeded in all personnel related roles (team roles end up with letter 'T' in their acronyms). However, one should note that it is not equal to the absolute number of roles available to be put into work in the damaged zone. Recall that a resource can work only for a single coalition at the TF/ST level. Once it has decided to adopt a role, the remaining two roles are no longer available to be assigned to other coalitions.

6.5 A Hierarchy of Resources for the RRF Problem

In this section, we aim to form a hierarchy of resources in which each ordinary resource is either accompanied by a superior of the corresponding level or isolated in a singleton

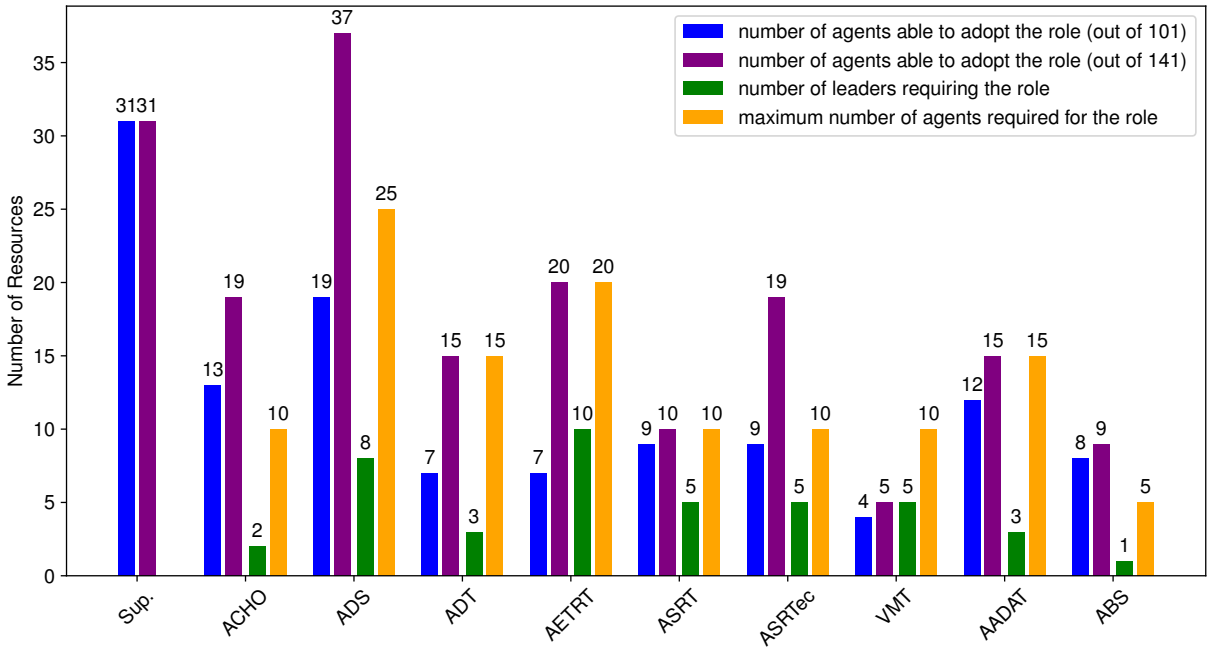


Figure 6.4: Comparison between required and available roles when resources may adopt at most three roles. Refer to Table B.2 for a description of the roles.

coalition. We refer to this problem as a fixed-hierarchy. To do so, we introduce below: (i) the instantiation of the two components of the valuation functions (i.e., the relationship and the role disturbance); and (ii) the constraints placed on the interaction graphs. Then, we move on to the experiments.

6.5.1 Components of the Valuation Functions

In this problem, ordinary agents are supposed to be subordinated to a superior. Therefore, we evaluate the relationship between subordinate agents and the superiors of the corresponding level. To do so, we use Equation 6.3, where C is a coalition, and S_i is the set of superiors of the corresponding level i.

$$relationship_i(C) = \frac{1}{|C \setminus S_i|} \times \sum_{a \in C \setminus S_i} relationship(a, C \cap S_i) \tag{6.3}$$

In case C is a singleton coalition, then the relationship is set to 0.

Regarding the roles, we do not assume a precise number of roles required by each superior at the last level of the hierarchy (nonetheless, for the upper hierarchical levels this can be computed). To figure out which roles are required in a coalition C, we look at the pivotal agents of the TF and ST level that are in C. Let $require(C)$ be a ***multiset*** that counts the role demands (which role and which quantity) required by a coalition C. That is, $require(C) = \{r \in expect(a) \mid a \in C \cap S_1\}$. Note that the quantity per role in $expect(a)$ is always a single unity

because in the TF and ST level, we do not assume a predefined requirement on the quantity of roles. Let $m_{require(C)}(.)$ count the multiplicity of an element in a multiset $require(C)$.

Example 12. Assume a leader $l_1 \in S_1$ and a coalition $C : C \cap S_1 = \emptyset$ formed for the third game Γ_3 (i.e., the branch level). Moreover, let $expect(l_1) = \{r_1, r_2\}$. Thus, $require(C \cup \{l_1\}) = \{r_1 : 1, r_2 : 1\}$. Now, assume another leader $l_2 \in S_1$ such that $expect(l_2) = \{r_2\}$. Thus, $require(C \cup \{l_1\} \cup \{l_2\}) = \{r_1 : 1, r_2 : 2\}$.

Similarly, let $available(C)$ be a multiset of roles in C adopted by its ordinary agents. That is, $available(C) = \{r \in adopt(a) \mid a \in C \setminus L\}$.

Now we are in position to calculate the role disturbance within a coalition C. To do so, we calculate the relative entropy (Kullback-Leible distance) (Dragomir et al., 2000) within a coalition according to the roles adopted by its agents. Let $P(r)$ be the proportion of agents in C that adopt role r computed using Equation 6.4.

$$P(r) = \frac{m_{available(C)}(r)}{\sum_{r' \in available(C)}(m_{available(C)}(r'))} \tag{6.4}$$

Similarly, let $Q(r)$ be the proportion of agents that must adopt role r computed using Equation 6.5.

$$Q(r) = \frac{m_{require(C)}(r)}{\sum_{r' \in require(C)}(m_{require(C)}(r'))} \tag{6.5}$$

Then, we can compute the relative entropy using Equation 6.6.

$$D(P \parallel Q) = \sum_{r \in require(C)} P(r) \ln \frac{P(r)}{Q(r)} \tag{6.6}$$

To compute the role disturbance within a coalition in the fixed-hierarchy problem we use Equation 6.7.

$$disturbance_i(C) = \frac{D(P \parallel Q)}{\ln |require(C)|} \tag{6.7}$$

Consider the example below.

Example 13. Assume a coalition C such that $require(C) = \{r_1 : 1, r_2 : 2\}$ and $available(C) = \{r_1 : 1, r_2 : 2\}$. Then, $disturbance(C) = 0$. In case, $available(C) = \{r_2 : 2\}$, then $disturbance(C) \approx 0.58$. If the quantity of available roles move away from the baseline requirement, then the disturbance increases as well. For instance, if $available(C) = \{r_1 : 3, r_2 : 2\}$, then $disturbance(C) \approx 0.21$. If $available(C) = \{r_1 : 4, r_2 : 2\}$, then $disturbance(C) \approx 0.33$.

However, it is not always the case that the disturbance function above can/should compute a valid result. We consider some special cases:

- if $available(C) = \emptyset$, then $disturbance(C) = 1$;
- if $require(C) = \emptyset$, then $disturbance(C) = 1$; and

- if $|require(C)| = 1$, then $disturbance(C) = 0$ (only resources of the same role required).

6.5.2 Constraints on the Interaction Graph

The set of pivotal agents and permitted sizes are already defined for the problem (see Section 6.2). All that remains to be done is for us to model the constraints on the interaction graphs. To do so, given an interaction graph $G_i : 1 \leq i \leq h$, an edge $(u, r) \in E_i$ connects a superior $u \in S_i$ to any resource if either:

