Optimal Anytime Coalition Structure Generation Utilizing Compact Solution Space Representation

Redha Taguelmimt[1] , **Samir Aknine**[1] , **Djamila Boukredera**[2] , **Narayan Changder**[3] and **Tuomas Sandholm**[4,5,6,7]

[1]Univ Lyon, UCBL, CNRS, INSA Lyon, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Lyon, France
[2]Laboratory of Applied Mathematics, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria
[3]TCG Centres for Research and Education in Science and Technology, Kolkata, India
[4]Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
[5]Strategic Machine, Inc.
[6]Strategy Robot, Inc.
[7]Optimized Markets, Inc.
redha.taguelmimt@gmail.com, samir.aknine@univ-lyon1.fr, djamila.boukredera@univ-bejaia.dz, narayan.changder@tcgcrest.org, sandholm@cs.cmu.edu

Abstract

Coalition formation is a central approach for multi-agent coordination. A crucial part of coalition formation that is extensively studied in AI is *coalition structure generation*: partitioning agents into coalitions to maximize overall value. In this paper, we propose a novel method for coalition structure generation by introducing a compact and efficient representation of coalition structures. Our representation partitions the solution space into smaller, more manageable subspaces that gather structures containing coalitions of specific sizes. Our proposed method combines two new algorithms, one which leverages our compact representation and a branch-and-bound technique to generate optimal coalition structures, and another that utilizes a preprocessing phase to identify the most promising sets of coalitions to evaluate. Additionally, we show how parts of the solution space can be gathered into groups to avoid their redundant evaluation and we investigate the computational gain that is achieved by avoiding that redundant processing. Through this approach, our algorithm is able to prune the solution space more efficiently. Our results show that the proposed algorithm is superior to prior state-of-the-art methods in generating optimal coalition structures under several value distributions.

1 Introduction

Coalition formation is an important research area in multiagent systems. Forming certain coalitions can be more valuable than forming others. A crucial part of coalition formation is *coalition structure generation (CSG)*: partitioning agents into coalitions to maximize overall value, that is, finding an optimal *coalition structure*. This has a number of important applications such as collaboration among trucking companies [Sandholm and Lesser, 1997], distributed sensor networks [Dang *et al.*, 2006], etc.

This coalition structure generation problem has been extensively studied in AI. It is NP-complete [Sandholm *et al.*, 1999], and several algorithms have been proposed to solve it either optimally or approximately. Many works [Yeh, 1986; Rahwan and Jennings, 2008; Michalak *et al.*, 2016; Taguelmimt *et al.*, 2022b] proposed *dynamic programming* algorithms, which guarantee finding the optimal coalition structure but must be run to completion to do so. To allow returning solutions prior to termination, a number of works [Sandholm *et al.*, 1999; Dang and Jennings, 2004; Rahwan *et al.*, 2009; Taguelmimt *et al.*, 2022a] have turned to developing *anytime* algorithms in an effort to allow premature termination while providing intermediate solutions during the algorithm's execution. When the number of agents increases and the problem becomes too hard, *heuristic* algorithms present a practical option. For this reason, many works [Sen and Dutta, 2000; Keinänen, 2009; Di Mauro *et al.*, 2010; Taguelmimt *et al.*, 2021] have presented algorithms that focus on speed and do not guarantee finding an optimal solution.

The fastest exact algorithms to date are hybrid solutions called ODP-IP [Michalak *et al.*, 2016], ODSS [Changder *et al.*, 2020], and BOSS [Changder *et al.*, 2021] that combine IDP [Rahwan and Jennings, 2008] and IP [Rahwan *et al.*, 2009]. IDP is based on dynamic programming and computes the optimal solution for n agents by computing an optimal partition of all the coalitions $\mathcal{C}$ of size $|\mathcal{C}| \in \{2, ..., n\}$. In contrast, IP uses an integer representation of the search space and computes the optimal solution by traversing in a depth-first manner multiple search trees and uses branch-and-bound to speed up the search. Those algorithms are efficient on some value distributions. However, for many value distributions, they fail to produce an optimal solution within reasonable time. Also, the hybridization of IDP and IP in these two algorithms relies heavily on the effectiveness of IP. Thus, the time required by the algorithms grows considerably when IP is not fast enough. Moreover, several coalition structures share common parts, that is, they contain common coalitions. The evaluation of such common coalitions is duplicated in those algorithms because they evaluate the common parts of some coalition structures separately. This repeated processing is wasteful and can be highly time consuming. In light of these observations, and to address the aforementioned issues, we present a new method that uses a novel compact search space representation and a new algorithm to efficiently explore the coalition structure search space. In short, our main contributions are the following.