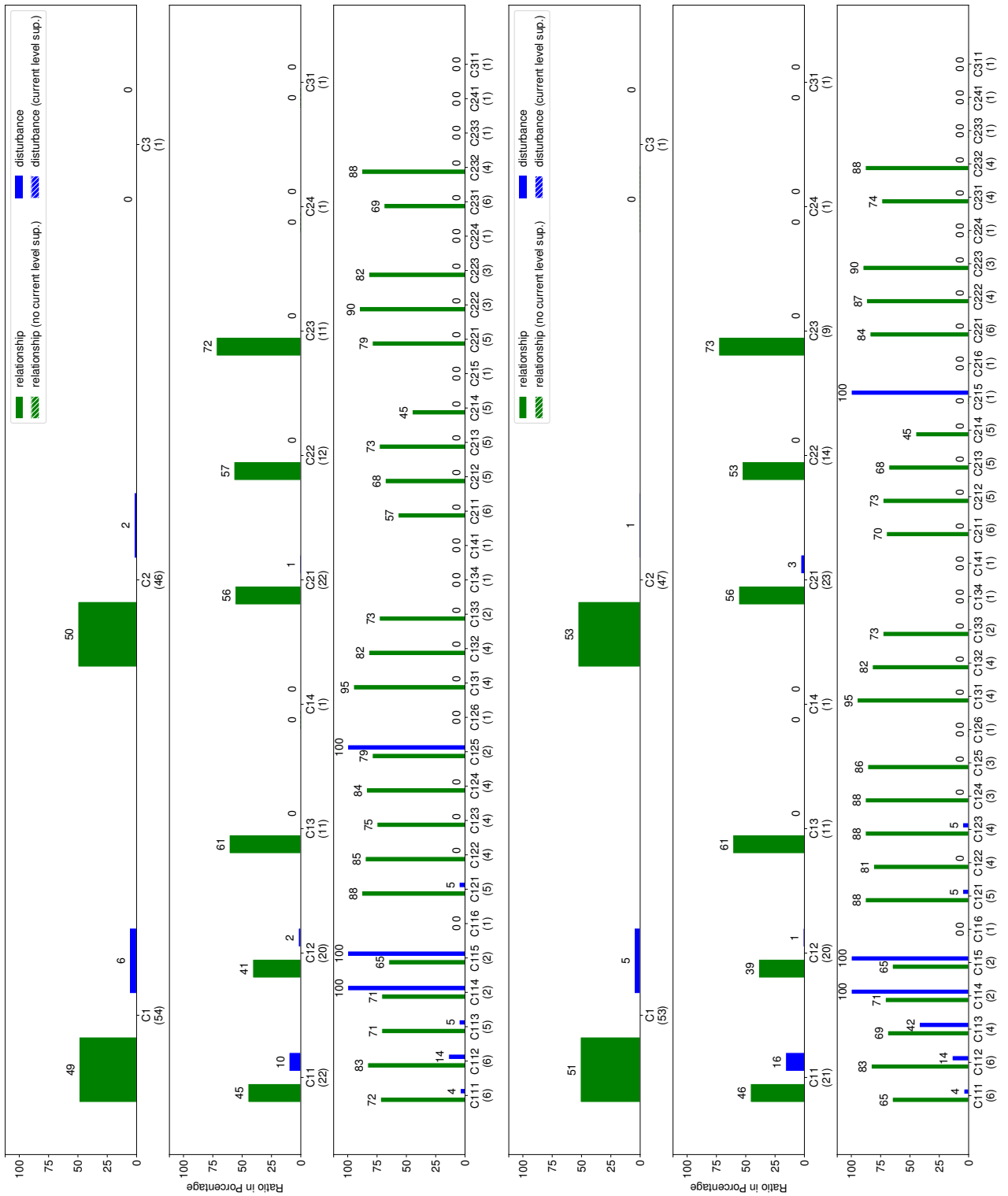
- $r \in L \setminus S_i$ and $resp(r) \subseteq resp(u)$; or
- $|adopt(r) \cap expect(u)| \geq 1$.

6.5.3 Experiments

Given the instance above, we run UCT-Seq using the following parameters: (i) $\bar{b} = 5$; (ii) $\bar{d} = 3$; (ii) $\gamma = 0.7$. These are determined empirically. We set a timeout of 30 minutes and record the best hierarchy computed within the time budget. We conducted all experiments in a machine with 32 GB of RAM and a CPU with four single cores of 3400 MHz each. In our first experiment, we aim to form a hierarchy of 101 resources in which each resource adopts a single role. We use the relation $\mathcal{R}^{\text{BASIS}}$ (Definition 27) to determine the feasibility in an FCSS. We depict the resulting hierarchy (without the OPS level) in a bar chart representation in Figure 6.5a. Note that we depict the hierarchy in its natural representation, that is, first the branch level, then the group level and finally, the TF and ST level.

One can link the coalitions of different levels by following the coalition Ids. For instance, in Figure 6.5a, at the uppermost level there exists coalitions C1, C2, and C3. The coalitions subordinated to C1 are the coalitions C11, C12, C13 and C14 in the second level. Similarly, the coalitions subordinated to C11 are C111 through C116. Underneath each coalition Id, we depict the corresponding coalition size. In case a singleton coalition corresponds to a superior of an upper level, we set both relationship and disturbance metrics to 0 (e.g., C14).

The resulting hierarchy in Figure 6.5a let no ordinary resource or superior isolated; all coalitions have a corresponding superior and a subset of subordinate resources. Our primary goal was achieved and a fixed hierarchy is delivered by UCT-Seq. Moreover, one can notice greater relationship values in coalitions located at bottom levels. In the branch level, the average relationship is $\approx 50\%$. In the subsequent level it increases to $\approx 56\%$, and in the third level it achieves $\approx 76\%$. Regarding the roles, three coalitions depict a great disturbance value at the bottom level (C114, C115, and C125). All three coalitions are task forces each requiring 3 roles in which only 1 role is available and 2 roles are missing. The missing roles are AETRT (required by all three leaders), ASRTEC, and VMT. In the role distribution in Figure 6.3, one can note



(a) Personnel resources may adopt up to 1 role. (b) Personnel resources may adopt up to 3 roles.

Figure 6.5: A hierarchy of 101 resources computed by UCT-Seq for an RRF instance. The first chart (in both figures) represents the branch level of the hierarchy and the third chart the TF and ST level.

that those roles were in higher demand than availability. Therefore, we seek to minimise the loss regarding the final allocation of those resources.

In our next experiment, we investigate a setting in which a *personnel* resource may adopt more than one role (up to three roles). We depict the resulting hierarchy in Figure 6.5b.

One can see in the chart that an ordinary resource is left apart in the resulting hierarchy (C215), even though this resource contributes to the two upper levels. This resource adopts *only* role ABS. Looking again at the role distribution in this instance in Figure6.4, one notices that more resources can adopt this role than it is demanded. Therefore, it is reasonable that an agent adopting this role cannot form a coalition with the corresponding leader of the strike team. Moreover, we see once more two coalitions with a great disturbance value (C114, C115). Those coalitions require additionally roles AETRT and VMT, which are roles adopted by resources of kind *team* and therefore, none of those resources is allowed to adopt more than a single role. When it comes to relationship, at the uppermost level the average value is $\approx 52\%$, followed by $\approx 55\%$ in the second, and $\approx 78\%$ in the third. That means, the overall relationship was increased when compared with the instance in which any resource must adopt a single role.

Next, we investigate the hierarchy formed by 141 resources when each agent may adopt only a single role. Recall that for the RRF problem, in this instance, there are enough ordinary resources (i.e., resources who are not superiors) so that each leader has exactly λ-subordinate resources. We depict the resulting hierarchy in Figure 6.6.

In this setting no ordinary resource is left apart and one can see that all coalitions in the last hierarchical are full. That is, each leader has at its disposal λ subordinate resources. The relationship within the coalitions follow the same pattern as we saw in the former experiments: greater values at the last hierarchical level. Precisely, in the branch level it achieves 49%, in the group level $\approx 50\%$, and 78% in the bottom level. One note a decrease in the relationship value in the first two levels (when compared to the former experiments), but it matches the value for the third level. We can also note that the role disturbance plays a role only in the task forces. This is due to fact that a TF requires 3 different roles but only 5 ordinary resources are allowed in each coalition.

6.6 A Hybrid Hierarchy of Resources

In the previous section, the resources had no choice other than to form coalitions with predefined superiors or stay in singleton coalitions. In a variation of the RRF problem, we relax the fact that all superiors are given as input; some may be given and others will be decided depending on the result of the computed FCSS, hence the name *hybrid*. Apart from the span of control and sets of pivotal agents, the algorithm also determines new coalitions in the hierarchy based on the characteristic functions. To model this particular problem, we need to make sure that the span of control continues to be followed. Moreover, we slightly modify the valuation functions (more precisely, their two components) and the constraints placed on the interaction graphs. We refer to this particular variation of the problem as a hybrid hierarchy.

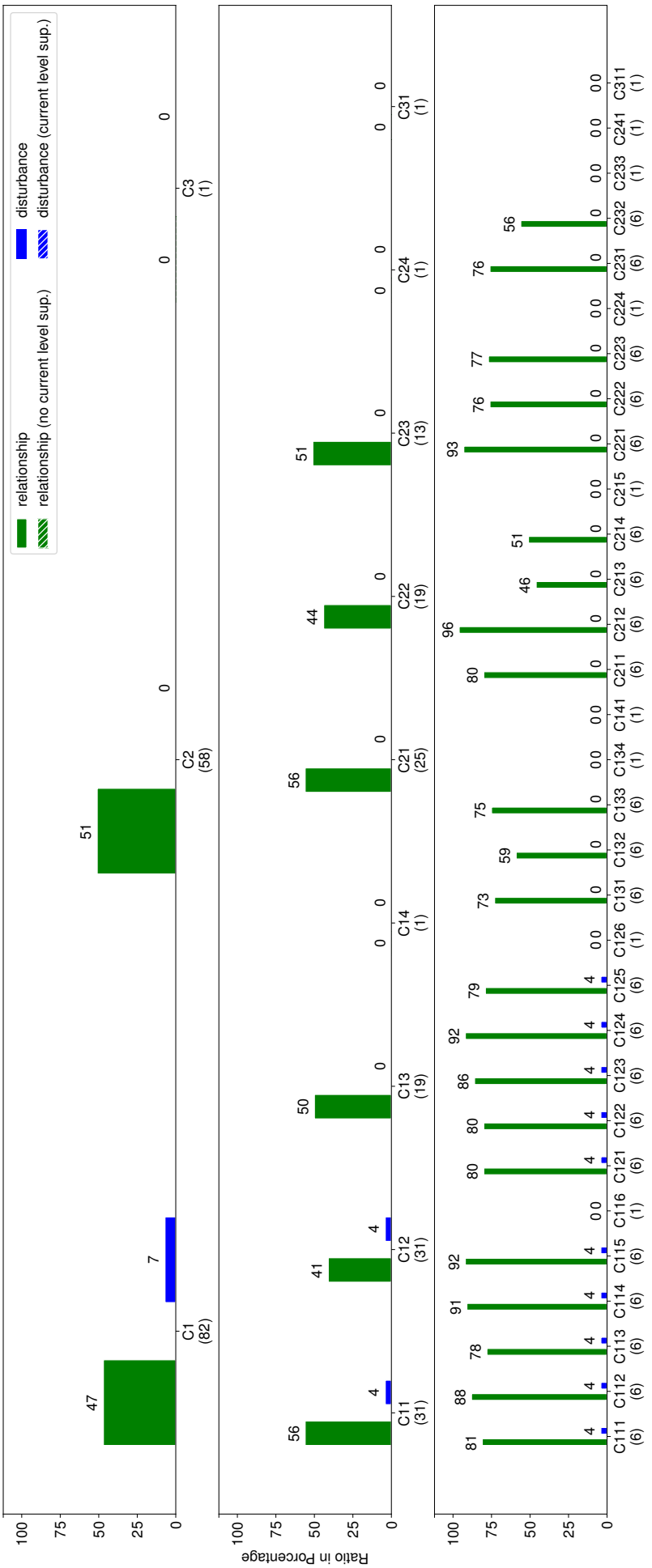


Figure 6.6: A fixed hierarchy of 141 agents computed by UCT-Seq for a RRF instance.

6.6.1 Components of the Valuation Functions

Given the fact that ordinary resources do not need to be in the same coalition as the pivotal agents, we modify the relationship component to take into account any two agents in a coalition. This is given in Equation 6.8.

$$relationship_i(C) = \frac{1}{\binom{|C|}{2}} \times \sum_{a,a' \in C : a \neq a'} relationship(a, a') \tag{6.8}$$

In case C is a singleton coalition, then the relationship value is set to 0.

Regarding the role disturbance, we just need to adjust how we compute the proportion of roles to ignore roles required by superiors. Recall that $available(C)$ is a multiset counting the quantity of roles adopted by the ordinary agents in a coalition C. This time we use an entropy value (Dragomir et al., 2000) computed using Equation 6.9. Recall that $P(r)$ is the proportion of agents in coalition C adopting role r.

$$H(P) = -\sum_{r \in available(C)} P(r) \ln P(r) \tag{6.9}$$

Then we compute the role disturbance using Equation 6.10.

$$disturbance_i(C) = \frac{H(P)}{\ln |available(C)|} \tag{6.10}$$

We also consider some special cases:

- if $available(C) = \emptyset$, then $disturbance(C) = 1$; and
- if $|available(C)| = 1$, then $disturbance(C) = 0$ (only resources of the same role in C).

6.6.2 Constraints on the Interaction Graph

To allow the resources to form coalitions that do not contain superiors, we add more edges in the interaction graphs. Let us introduce an auxiliary notation. Let R^a be a set of roles that are compatible with the roles that resource a may adopt. That is, $R^a = \{r \in demand(o) \mid o \in IO, adopt(a) \cap demand(o) \neq \emptyset\}$. Then, given an interaction graph $G_i : 1 \leq i \leq h$, an edge $(u, r) \in E_i$ such that $r \notin L$ connects any two resources if either:

- $u \in L$ and $adopt(r) \cap expect(u) \neq \emptyset$; or
- $u \notin L$ and $adopt(r) \cap R^u \neq \emptyset$.

6.6.3 Connecting the Outcomes of the Games

We slightly modify the binary relation $\mathcal{R}^{\text{BASIS}}$ (Definition 27) to achieve a hierarchy with no mandatory superiors.

Definition 29 ($\mathcal{R}^{\text{HYBRID}}$). Given $CS, CS' \in \boldsymbol{\mathcal{CS}}^A$, the pair $(CS, CS') \in \mathcal{R}^{\text{HYBRID}}$ iff:

1. $|CS| > |CS'|$;
2. for all $C' \in CS'$ there exist at most $\lambda + 1$ coalitions $C \in CS$ such that $\bigcup_j C = C' : 1 \leq j \leq \lambda + 1$;
3. if $j = \lambda + 1$, then $\exists \widehat{C} \in CS : \widehat{C} \subseteq C', |\widehat{C}| = 1, \widehat{C} \cap L \neq \emptyset$.

Moreover, $(\emptyset, CS) \in \mathcal{R}^{\text{HYBRID}}$ for all $CS \in \boldsymbol{\mathcal{CS}}^A$.

In the definition above, we introduced Rule 3 to make sure a superior is assigned to a set of coalitions whenever we reach the span of control plus one. If that is not the case, the algorithm forms at most λ coalitions and no superior *must* be assigned to them. As $\mathcal{R}^{\text{HYBRID}}$ is enforced on the whole sequence of coalition structures, we cannot make Rule 3 to hold only for the superiors of the subsequent level in the hierarchy. To address this, one would have to model it using a binary relation on each transition between games as discussed in Section 3.4.

6.6.4 Experiments

We experiment with the instances for 141 resources introduced in Section 6.4.2. However, this time considering a hybrid hierarchy. We conducted all experiments in a machine with 32 GB of RAM and a CPU with four single cores of 3400 MHz each. We shall use the same bar chart representation to show the resulting hierarchy as described in Section 6.5. Recall that we depict the hierarchy in its natural representation, that is, first the branch level, then the group level and finally, the TF and ST level. We depict the results in Figure 6.7.

When each personnel resource adopts a single role, two new coalitions are formed at the uppermost level (see Figure 6.7a). They are coalitions C3 and C4. Quite interestingly, those coalitions contains 4 out of the 22 leaders of the last level, and only one supervisor of the intermediate level (out of six supervisors). In fact, the new coalitions remain the same in both branch and group levels. That is, they do not split themselves from the first level to the next one. In total, 13 new coalitions are formed: 2, 3, and 8, in the uppermost level to the bottom level respectively. The average relationship value in the first two levels achieved: $\approx 54\%$ in the branch level and $\approx 56\%$ in the group level. In the last hierarchical level that average dropped to $\approx 75\%$. Thus one can see an increase in upper levels' relationship at the expense of the last level's relationship value.

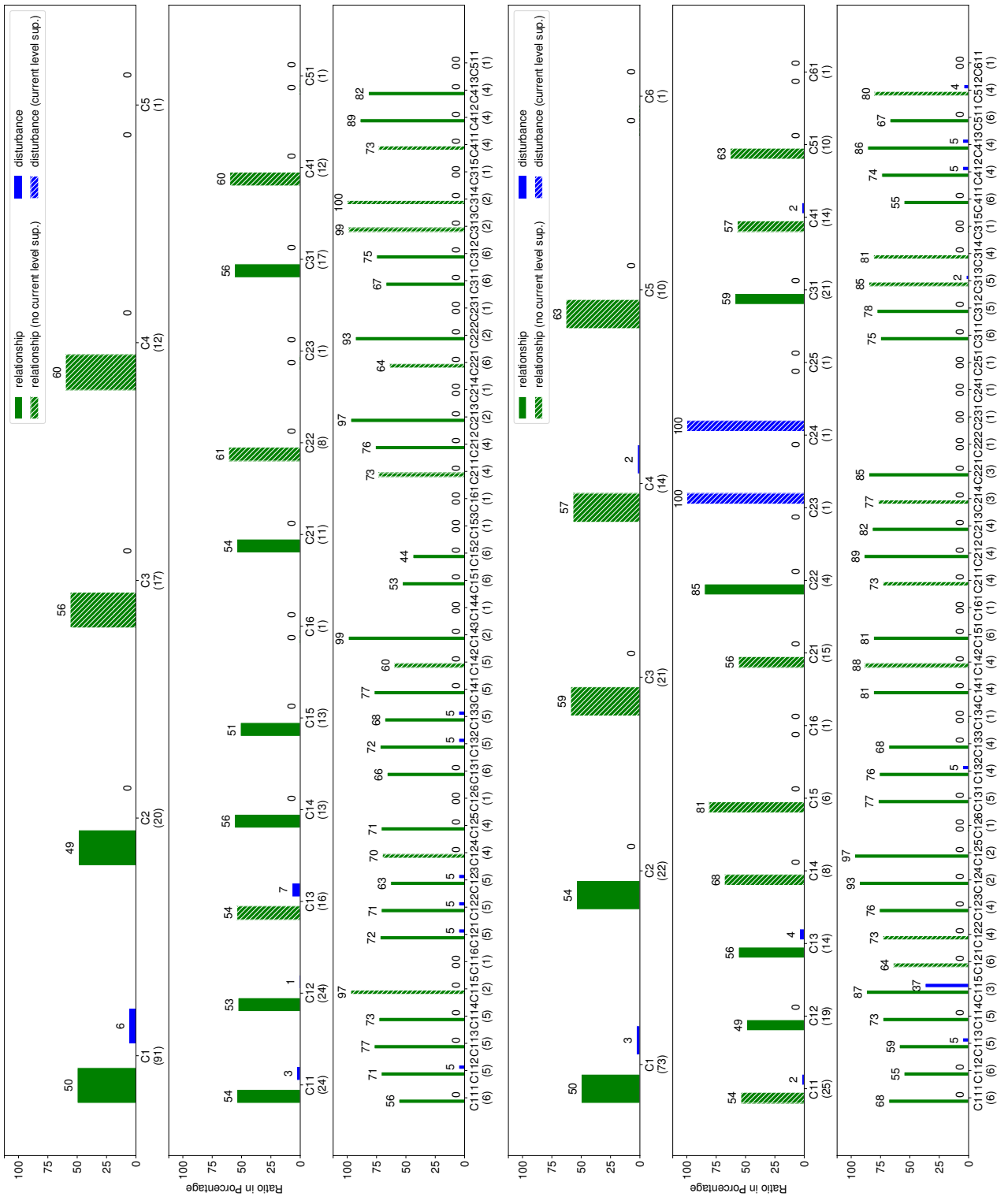
The main question is then which goals the new coalitions are prepared to achieve. A further look reveals that in the branch level the two new coalitions can achieve 2 out of the 7 IOs. Interestingly, coalition C1 has resources who adopt roles that can achieve those two IOs as well. However, at the bottom level, the average number of IOs each coalition can achieve is $\approx 1,2$ whilst in a fixed hierarchy it approaches $\approx 1,23$. That means most coalitions can work on a single goal, and whether a hierarchy is hybrid or fixed does not impact this index. Instead, we expect it to be influenced by the number of roles an agent may adopt, which we experiment with next.