- A novel compact representation of the search space that gathers solution subspaces in a new way. This representation not only enables to optimize the evaluation of coalition structures but also enables to eliminate the redundant processing of common coalition structures, resulting in a significant reduction in computational time.
- A hybrid algorithm, ELIXIR, that combines the strengths of graph search and dynamic programming to provide both speed and optimality in exploring the search space. It uses the new representation to optimize the evaluation of the coalition structures and a new and refined selection technique for the coalitions to evaluate in dynamic programming.
- Empirical evaluation showing that ELIXIR generates optimal solutions faster than prior state-of-the-art algorithms. Moreover, we provide a thorough theoretical analysis of the proposed method to prove its soundness.

The paper is organized as follows. We begin by presenting the CSG problem in Section 2. In Sections 3 and 4, we describe our novel representation of the search space and our algorithm. In section 5, we present our experiments. Finally, Section 6 draws some conclusions.

2 Problem Formulation

In this paper, we use the terms *coalition structure* and *solution* interchangeably. A CSG problem instance is specified by a set $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ of n agents and a characteristic function v assigns a real value $v(\mathcal{C})$ to each coalition $\mathcal{C}$. The size of the CSG problem is n. A coalition $\mathcal{C}$ in $\mathcal{A}$ is any non-empty subset of $\mathcal{A}$. The size of a coalition $\mathcal{C}$ is $|\mathcal{C}|$. This value captures the desirability (e.g., efficiency) of the coalition. A coalition structure $\mathcal{CS}$ is a partition of the set of agents $\mathcal{A}$ into disjoint, exhaustive coalitions. Given a set of non-empty coalitions $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, a coalition structure is a collection of coalitions $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$, which satisfies the following constraints: $\bigcup_{j=1}^{k} \mathcal{C}_i = \mathcal{A}$ and for all $i, j \in \{1, 2, ..., k\}$ where $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$.

$\Pi(\mathcal{A})$ denotes the set of all coalition structures. The value of a coalition structure $\mathcal{CS}$ is assessed as the sum of the values of the coalitions that comprise it: $v(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$. An optimal solution to the coalition structure generation problem is a value-maximizing coalition structure: $\mathcal{CS}^* \in \Pi(\mathcal{A})$ such that $\mathcal{CS}^* = \arg\max_{\mathcal{CS} \in \Pi(\mathcal{A})} V(\mathcal{CS})$.

3 Compact Solution Subspace Graph Representation

In this section we introduce our compact representation for the problem. The *compact solution subspace (CSS) graph* is an undirected graph that consists of a number of nodes that are made up of two components each: a set of numbers $x_1, .., x_p$ and a set of variables $v_1, .., v_q$. The vector composed of these numbers and variables $[x_1, .., x_p, v_1, .., v_q]$, where $p + q \leq n$, represents an ordered integer partition of n, that is, a vector of positive integers, ordered from smallest to largest that sum to n. This vector satisfies certain constraints, $\forall i, j \in \{1, .., p\}$, where $i < j$, $x_i \leq x_j$ and $\forall i, j \in \{1, .., q\}$, where $i < j$, $v_i \leq v_j$ and $x_p \leq v_1$), which guarantees the order. The series of numbers $x_1, .., x_p$ of this vector is called an integer prefix. Given this, every node in this graph represents a set of ordered integer partitions that share the same integer prefix. For instance, in Figure 1 with ten agents, the node $[1, 1, v_4, v_4']$ gathers 4 ordered integer partitions, namely $[1, 1, 1, 7]$, $[1, 1, 2, 6]$, $[1, 1, 3, 5]$, $[1, 1, 4, 4]$, which share the same integer-prefix $1, 1$. These partitions correspond to the different possible values that the variables v_4 and v_4' can take, while ensuring that the sum of the parts of each partition is equal to n, that is, $1 + 1 + v_4 + v_4' = n = 10$. The *index* e of a node is the number of partitions that the node gathers, which is 4 for the node $[1, 1, v_4, v_4']$ (see the red rectangles in Figure 1). Additionally, the degree k of a CSS graph is defined as the number of variables its nodes have. Using this, we can define the level of compaction in our graph. Figure 1 illustrates a degree 2 CSS graph where there are two variables. However, by using more variables in building a CSS graph, we can increase the level of compaction and decrease the number of nodes in the graph. Conversely, using fewer variables decreases the level of compaction. Therefore, by adjusting the number of variables used in the CSS graph, we can fine-tune the level of compaction and find the most efficient degree for the problem at hand. For example, if we have partitions [1,1,2,2,4] and [1,1,1,3,4], if we use three variables, both partitions will be gathered in the same node as they share the prefix 1,1, however, if we use only two variables, these two partitions will be separated as the prefix for the first one is 1,1,2 and for the second one is 1,1,1.

The nodes in the CSS graph are distributed across several levels, each representing a different number of parts of the integer partitions. Each level $l \in \{1, .., n\}$ contains nodes representing groups of ordered integer partitions of n that contain l parts. For instance, level 4 contains nodes where integer partitions of n have four parts. It should also be noted that a node cannot have more variables than parts. As an example, if $k = 5$, the node in level 2, which contains 2 parts, will only have two variables. Each group represents a set of coalition structures in which the sizes of the coalitions match the parts of the group. For instance, the group $[1, 1, v, v', v'']$, with $n = 10$ agents, consists of all coalition structures that contain two coalitions of size 1 and three other coalitions, where the sum of the sizes of these three coalitions is 8 (10-1-1). We denote the set of all integer partitions that belong to a group $\mathcal{G}$ by $I^{\mathcal{G}}$. A special case is the node $[n]$, which is at level 1; it always contains a unique coalition of size n (i.e., which contains all agents). In the CSS graph, two nodes representing groups of ordered integer partitions $\mathcal{G}_1$ and $\mathcal{G}_2$ are connected by an edge if and only if one of the integer partitions gathered in the group $\mathcal{G}_2$ can be obtained from one of the integer partitions gathered in the group $\mathcal{G}_1$ by splitting only one integer. For example, the nodes $[1, v_1, v'_1]$ and $[1, 1, v_4, v'_4]$ are connected because the integer partition $[1, 1, 2, 6]$ (gathered in $[1, 1, v_4, v'_4]$) can be obtained from the partition $[1, 1, 8]$ (gathered in $[1, v_1, v'_1]$) by splitting the integer 8 into 2 and 6. It should be noted that the number of considered variables can range from 0 to n. In the case where there are no variables, each node will contain only one integer partition. If there is only one variable, there is only one integer that the variable can take, hence each node will still contain only one partition. Based on this, the following theorem holds.

Theorem 1. *Let $\mathcal{N}$ denote the number of nodes in the CSS graph. For every problem size n, we have $n \leq \mathcal{N} \leq \mathcal{O}(\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n})$.*

Proof. The number of nodes $\mathcal{N}$ of the CSS graph depends on its degree. If it is 0 or 1, each node gathers one integer partition and therefore, $\mathcal{N}$ is equal to the number of integer partitions, which is known to be $O(\frac{e^{\pi\sqrt{2n/3}}}{n})$ [Wilf, 2000]. Then, the higher the degree, the more partitions are gathered in the nodes and the fewer nodes the CSS graph contains. Thus, $\mathcal{N} \leq \mathcal{O}(\frac{e^{\pi\sqrt{2n/3}}}{n})$. Since at least one subspace must appear on a level, and for a problem of size n there are n levels, there are at least n nodes in the graph. Thus, $\mathcal{N} \geq n$. □

For the remainder of this paper, we use the terms *partition group*, *subspace group*, and *node* interchangeably.