Our last experiment increases the number of roles a personnel resource may adopt to three. We consider again 141 resources that must form a hierarchy for the RRF problem. We depict the resulting hierarchy in Figure 6.7b. This time, 17 coalitions with no superior were formed. Nonetheless, 9 coalitions out of the 17 belong to the two upper hierarchical levels (3 and 6 coalitions respectively), and only 8 to the last one. Interestingly, two superiors of the intermediate level are left apart in the hierarchy. Moreover, the average relationship metric outperforms the values obtained in the former experiment in all three levels: $\approx 57\%$, $\approx 63\%$ and $\approx 78\%$ in the first, second, and third level respectively. When it comes to the average of IOs that the coalitions can achieve at the bottom level, it rises up to $\approx 2,1$. This is expected as the resources can adopt more roles and hence, in general, the superiors have more flexibility to assign an IO to a given coalition.

6.7 Discussion

In the two main experiments reported in this chapter, we showed how our approach can be used to model two variations of the RRF problem. The first one considering a fixed hierarchy in which all superiors are determined by the OPS chief. The resources have no choice but to comply with it. In the other variation of the problem, we relax that constraint and allow ordinary resources to form coalitions on their own. That means, for instance, the resources could elect a superior among themselves afterwards. In both cases UCT-Seq computed a hierarchy for the problem at hand, which, in our viewpoint, makes our overall goal achieved. Moreover, the fact that we are forming coalitions to achieve incident objectives seems to fit well the underlying core concept in coalition formation. One is interested in grouping agents that want to bring about some given goal. No concrete task is assigned at this point of the response operation. For instance, a task in which a resource should dispose of 5 animal corpses at coordinate (x, y). Doing so, we deliver groups of agents for the first stage of cooperation life-cycle (described in Section 2.1.1).

We showed how to model task forces and strike teams in our framework. Although not required in the RRF problem, we believe single resources can also be modelled easily in our proposed formalism. Regarding modelling divisions, one approach could be to connect agents over the interaction graph that are in the same jurisdiction; that is, edges connection



(a) Personnel resources may adopt up to 1 role. (b) Personnel resources may adopt up to 3 roles.

Figure 6.7: A hybrid hierarchy of 141 resources computed by UCT-Seq for an RRF instance. The first chart (in both figures) represents the up-most level of the hierarchy and the third chart the bottom level.

agents of different jurisdictions are removed. Doing so, one can compute all main elements in the Operations Section. Our modelling might be extended to take into account the entire ICS hierarchy as well. However, the problem then turns out to be how to design valuation

functions that are general enough to describe well coalitions aimed for quite different purposes. For instance, coalitions of the Planning and Finance Sections (FEMA, 2017). An important characteristic of MAS and yet to be addressed in the ICS is autonomous software entities. In the light of the advances in the disaster robotics area (Murphy, 2014), in which an agent might control a robot, many challenges arise, e.g., how to design a valuation function that brings together resources of kind personnel and team as well as those autonomous entities. However, note that those are challenges in any coalition formation framework, not restricted only to SCFG.

When it comes to modelling the ICS problem using SEQSVS (or SEQVS), the interaction graphs are the main tools to model different sorts of constraints. For instance, in our modelling we used them to link the roles adopted by the resources to the ones required by the superiors. Although those constraints can be very narrow (i.e., a few edges in an interaction graph), for the problems introduced in this chapter many FCSSs can be formed. We depict in Figure 6.8 the quality improvement of the solutions computed by UCT-Seq over time (in all experiments). We mark in the chart the precise moment an FCSS of greater value was found. Moreover, for each FCSS computed by UCT-Seq throughout the execution, we mark with a dot the precise point in time it was found and the current value of the best FCSS found so far; that explains the discontinuous lines in Figure 6.8. Recall that the time budget in our experiments was 30 minutes for instances requiring a fixed hierarchy and 1 hour for the hybrid ones.

One can see that the number of improvements on solutions found by UCT-Seq is greater for hybrid hierarchies, regardless of the time budget. In fact, in those instances, from 30 minutes on, only 5 updates on FCSSs were carried out. That suggests the heuristic applied by UCT-Seq works well for the problems as we can see that many more FCSSs were computed. In fact, for a fixed hierarchy of 101 resources adopting a single role 10.502 FCSSs were computed. When each personnel resource adopts at most three roles, that number increases to 11.343 FCSSs. When it comes to 141 resources adopting a single role, 5.151 FCSSs were found after 30 minutes.

On the other hand, for hybrid hierarchies, after approximately 10 minutes UCT-Seq starts increasing the interval between finding better solutions. Interestingly, to compute a hybrid hierarchy of 141 resources adopting a single role, UCT-Seq went through 5.434 FCSSs in twice the time than for a fixed hierarchy of the same size. When we increase the number of possible roles an agent can adopt, the quantity of FCSSs found increases to 12.865. Note that in both cases the algorithm found FCSSs until the time budget ran out. This suggests it is much more difficult to find FCSSs in the hybrid-1 scenario than in the hybrid-3.

As a matter of fact, one should bear in mind that the ICS application modelled here lacks feedback from experts on disaster response operations. In particular, incident commanders and section chiefs. They might modify (narrow/relax) the constraints as they see fit as well as the valuation functions. Our main motivation in this chapter was to show that modelling a real-world problem is feasible using SCFG, and in particular using SEQSVS. Further effort is

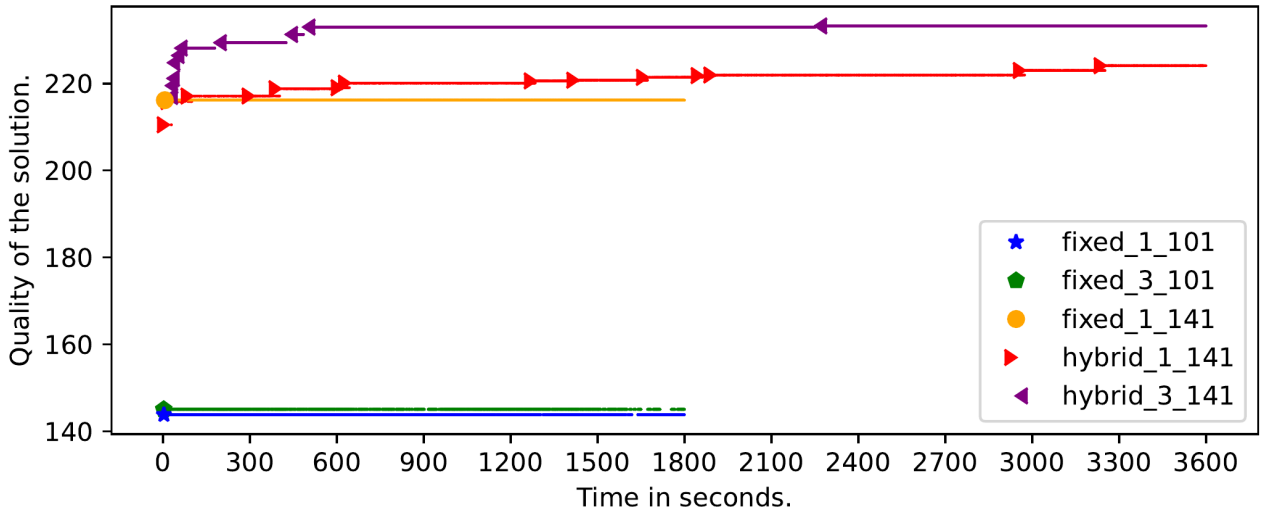


Figure 6.8: Comparison of the running time used by UCT-Seq to progressively improve on the solution quality. The first component of an approach indicates if the resources must form coalitions only with superiors (fixed description). The second component indicates how many roles a personnel resource is allowed to adopt and the third the number of resources available.

required to validate the proposed modelling of the ICS with experts on the topic, although we followed the ICS, a well-known approach in that area, exactly as reported in the literature.