4 ELIXIR Algorithm

Our main algorithm, ELIXIR (ordEred-soLutIon-subspace focused eXploration with refined dynamIc pRogramming), is based on the CSS graph and combines dynamic programming with CSS graph search. It combines two algorithms, CSSA and RDP, which we present in the following two subsections, respectively. The subsection after that describes how those two algorithms are combined in ELIXIR.

4.1 CSS Search Algorithm (CSSA)

The *CSS Search Algorithm (CSSA)* is based on our *compact solution subspace graph representation* of the space of possible coalition structures. Using this representation, CSSA starts by computing upper bounds for the values of the best coalition structures that can be found in each subspace group. These upper bounds are calculated by identifying the highest value of the coalitions of each size, denoted as Max_s, and then summing them for each subspace, $UB_{\mathcal{S}} = \sum_{s \in Integers(\mathcal{S})} Max_s$, where $Integers(\mathcal{S})$ is the set of integers that form the integer partition of the subspace $\mathcal{S}$. For example, for $\mathcal{S} = [1, 2, 3, 4]$, the upper bound would be $UB_{[1,2,3,4]} = Max_1 + Max_2 + Max_3 + Max_4$. By computing the upper bound for every subspace, CSSA is able to compute the group upper bound of each subspace group: $GUP_{\mathcal{G}} = \text{Max}_{\mathcal{S} \in I^{\mathcal{G}}} UB_{\mathcal{S}}$. For example,

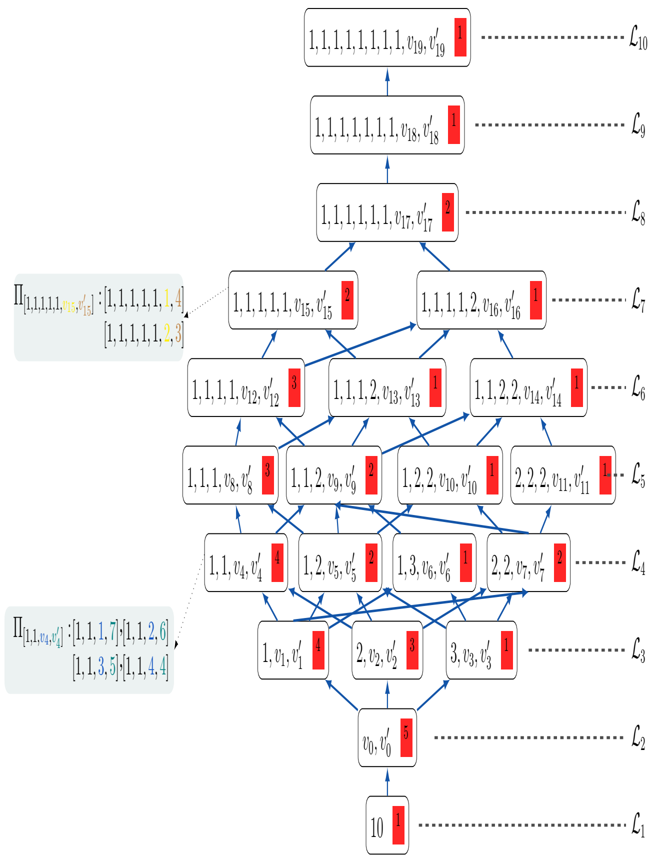

Figure 1: A ten-agent Compact Solution Subspace (CSS) graph of degree 2. This graph has 10 levels of nodes: $\mathcal{L}_1$ to $\mathcal{L}_{10}$

for $\mathcal{G} = [1, 2, v_5, v'_5]$, $I^{\mathcal{G}} = \{[1, 2, 3, 4], [1, 2, 2, 5]\}$ and $GUP_{\mathcal{G}} = \text{Max}_{\mathcal{S} \in \{[1,2,3,4],[1,2,2,5]\}} UB_{\mathcal{S}} = \max(UB_{[1,2,3,4]}, UB_{[1,2,2,5]})$.