Chapter Remarks

In this chapter, we introduced the ICS framework and discussed in detail the Operations Section, which is responsible for the tactical operations in the environment affected by a disaster incident. To demonstrate how one can use one of our proposed frameworks and algorithms to solve a real-world problem, we modelled the roaring river flooding scenario introduced by an agency in the US as part of an ICS training course. In particular, we modelled two cases: (i) when resources are required to form coalitions with predefined superiors; and (ii) cases in which there are designated superiors for the response operation, but the resources are allowed to form coalitions on their own provided it increases the overall social welfare. We computed a three-level hierarchy for these problems and showed how one can define metrics to evaluate the resulting chain of command in terms of the relationship among resources and the role disturbance (i.e., missing or more than necessary roles in a given coalition). Although the application introduced in this chapter was not reviewed by an expert on the topic, we are confident that the proposed modelling can be used to address the requirements of a disaster response operation. This concludes Contribution 8. Moreover, we make available all code generated as a product of this research at: https://github.com/smart-pucrs/SCFG. This concludes Contribution 9.

7. CONCLUSIONS

In this thesis, we investigated the interdependence of the solutions computed by different Characteristic-Function Games (CFG) placed in a sequence of games. In particular, we noted that it is not always the case that individual optimal outcomes (one per game) are compatible with one another. Moreover, no game is to be solved in isolation, which means the solution produced by a CFG might lead to an unfeasible overall solution. To the best of our knowledge, no formal framework has been proposed to address this particular problem in the coalition formation literature. Given this unexplored venue, we proposed a new framework called Sequential Characteristic-Function Games (SCFG) in which one aims to compute a sequence of coalition structures. When every CS in that sequence is compatible with its respective subsequent one (modelled using a binary relation), we call the sequence a Feasible Coalition Structure Sequence (FCSS). This forms the solution concept for our game.

We extended that new sort of game to take into account constraints that come from individual games in the sequence. That means, a particular coalition structure might appear only at a particular position in an FCSS. As a result of this approach, we proposed two new frameworks. The first assumes a sequence of Valuation Structures (VS) that are induced over the corresponding CFGs in an SCFG (called SEQVS). A VS allows us to express constraints in the form of a set of agents in the system that must stay in different coalitions. In addition, one can further express constraints using an interaction graph over the agents in the system. To make it possible to model constraints on the size of the coalitions as well, we extend VS. The resulting framework is called Sized Valuation Structures (SVS), and when combined with SCFG is called SEQSVS.

Our overall goal was to model a real-world application, in particular, the challenging Incident Command System (ICS). It establishes guidelines on the best practices one should follow in order to respond to a disaster incident. It has been put in practice first in the US and then ended up being used by many agencies in different countries. The problem of interest for us is how to form a hierarchy of resources taking into account the designated superiors throughout the chain of command (i.e., a hierarchy) as well as the span of control which defines a manageable ratio between superiors and subordinate units. Our approach is to interpret each level of the hierarchy as a different CFG. The overall hierarchy is then modelled by an SCFG. In particular, we used the SEQSVS framework to model the specifics of each level.

7.1 Summary of Results and Discussion

We investigated the problem above under both theoretical and practical perspectives. We showed that the corresponding SCFG decision problem lays in the **PSPACE**-complete class (Theorem 2). This is already the case when one considers an algorithm that generates precisely

the coalition structures that may follow each other in the sequence of games. From an empirical viewpoint, we proposed an exact algorithm based on dynamic programming (called SDP) to solve SCFG based instances. We showed that even for a reasonable amount of time (1 hour in the experiments reported here), it can solve only instances containing a small number of agents (up to 10 agents for large binary relations). However, if compared with a brute-force algorithm, it is faster by several orders of magnitude. One should bear in mind that SDP requires a supporting algorithm to generate the CSs that follow each other in the sequence. To obtain the best performance from SDP, such an algorithm should be effective; that is, it should generate only coalition structures compatible with a given one. Those are not trivial procedures as we are enumerating the coalition structures that fit together in the binary relation of the problem at hand. Recall that solving the coalition structure generation problem is shown to be $\boldsymbol{F}\boldsymbol{\Delta}_2^P$-complete (Greco and Guzzo, 2017).

The question is then to show how to avoid the generation of pairs of coalition structures. We proposed MC-Link which uses a hierarchical clustering method to compute an FCSS starting with a sequence of CSs of singleton coalitions. It assumes as input a procedure to check whether or not subsequent CSs in the sequence are allowed. To determine the next CS to check, it employs a simple greedy procedure based on the gain of merging any two coalitions (Equation 5.1). This strategy indeed computes a solution in shorter periods of time if compared to all algorithms introduced in this thesis. However, the side effect is that MC-Link does not handle efficiently constraints on the size of coalitions and coalition structures. The above characteristic makes MC-Link incomplete for many problems.

In the absence of a general procedure to compute solutions for SCFG, we turned our attention once again to algorithms that use a procedure to generate compatible CSs. We proposed an algorithm based on Monte Carlo Tree Search (MCTS) named UCT-Seq. This algorithm is complete (Theorem 5) and is guaranteed to produce an optimal outcome given a sufficient amount of time. Two main challenges arise in this approach: (i) the generation of pairs of CSs; and (ii) the roll-out of a path to reach a terminal state. To address the first problem, we exploited the constraints imposed by both VS and SVS. This means we generate a compatible CS based on the constraints on the next game and filter out incompatible CSs based on the binary relation of the game. However, even in this approach, the generative algorithm is not efficient; it produces many more CSs than required.

To address the second problem, we decided to perform a series of split and mergers operations in a random manner. We observed that, in general, constraints might not be related to the values assigned to coalitions. That is, incompatible coalition structures might be good solutions in terms of valuation in subsequent games. Therefore, there exists a trade-off in employing an informed search during a roll-out for this problem. In the experiments reported here, with up to 50 agents, we noted that MC-Link is faster than UCT-Seq (given a general generation and roll-out procedure). However, given a reasonable additional amount of time, UCT-Seq computes solutions of better quality than MC-Link. The combination of the two problems above

represents the main challenge to come up with a good algorithm to tackle any SCFG related problem using an MCTS approach.

Aware of the limitations in designing algorithms to compute FCSSs, we investigated how to apply the mechanism developed so far to a real-world problem. In particular, we considered the Roaring River Flooding (RRF) scenario used in an ICS training course (U.S. Department of Agriculture, 2021). We showed that SEQSVS provides an elegant way of modelling the problem of forming a hierarchy of resources to respond to that synthetic disaster incident. Moreover, we slightly modify our modelling to allow a hybrid hierarchy of resources. That is, not all superiors of the hierarchy are given, and then we allow coalitions with no pivotal agent of the corresponding level to form. We then used UCT-Seq to compute solutions for problem instances containing 101 and 141 resources given a modification in its generation of CSs process. The resulting hierarchy delivered a way to analyse and interpret possible vulnerabilities in the organisation of the resources throughout the chain of command.

7.2 Future Work

Our new game is clearly challenging and opens up various research directions. For instance, consider the payoff distribution problem, another important aspect of the coalition formation process. In case the solutions of many CFGs are interdependent, the question is then if we can reach a stable distribution of the payoff given the current solution concepts available in the literature (e.g., the Shapley value (Shoham and Leyton-Brown, 2009)). For instance, an agent might willingly agree to reduce its reward in a set of particular games in the sequence provided an increase in its reward in a particular subsequent game. This direction will also require further complexity analysis as well.

Another natural extension to SCFG regards the compatibility of a CS throughout the sequence rather than in pairs. The SCFG framework models whether two subsequent CSs are compatible with each other. However, in some applications (refer to the discussion in Section 3.2) the complete subsequence that precedes a CS might be relevant to determine its compatibility, not only the preceding CS. The precise relation between SCFG and this sort of game is to be determined, in particular how it affects complexity results presented in this work.

Further efforts should be made to identify tractable classes of SCFG. As we discussed throughout this thesis, evaluating the interdependence of the outcome produced by different games is challenging. Our theoretical analysis combined with our modelling of real-world situations confirms this hypothesis. While modelling the RRF application, we noted that a small change in the process of generating compatible coalition structures allowed us to scale up the experiment to more than 101 agents. That change is relation-dependent; it applies to all problems that require all coalitions to be a superset of a coalition in a subsequent CS. This raises the question of how to model constraints that impact on other games. Certainly, one can achieve this by listing out the CSs of interest in $\mathcal{R}$. However, this is not desirable for real-world ap-

plications, unless the corresponding relation $\mathcal{R}$ can be compactly represented and computed. Therefore, future work should investigate heuristics and techniques that can be employed by algorithms to generate pairs of coalition structures belonging to the SCFG relations on-the-fly.