With the knowledge of the upper bounds of each subspace and subspace group, the CSSA algorithm is able to prioritize and explore the most promising groups first. It does this by sorting the subspace groups according to their group upper bounds. To keep track of the best solution found so far, CSSA maintains a record of the current best coalition structure found $\mathcal{CS}^+$ and its value $v(\mathcal{CS}^+)$. To optimize the search, CSSA uses a pruning strategy. As it explores the subspace groups, it prunes away those that have a group upper bound $GUB < v(\mathcal{CS}^+)$. This is because these subspace groups cannot contain an optimal solution. Furthermore, within each subspace group that has a group upper bound greater than $v(\mathcal{CS}^+)$, CSSA identifies subspaces that have an upper bound $UB < v(\mathcal{CS}^+)$ and eliminates them from further consideration, further reducing the number of subspaces to be explored. By eliminating these subspaces, they will not be evaluated when searching the subspace group, which helps to decrease the number of elements of the groups during execution. This process allows CSSA to focus its search on the most promising subspaces and reduces the overall search time. The order of subspace and subspace group exploration is always based on their upper bounds, with the group with the highest group upper bound being searched first, and within a group, the subspace with the highest upper bound is considered first.

To search the subspace groups, CSSA employs a combination of a depth-first search and a branch-and-bound technique. For each subspace group that has a remaining subspace with an upper bound greater than $v(\mathcal{CS}^+)$, CSSA builds search trees, with each node representing a coalition, and each path from the root to a leaf representing a coalition structure (Figure 2). During the depth-first search, CSSA explores coalitions of increasing size, ensuring that the coalitions considered do not overlap. To explore a node $\mathcal{G} = [x_1, x_2, ..., x_p, v_1, .., v_q]$, CSSA iterates over the coalitions of size x_1 and, for every coalition $C_1^{x_1}$, it iterates over the coalitions of size x_2 that do not overlap with $\mathcal{C}_1^{x_1}$. Likewise, for every coalition $\mathcal{C}_2^{x_2}$, it iterates over the coalitions of size x_3 that do not overlap with $\mathcal{C}_1^{x_1} \cup \mathcal{C}_2^{x_2}$, and so on up to v_q. For v_1 to v_q, CSSA iterates over the coalitions of the sizes that can correspond to each variable and that have not been pruned out. This way, each time CSSA iterates over the coalitions of the sizes v_q, it constructs coalition structures composed of $p + q$ coalitions. This process is repeated until every coalition structure of the group is examined. To improve the efficiency of the search process, CSSA applies a branch-and-bound technique to identify and prune search branches that have no chance of leading to an optimal solution. Specifically, at each depth d of the search, CSSA evaluates the current set of coalitions $\mathcal{C}_1^{x_1}, .., \mathcal{C}_d^{x_d}$ and checks if they have the potential to lead to a coalition structure with a value greater than the current best solution $v(\mathcal{CS}^+)$. This is done by comparing the current accumulated value of the coalitions $\sum_{i=1}^{d} v(\mathcal{C}i)$ plus the upper bound of the remaining coalitions, $\sum_{i=d+1}^{p+q} Max_{x_i}$, to the value of the current best solution $v(\mathcal{CS}^+)$. If the comparison shows that the current search path cannot lead to a better solution ($\sum_{i=1}^{d} v(\mathcal{C}_i) + \sum_{i=d+1}^{p+q} Max_{x_i} < v(\mathcal{CS}^+)$), then it is pruned and the search process continues with other branches, meaning that all the coalition structures that contain $\mathcal{C}_1^{x_1}, .., \mathcal{C}_d^{x_d}$ can be skipped because their values cannot be greater than $v(\mathcal{CS}^+)$. Figure 2 provides an illustration of how CSSA proceeds on an example with ten agents and a four-part subspace group $[1, 2, v_5, v_5']$ of a degree 2 CSS.

We refer to every non-empty collection of coalitions within a coalition structure as a *coalition structure part*. Formally, for a coalition structure $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, any collection $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_j\}$, where $j \leq k$, is a coalition structure part of $\mathcal{CS}$. For example, $\{\{a_2\}\}$ and $\{\{a_2\}, \{a_1, a_5\}\}$ are coalition structure parts of the coalition structure $\{\{a_2\}, \{a_1, a_5\}, \{a_3, a_4, a_6\}\}$. With this definition, the following theorem quantifies the reduction in the number of coalition structure parts that are evaluated by CSSA.