We also mention the improvement of the algorithms proposed here as a possible direction for future work. For instance, regarding MC-Link, we would like to compute solutions that are within a bound of an optimal solution. When it comes to UCT-Seq, faster and more efficient methods to compute the generation of CSs as well as the roll-outs are desirable.

Another venue for future work is related to the ICS. We were not able to validate the proposed modelling with experts on the topic. This is important to build a robust mechanism. It is desirable to collect more features that are of interest to those responsible for minimising the losses in disaster incidents. Also, we aim to gather realistic data regarding disaster response operations to feed into our proposed SEQSVS model. Unfortunately, for the time being, this sort of information is not publicly available, even for research purposes. Nevertheless, we believe the produced material is mature enough to be brought into discussion in the disaster response community, for instance, the information systems for crisis response and management community[1].

As a final remark, it is also important to model different domains using SCFG. In this work we focused mainly on the disaster response domain. However, we expect this type of game can be used to model other challenging applications. A promising application is training sessions as discussed in Section 3.2. Instead of considering a single project, one might be interested in the overall outcome throughout a sequence of projects assessing different abilities of the participants. Again, the main challenge regards gathering the data for the experiments. For instance, we would need data about projects carried out by companies, specially large corporations.

[1] https://iscram.org/

REFERENCES

Akasiadis, C. and Chalkiadakis, G. (Sep., 2017). Cooperative electricity consumption shifting. *Sustainable Energy, Grids and Networks*, vol. 9, pp. 38–58.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (May, 2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, vol. 47, pp. 235–256.

Ayari, E., Hadouaj, S., and Ghedira, K. (2017). A dynamic decentralised coalition formation approach for task allocation under tasks priority constraints. In: *Proceedings of the 18th International Conference on Advanced Robotics*, pp. 250–255. IEEE.

Basegio, T. L. and Bordini, R. H. (Nov., 2018). Allocating structured tasks in heterogeneous agent teams. *Computational Intelligence*, vol. 35, pp. 124–155.

Ben Jouida, S., Krichen, S., and Klibi, W. (Jan., 2017). Coalition-formation problem for sourcing contract design in supply networks. *European Journal of Operational Research*, vol. 257, pp. 539–558.

Bistaffa, F., Farinelli, A., Cerquides, J., Rodríguez-Aguilar, J., and Ramchurn, S. D. (2014). Anytime coalition structure generation on synergy graphs. In: *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems*, pp. 13–20. IFAAMS.

Bistaffa, F., Farinelli, A., Cerquides, J., Rodríguez-Aguilar, J., and Ramchurn, S. D. (Feb., 2017a). Algorithms for graph-constrained coalition formation in the real world. *ACM Transactions on Intelligent Systems and Technology*, vol. 8, pp. 1–24.

Bistaffa, F., Farinelli, A., Chalkiadakis, G., and Ramchurn, S. D. (May, 2017b). A cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, vol. 246, pp. 86–117.

Björklund, A., Husfeldt, T., and Koivisto, M. (Jul., 2009). Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, vol. 39, pp. 546–563.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (Mar., 2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, pp. 1–43.

Chalkiadakis, G., Elkind, E., Markakis, E., and Jennings, N. R. (2008). Overlapping coalition formation. In: *Proceedings of the 4th International Workshop on Internet and Network Economics*, pp. 307–321. Springer.

Chalkiadakis, G., Elkind, E., Markakis, E., Polukarov, M., and Jennings, N. R. (Sep., 2010). Cooperative games with overlapping coalitions. *Journal of Artificial Intelligence Research*, vol. 39, pp. 179–216.

Chalkiadakis, G., Elkind, E., and Wooldridge, M. J. (2011). *Computational Aspects of Cooperative Game Theory.* Morgan & Claypool Publishers.

Childs, B. E., Brodeur, J. H., and Kocsis, L. (2008). Transpositions and move groups in monte carlo tree search. In: *Proceedings of the 3rd Symposium on Computational Intelligence and Games*, pp. 389–395. IEEE.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms.* The MIT Press.

Dang, V. D. and Jennings, N. R. (2004). Generating coalition structures with finite bound from the optimal guarantees. In: *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 564–571. IEEE.

Dann, M., Thangarajah, J., Yao, Y., and Logan, B. (2020). Intention-aware multiagent scheduling. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pp. 285–293. IFAAMS.

Dragomir, S., Scholz, M., and Sunde, J. (May, 2000). Some upper bounds for relative entropy and applications. *Computers & Mathematics with Applications*, vol. 39, pp. 91–100.

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

Feldman, M., Gravin, N., and Lucier, B. (2015). Combinatorial auctions via posted prices. In: *Proceedings of the 26th Symposium on Discrete Algorithms*, pp. 123–135. SIAM.

FEMA (2015). *National Preparedness Goal.* Independently Published.

FEMA (2017). *National Incident Management System.* Independently Published.

FEMA (2019). IS-0200.c basic incident command system for initial response. Retrieved from https://training.fema.gov/emiweb/is/is200c/student%20manual/is0200c%20sm.pdf. Sep. 2020.

Feo, T. A. and Resende, M. G. C. (Oct., 1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, vol. 6, pp. 109–133.

Ferone, A. and Maratea, A. (2020). Decoy meta–clustering through rough graded possibilistic c-medoids. In: *Proceedings of the 9th Conference on Evolving and Adaptive Intelligent Systems*, pp. 1–7. IEEE.

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

Guedes, G. P., Ogasawara, E., Bezerra, E., and Xexeo, G. (Oct., 2016). Discovering top-k non-redundant clusterings in attributed graphs. *Neurocomputing*, vol. 210, pp. 45–54.

Hadjres, S., Belqasmi, F., El Barachi, M., and Kara, N. (Oct., 2020). A green, energy, and trust-aware multi-objective cloud coalition formation approach. *Future Generation Computer Systems*, vol. 111, pp. 52–67.

Horling, B. and Lesser, V. (Dec., 2004). A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, vol. 19, pp. 281–316.

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (May, 2010). Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, vol. 20, pp. 369–400.

Hübner, J. F., Sichman, J. S., and Olivier, B. (Dec., 2007). Developing organised multi-agent systems using the Moise+ model: Programming issues at the system and agent levels. *Agent-Oriented Software Engineering*, vol. 1, pp. 370–395.

Ieong, S. and Shoham, Y. (2005). Marginal contribution nets: A compact representation scheme for coalitional games. In: *Proceedings of the 6th Conference on Electronic Commerce*, pp. 193–202. ACM.

Irwin, R. L. (1989). The incident command system (ICS). In: *Disaster response: Principles of preparation and coordination*, vol. 1, pp. 30. C.V. Mosby, 1 ed..

Jensen, J. and Waugh Jr, W. L. (Feb., 2014). 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, pp. 5–17.

Keinänen, H. (2009). *Simulated Annealing for Multi-agent Coalition Formation*, chap. 4, pp. 30–39. Springer.

Kim, B., Lee, K., Lim, S., Kaelbling, L. P., and Lozano-Pérez, T. (2020). Monte carlo tree search in continuous spaces using Voronoi optimistic optimization with regret bounds. In: *Proceedings of the 34th Conference on Artificial Intelligence*, pp. 9916–9924. AAAI Press.

Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In: *Proceedings of the 17th European Conference on Machine Learning*, pp. 282–293. Springer.

Krausburg, T., Dix, J., and Bordini, R. H. (2021a). Computing sequences of coalition structures. In: *Proceedings of the 2nd Symposium Series on Computational Intelligence*, pp. 01–07. IEEE.

Krausburg, T., Dix, J., and Bordini, R. H. (2021b). Feasible coalition sequences. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 719–727. IFAAMAS.

Krysta, P. and Ventre, C. (Mar., 2015). Combinatorial auctions with verification are tractable. *Theoretical Computer Science*, vol. 571, pp. 21–35.

Lee, J., Jeon, W., Kim, G., and Kim, K. (2020). Monte-carlo tree search in continuous action spaces with value gradients. In: *Proceedings of the 34th Conference on Artificial Intelligence*, pp. 4561–4568. AAAI Press.

Lichtenstein, D. and Sipser, M. (Apr., 1980). GO is polynomial-space hard. *Journal of the ACM*, vol. 27, pp. 393–401.

Lin, C.-F. and Hu, S.-L. (2007). Multi-task overlapping coalition parallel formation algorithm. In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 211:1–211:3. ACM.

Mauro, N. D., Basile, T. M. A., Ferilli, S., and Esposito, F. (2010). Coalition structure generation with GRASP. In: *Proceedings of the 14th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pp. 111–120. Springer.

Michalak, T., Marciniak, D., Szamotulski, M., Rahwan, T., Wooldridge, M., McBurney, P., and Jennings, N. R. (2010a). A logic-based representation for coalitional games with externalities. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 125–132. IFAAMS.

Michalak, T., Rahwan, T., Elkind, E., Wooldridge, M., and Jennings, N. R. (Oct., 2015). A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, vol. 230, pp. 14–50.

Michalak, T., Sroka, J., Rahwan, T., Wooldridge, M., McBurney, P., and Jennings, N. R. (2010b). A distributed algorithm for anytime coalition structure generation. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1007–1014. IFAAMS.

Michalak, T. P., Dowell, A. J., McBurney, P., and Wooldridge, M. J. (2008). Optimal coalition structure generation in partition function games. In: *Proceedings of the 18th European Conference on Artificial Intelligence*, pp. 388–392. IOS Press.

Michalak, T. P., Tyrowicz, J., McBurney, P., and Wooldridge, M. (Jul., 2009). Exogenous coalition formation in the e-marketplace based on geographical proximity. *Electronic Commerce Research and Applications*, vol. 8, pp. 203–223.

Mouradian, C., Sahoo, J., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (Apr., 2017). A coalition formation algorithm for multi-robot task allocation in large-scale natural disasters. *Computing Research Repository*, vol. abs/1704.05905, pp. 1909–1914.

Mousavi, S., Afghah, F., Ashdown, J. D., and Turck, K. (May, 2019). Use of a quantum genetic algorithm for coalition formation in large-scale uav networks. *Ad Hoc Networks*, vol. 87, pp. 26–36.

Murphy, R. R. (2014). *Disaster Robotics.* The MIT Press.

Nguyen, T. (Jun., 2015). Coalitional bargaining in networks. *Operations Research*, vol. 63, pp. 501–511.

Power, N. (May-Jun., 2018). Extreme teams: Toward a greater understanding of multiagency teamwork during major emergencies and disasters. *American Psychologist*, vol. 73, pp. 478–490.

Präntare, F., Appelgren, H., and Heintz, F. (2021). Anytime heuristic and monte carlo methods for large-scale simultaneous coalition structure generation and assignment. In: *Proceedings of the 35th Conference on Artificial Intelligence*, pp. 11317–11324. AAAI Press.

Rahwan, T. and Jennings, N. (2008a). Coalition structure generation: dynamic programming meets anytime optimisation. In: *Proceedings of the 23rd National Conference on Artificial intelligence*, pp. 156–161. AAAI Press.

Rahwan, T. and Jennings, N. R. (Mar., 2007). An algorithm for distributing coalitional value calculations among cooperating agents. *Artificial Intelligence*, vol. 171, pp. 535–567.

Rahwan, T. and Jennings, N. R. (2008b). An improved dynamic programming algorithm for coalition structure generation. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 1417–1420. IFAAMS.

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

Rahwan, T., Michalak, T. P., Jennings, N. R., Wooldridge, M. J., and McBurney, P. (2009a). Coalition structure generation in multi-agent systems with positive and negative externalities. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 257–263. IJCAI Organization.

Rahwan, T., Michalak, T. P., Wooldridge, M., and Jennings, N. R. (Dec., 2015). Coalition structure generation: A survey. *Artificial Intelligence*, vol. 229, pp. 139–174.

Rahwan, T., Nguyen, T., Michalak, T. P., Polukarov, M., Croitoru, M., and Jennings, N. R. (2013). Coalitional games via network flows. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pp. 324–331. AAAI Press.

Rahwan, T., Ramchurn, S. D., Dang, V. D., Giovannucci, A., and Jennings, N. R. (2007a). Anytime optimal coalition structure generation. In: *Proceedings of the 6th International Conference on Natural Computation*, pp. 1184–1190. AAAI Press.

Rahwan, T., Ramchurn, S. D., Dang, V. D., and Jennings, N. R. (2007b). Near-optimal anytime coalition structure generation. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2365–2371. IJCAI Organization.

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

Ramchurn, S. D., Farinelli, A., Macarthur, K. S., and Jennings, N. R. (Mar., 2010). Decentralized Coordination in RoboCup Rescue. *The Computer Journal*, vol. 53, pp. 1447–1461.

Ramchurn, S. D., Wu, F., Jiang, W., Fischer, J. E., Reece, S., Roberts, S., Jennings, N. R., Rodden, T., and Greenhalgh, C. (Jan., 2016). Human-agent collaboration for disaster response. *Autonomous Agents and Multi-Agent Systems*, vol. 30, pp. 82–111.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (Jul., 1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, vol. 111, pp. 209–238.

Schadd, M. P. D., Winands, M. H. M., Tak, M. J. W., and Uiterwijk, J. W. H. M. (Oct., 2012). Single-player monte-carlo tree search for samegame. *Knowledge Based Systems*, vol. 34, pp. 3–11.

Sen, S. and Dutta, P. S. (2000). Searching for optimal coalition structures. In: *Proceedings of the 4th International Conference on Multi-Agent Systems*, pp. 287–292. IEEE.

Shehory, O. and Kraus, S. (May, 1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, vol. 101, pp. 165–200.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

Sierra, C., Aguilar, J. A. R., Noriega, P., Esteva, M., and Arcos, J. L. (Jul., 2004). Engineering multi-agent systems as electronic institutions. *The European Journal for the Informatics Professional*, vol. 4, pp. 33–39.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (Jan., 2016). Mastering the game of go with deep neural networks and tree search. *Nature*, vol. 529, pp. 484–489.

Skibski, O., Matejczyk, S., Michalak, T. P., Wooldridge, M., and Yokoo, M. (2016). k-coalitional cooperative games. In: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems*, pp. 177–185. IFAAMS.

Skibski, O., Michalak, T., Sakurai, Y., Wooldridge, M., and Yokoo, M. (2015). A graphical representation for games in partition function form. In: *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 1036–1042. AI Access Foundation.

Skibski, O., Michalak, T. P., Sakurai, Y., Wooldridge, M. J., and Yokoo, M. (Dec., 2020a). Partition decision trees: Representation for efficient computation of the shapley value extended to games with externalities. *Autonomous Agents and Multi-Agent Systems*, vol. 34, pp. 1–39.

Skibski, O., Suzuki, T., Grabowski, T., Michalak, T. P., and Yokoo, M. (2020b). Signed graph games: Coalitional games with friends, enemies and allies. In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pp. 1287–1295. IFAAMS.

Sukstrienwong, A. (Oct., 2018). A price-based mechanism for online buyer coalition by genetic algorithms. *International Journal of Innovative Computing, Information and Control*, vol. 14, pp. 1653–1679.

Thrall, R. M. and Lucas, W. F. (Mar., 1963). N-person games in partition function form. *Naval Research Logistics Quarterly*, vol. 10, pp. 281–298.

U.S. Department of Agriculture (2021). ICS 200 – incident command system. Retrieved from https://www.usda.gov/sites/default/files/documents/ICS200.pdf. Sep. 2021.

Voice, T., Polukarov, M., and Jennings, N. R. (Sep., 2012a). Coalition structure generation over graphs. *Journal of Artificial Intelligence Research*, vol. 45, pp. 165–196.

Voice, T., Ramchurn, S. D., and Jennings, N. R. (2012b). On coalition formation with sparse synergies. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 223–230. IFAAMS.

Winands, M. H. M., Björnsson, Y., and Saito, J.-T. (2008). Monte-carlo tree search solver. In: *Proceedings of the 6th International Conference on Computers and Games*, pp. 25–36. Springer.

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems.* Wiley Publishing.

Wu, F. and Ramchurn, S. D. (2020). Monte-carlo tree search for scalable coalition formation. In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence*, pp. 407–413. IJCAI Organization.

Xu, J. and Li, W. (2008). Solution of overlapping coalition formation based on discrete particle swarm optimization. In: *Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1–4. IEEE.

Yeh, D. Y. (Dec., 1986). A dynamic programming approach to the complete set partitioning problem. *BIT Numerical Mathematics*, vol. 26, pp. 467–474.

Yoshizoe, K., Kishimoto, A., Kaneko, T., Yoshimoto, H., and Ishikawa, Y. (2011). Scalable distributed monte-carlo tree search. In: *Proceedings of the 4th Symposium on Combinatorial Search*, pp. 180–187. AAAI Press.

Zha, A., Nomoto, K., Ueda, S., Koshimura, M., Sakurai, Y., and Yokoo, M. (2017). Coalition structure generation for partition function games utilizing a concise graphical representation. In: *Proceedings of the 20th International Conference on Principles and Practice of Multi-Agent Systems*, pp. 143–159. Springer.

Zhang, G., Jiang, J., Lu, C., Su, Z., Fang, H., and Liu, Y. (Mar., 2011). A revision algorithm for invalid encodings in concurrent formation of overlapping coalitions. *Applied Soft Computing*, vol. 11, pp. 2164–2172.

Zhang, G., Jiang, J., Su, Z., Qi, M., and Fang, H. (Sep., 2010). Searching for overlapping coalitions in multiple virtual organizations. *Information Sciences*, vol. 180, pp. 3140–3156.

Zhang, J., Yu, P. S., and Lv, Y. (2017a). Enterprise employee training via project team formation. In: *Proceedings of the 10th International Conference on Web Search and Data Mining*, pp. 3–12. ACM.