Theorem 2. *Let q be the degree of a CSS graph for a problem size n, and let $N(x_i)$ denote the number of coalitions of size x_i. For every subspace group $\mathcal{G} = [x_1, ..., x_p, v_1, .., v_m]$, CSSA avoids evaluating $(d-1) \times \prod_{i=1}^{p} N(x_i)$ coalition structure parts, where d is the number of integer partitions of the subspace group.*

Proof. Fix a number of agents n, a CSS graph degree q and a subspace group $\mathcal{G} = [x_1, ..., x_p, v_1, .., v_m]$. All integer partitions gathered in $\mathcal{G}$ have the same coalition structure parts. The number of the common coalition structure parts is $\prod_{i=1}^{p} N(x_i)$. When searching the subspace group as a whole, these parts are considered only once. However, if each integer partition is searched apart from the others, these coalition structure parts will be evaluated for each integer partition, which makes the number of evaluated parts in this case $d \times \prod_{i=1}^{p} N(x_i)$. Hence, CSSA avoids evaluating $(d - 1) \times \prod_{i=1}^{p} N(x_i)$ coalition structure parts. □

In other words, CSSA evaluates only one coalition structure part for each possible combination of coalitions of sizes x_1 to x_p, and avoids evaluating the same part for each of the $d - 1$ remaining integer partitions of the subspace group. This unique approach sets our algorithm, ELIXIR, apart from other methods such as ODP-IP and BOSS, which rely on commonly used representations.

4.2 Refined Dynamic Programming (RDP)

Our RDP algorithm is based on dynamic programming. Given n agents, to find an optimal coalition structure, RDP computes for every selected coalition $\mathcal{C} \subseteq \mathcal{A}$ the best partition of $\mathcal{C}$ and stores it in a partition table $\mathcal{P}_t$ as $\mathcal{P}_t(\mathcal{C})$. RDP also stores the value of the best partition of $\mathcal{C}$ in a value table $\mathcal{V}_t$ as $\mathcal{V}_t(\mathcal{C})$. When the best partitions of each coalition are obtained, the optimal coalition structure corresponds to the best partition of the grand coalition $\mathcal{A}$. An example is given in Section D of the appendix.

To select the coalitions to evaluate (for which we compute the best partitions), we use a preprocessor that returns a set of coalition sizes to consider for evaluating the coalitions. This set of coalition sizes $\mathcal{S}$ identifies all coalitions of size $|\mathcal{C}| \in \mathcal{S}$. We now introduce the fundamental principle of the preprocessing phase. Given n agents, to guarantee finding an optimal solution, not all coalitions need to be evaluated. In fact, several different sets of coalitions can be considered for evaluation to search all the subspaces and guarantee finding an optimal solution. This can be visualized in the CSS graph of degree 2 by unfolding the nodes, that is, by showing the partitions that a node gathers (see Figure 3 for an example with 4 agents). Each node in the graph on the left represents an integer partition (a subspace), and each edge connects two adjacent nodes if and only if the integer partition in level i can be obtained from the one in level $i - 1$ by splitting only one integer.

Starting from the bottom node, each edge that connects two nodes represents the splitting of a coalition size into two smaller sizes, meaning that all coalitions of that original size are evaluated. For example, the red edge that connects the node $[1, 3]$ to the node $[1, 1, 2]$ reflects a split of the coalition size 3 into the sizes 1 and 2 ($3 = 1 + 2$), meaning that all coalitions of size 3 are evaluated (they are divided into two coalitions of sizes 1 and 2 to obtain their best partitions). Hence, dividing a coalition into two corresponds to an upward movement in the graph (e.g. from $[1, 3]$ to $[1, 1, 2]$ with a coalition of size 3, or from $[2, 2]$ to $[1, 1, 2]$ and from $[1, 1, 2]$ to $[1, 1, 1, 1]$ with a coalition of size 2). The following theorem proves that to fully search a subspace of solutions, it is not necessary to consider all coalition sizes.

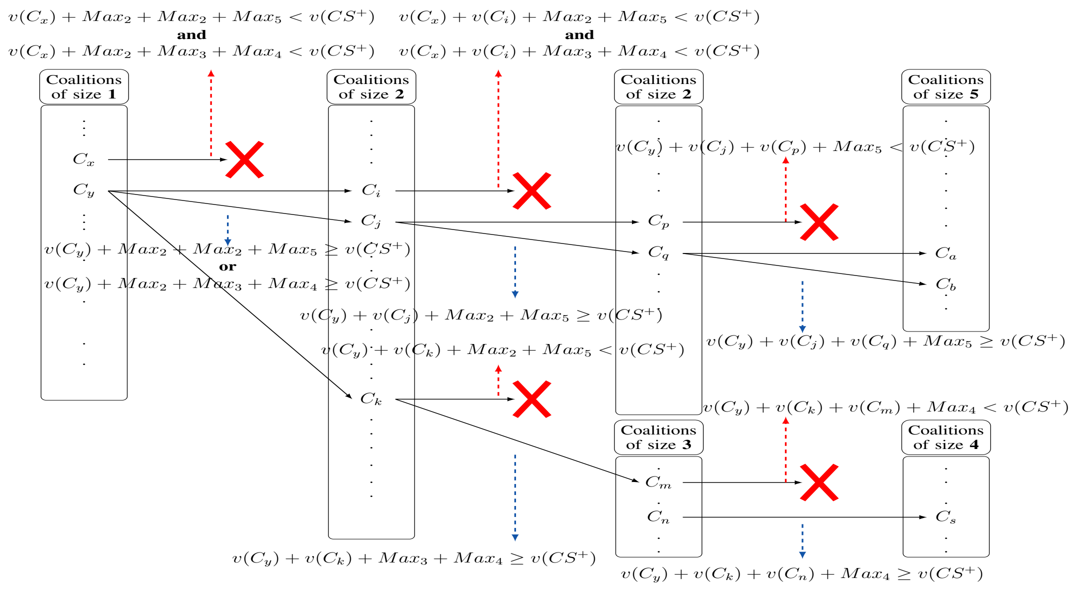
$v(C_x) + Max_2 + Max_2 + Max_5 < v(CS^+)$
and
$v(C_x) + Max_2 + Max_3 + Max_4 < v(CS^+)$
$v(C_x) + v(C_i) + Max_2 + Max_5 < v(CS^+)$
and
$v(C_x) + v(C_i) + Max_3 + Max_4 < v(CS^+)$
Coalitions of size 1
Coalitions of size 2
Coalitions of size 2
Coalitions of size 5
Coalitions of size 3
Coalitions of size 4
C_x
C_y
C_i
C_j
C_k
C_p
C_q
C_m
C_n
C_a
C_b
C_s
$v(C_y) + v(C_j) + v(C_p) + Max_5 < v(CS^+)$
$v(C_y) + Max_2 + Max_2 + Max_5 \geq v(CS^+)$
or
$v(C_y) + Max_2 + Max_3 + Max_4 \geq v(CS^+)$
$v(C_y) + v(C_j) + Max_2 + Max_5 \geq v(CS^+)$
$v(C_y) + v(C_k) + Max_2 + Max_5 < v(CS^+)$
$v(C_y) + v(C_j) + v(C_q) + Max_5 \geq v(CS^+)$
$v(C_y) + v(C_k) + v(C_m) + Max_4 < v(CS^+)$
$v(C_y) + v(C_k) + Max_3 + Max_4 \geq v(CS^+)$
$v(C_y) + v(C_k) + v(C_n) + Max_4 \geq v(CS^+)$

Figure 2: Illustration of the branch-and-bound technique when searching the subspace group $[1, 2, v_5, v'_5]$ of a graph with 10 agents and 2 variables as in Figure 1. In this example, the algorithm recognizes that the coalition structures containing the coalition C_x (resp. the coalitions C_y, C_k and C_m) because $v(\mathcal{C}_x) + Max_2 + Max_2 + Max_5 < v(\mathcal{CS}^+)$ and $v(\mathcal{C}_x) + Max_2 + Max_3 + Max_4 < v(\mathcal{CS}^+)$ (resp. because $v(\mathcal{C}_y) + v(\mathcal{C}_k) + v(\mathcal{C}_m) + Max_4 < v(\mathcal{CS}^+)$), cannot be optimal. Thus, CSSA does not search further in the tree. With this, CSSA searches several integer partitions of coalition structures simultaneously.