Zhang, R., Zhao, Z., Cheng, X., and Yang, L. (Aug., 2017b). Overlapping coalition formation game based opportunistic cooperative localization scheme for wireless networks. *IEEE Transactions on Communications*, vol. 65, pp. 3629–3642.

Zick, Y., Chalkiadakis, G., and Elkind, E. (2012). Overlapping coalition formation games: Charting the tractability frontier. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pp. 787–794. IFAAMS.

Zick, Y., Chalkiadakis, G., Elkind, E., and Markakis, E. (Jun., 2019). Cooperative games with overlapping coalitions: Charting the tractability frontier. *Artificial Intelligence*, vol. 271, pp. 74 – 97.

Zick, Y. and Elkind, E. (2011). Arbitrators in overlapping coalition formation games. In: *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, pp. 55–62. IFAAMS.

APPENDIX A – GLOSSARY

Acronym	Var	Meaning
—	A	a set of agents
—	n	the size of A
—	C	a subset of agents (coalition)
—	$\boldsymbol{\mathcal{C}}^A$	the set of all coalitions over A
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 tuple $\langle A, v \rangle$
—	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
—	Z	a set of allowed coalition sizes
SVS	π	a tuple $\langle G, S, Z \rangle$
—	$\bar{\pi}$	given a SVS π, a tuple $\langle G, S, \{1, 2, \ldots, \max(Z)\} \rangle$
—	$\boldsymbol{\mathcal{C}}^\pi$	the set of all coalitions induced by an SVS π
—	$\boldsymbol{\mathcal{CS}}^\pi$	the set of all CSs induced by an SVS π
—	Γ^π	a CFG game induced by an SVS π
—	$\mathcal{H}$	a totally ordered sequence of CFGs
—	h	the length of $\mathcal{H}$
—	$\mathcal{R}$	a binary relation on $\boldsymbol{\mathcal{CS}}^A$
—	X^{CS}	$\{CS' \in \boldsymbol{\mathcal{CS}}^A \mid CS \; \mathcal{R} \; CS'\}$
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	$\mathcal{G}$	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 $\langle A, \mathcal{H}, \Pi, \mathcal{R} \rangle$

continued on next page

Table A.1 – continued from previous page

Acronym	Var	Meaning
—	X_l^{CS}	a set of CSs in X^{CS} that follow the constraints of level l
—	$Tree(\mathsf{root})$	a rooted tree with root root which corresponds to $\emptyset$
—	$\mathsf{x}, \mathsf{y}, \mathsf{z}$	nodes in the tree
—	$Tree(\mathsf{x})$	the tree induced by the descendants of x
—	$path$	a sequence of nodes $\langle \emptyset, \mathsf{x}_1, \ldots, \mathsf{x}_k \rangle$
—	$Act[CS]$	a list of CSs built on the fly corresponding to X^{CS}
—	$\mathsf{x}.parent$	the parent node of x
—	$\mathsf{x}.CS$	the corresponding CS of x
—	$\mathsf{x}.l$	the level where x is placed in the tree
—	$\mathsf{x}.children$	a finite list of child nodes of x
—	$\mathsf{x}.N$	a counter of visits to x
—	$\mathsf{x}.val$	the cumulative reward of x
—	$\mathsf{x}.expanded$	a Boolean variable stating whether x has been fully expanded
—	$\mathsf{x}.terminal$	a Boolean variable stating whether x is terminal
—	$\mathsf{x}.i$	an index pointing to the last CS evaluated from $Act[\mathsf{x}.CS]$
—	L	a set $\bigcup_{i=1}^{h} S_i$
—	Cap	a set of capabilities
—	R	a set of roles
IO	IO	a set of incident objectives
—	cap	a function mapping $A \cup R \to 2^{Cap}$
—	$demand$	a function mapping $IO \to 2^R$
—	$resp$	a function mapping $S \to 2^{IO}$
—	$relationship$	a function mapping $A \times A \to [0, 1]$
—	$disturbance$	a function mapping $2^A \to [0, 1]$
—	$adopt(a_i)$	a set $\{r \in R \mid cap(r) \subseteq cap(a_i)\}$
—	$expect(a_i)$	a set $\{r \in R \mid o \in resp(a_i), r \in demand(o)\}$
—	$require(C)$	a multiset of roles required by the superiors in C
—	$available(C)$	a multiset of roles adopted by ordinary agents in C
—	e^r	the number of agents that should adopt role r

APPENDIX B – SYNTHETIC INFORMATION FOR THE ROARING RIVER FLOOD SCENARIO

To properly model the Roaring River Flood (RRF) problem, we assume two important information that are missing in the scenario description (U.S. Department of Agriculture, 2021). Those are: (i) capabilities of interest for the disaster response operation; and (ii) the set of roles that provide the expected capabilities. In the ICS training course (U.S. Department of Agriculture, 2021), six general incident objectives are introduced:

1. euthanize suffering animals;
2. dispose of animal corpses;
3. identify relocation sites and relocate animals;
4. control the movement of host material;
5. eradicate fruit flies; and
6. survey and identify fruit flies.

However, the last goal contains two main objectives that are unified for Survey/ID Group (Figure 6.2). Thus, we break it into two IOs as introduced in Table B.3.

Given the IOs above, we determine the set of capabilities required to properly respond to the flood incident. All capabilities are shown in Table B.1.

Table B.1: Capabilities proposed to model the RRF scenario.

Id	Capability Description	Id	Capability Description
(1)	handling of animals	(2)	medical care to ill or injured animals
(3)	enforce animal-related laws	(4)	provide safe and humane capture
(5)	containment of animals	(6)	depopulate animals
(7)	evacuation of animals	(8)	identify and document animals
(9)	animal tracking	(10)	load and unload animals
(11)	reunificate animals with their owners	(12)	search and rescue of animals
(13)	triage/prepare animals for transport	(14)	coordinate captures
(15)	medical triage of animals	(16)	clinical examinations
(17)	perform surgery	(18)	investigate cases of animal disease
(19)	support vet duties	(20)	assess damaged site
(21)	assess agriculture infrastructure	(22)	assist in planning response/recovery
(23)	spray chemical products	(24)	assess behaviour of animals

Our next step in to determine the roles for the RRF problem. To do so, we looked up in the Resource Typing Library Tool (RTLT)[1] roles that have functions similar to the capabilities above. The set of roles found and their corresponding capabilities are shown in Table B.2.

[1] https://rtlt.preptoolkit.fema.gov/Public

Table B.2: Roles collected from the resource typing library tool.

Role	Id	Acronym	Capability Ids
Animal Control/Humane Officer[2]	(1)	ACHO	3, 4, 5
Animal Depopulation Specialist[3]	(2)	ADS	6, 23
Animal Depopulation Team[4]	(3)	ADT	6, 23
Animal Evacuation, Transport, and Re-Entry Team[5]	(4)	AETRT	1, 7, 8, 9, 10, 11
Animal Search and Rescue Team[6]	(5)	ASRT	8, 12, 13, 14
Animal Search and Rescue Technician[7]	(6)	ASRTec	9, 12, 7
Veterinary Medical Team[8]	(7)	VMT	2, 15, 16, 17, 18, 19
Animal and Agriculture Damage Assessment Team[9]	(8)	AADAT	13, 20, 21, 22
Animal Behavior Specialist[10]	(9)	ABS	24

Finally, we map each incident objective to a set of roles. This is process is commonly conducted by the incident commander or Operations Section chief (FEMA, 2017).

Table B.3: Incident objectives for the RRF and their corresponding group and required roles.

General Incident Task	Group	Role Ids
euthanize suffering animals	Euthanasia	3
begin the disposal operation	Disposal	2, 4, 6
identify relocation sites and relocate animals	Relocation	4, 5, 7
control the movement of host material	Regulatory	1
eradicate the fruit flies	Control	2
survey for fruit fly locations	Survey/ID	8
identify fruit flies	Survey/ID	9

[2]https://rtlt.preptoolkit.fema.gov/Public/Position/View/1-509-1333?q=Animal%20Control%2FHumane%20Officer

[3]https://rtlt.preptoolkit.fema.gov/Public/Position/View/1-509-1335?q=Animal%20Depopulation%20Specialist

[4]https://rtlt.preptoolkit.fema.gov/Public/Resource/View/1-508-1222?q=Animal%20Depopulation%20Team

[5]https://rtlt.preptoolkit.fema.gov/Public/Resource/View/1-508-1223?q=Animal%20Evacuation%2C%20Transport%2C%20and%20Re-Entry%20Team

[6]https://rtlt.preptoolkit.fema.gov/Public/Resource/View/1-508-1224?q=Animal%20Search%20and%20Rescue%20Team

[7]https://rtlt.preptoolkit.fema.gov/Public/Position/View/1-509-1339?q=Animal%20Search%20and%20Rescue%20Technician

[8]https://rtlt.preptoolkit.fema.gov/Public/Resource/View/1-508-1230?q=Veterinary%20Medical%20Team

[9]https://rtlt.preptoolkit.fema.gov/Public/Resource/View/1-508-1221?q=Animal%20and%20Agriculture%20Damage%20Assessment%20Team

[10]https://rtlt.preptoolkit.fema.gov/Public/Position/View/1-509-1331?q=Animal20Behavior20Specialist

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br