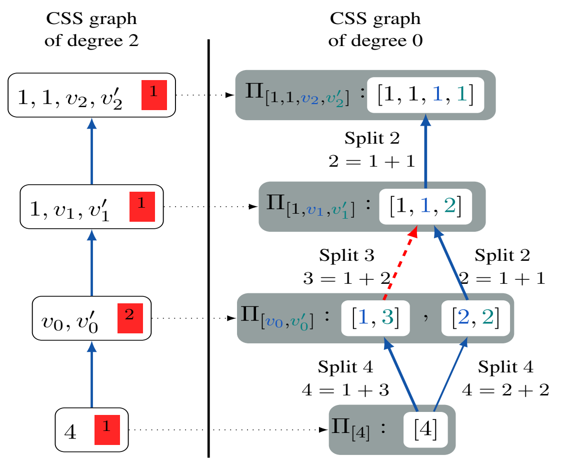

Figure 3: A four-agent CSS graph with two variables that compacts 4 levels of nodes. The left part of this figure is a CSS graph of degree 2 unfolded in the right part into a CSS graph of degree 0, which is equivalent to the representation presented in [Rahwan *et al.*, 2009].

Theorem 3. *By considering only a subset of the edges in a CSS graph, if there is a path in the graph between every node and the bottom node, the optimal solution will be found.*

Proof. An edge in the CSS graph represents a splitting of a coalition into two coalitions. A path between the bottom node and another node $\mathcal{N}$ represents the possibility of obtaining any coalition structure of that node through a series of recursive splittings of the coalitions starting from the grand coalition. Thus, if there is a path between the bottom node and $\mathcal{N}$, all the coalition structures of $\mathcal{N}$ will be generated, and hence $\mathcal{N}$ will be fully searched. Therefore, if there are paths between the bottom node and all other nodes, the optimal solution will be found. $\square$

Now, as there are several sets of edges that we can consider to connect all the nodes to the bottom node, and since an edge reflects a splitting of a coalition size, we can choose which coalition sizes (and hence which coalitions) to split to achieve this. For instance, in the graph of Figure 3, removing the red edge still keeps all the nodes connected to the bottom node. This means that, by only splitting and evaluating coalitions of sizes 2 and 4, the optimal solution can still be found. Thus, to find the optimal solution, several sets of coalition sizes can be considered ($\{2, 4\}$ and $\{2, 3, 4\}$ in our example). However, each set of sizes has a different run time, which corresponds to the time it takes to evaluate all the coalitions of those sizes. Thus there is an optimal set of sizes using which we can achieve the shortest run time.

To find this optimal set of sizes for a CSG problem size n, our preprocessor starts by determining the estimated run time for evaluating the coalitions of each size (i.e., the cost of considering each size). It corresponds to the evaluation time of all the different ways of splitting the coalitions of each size. The estimated time of a splitting is the computational cost associated with this hardware operation which is a fixed value like any other operation, such as an addition or a subtraction. Hence, the estimated time for a set of sizes is the sum of the evaluation times of the coalitions of the size set. Then, it computes the optimal set of sizes that minimizes the total run time by considering all possible sets of sizes. For each set, it verifies that all nodes in the CSS graph are connected to the bottom node by only considering the edges that result from splitting the sizes in the set. If a set satisfies this condition, the preprocessor compares its run time to the current best set and updates the best set if necessary. Once all possible sets have been considered, the preprocessor has identified the optimal set of sizes. For example, with $n = 4$, the preprocessor finds $2, 4$ and $2, 3, 4$ as the only sets that guarantee to find an optimal solution, and $2, 4$ as the optimal one. With $n = 10$, it finds $2, 4, 6, 8, 10$. Note that the preprocessor is executed independently of the process of finding an optimal solution, as its output is unique and serves as a configuration for the RDP algorithm. Therefore, it is executed only once to obtain the optimal set of sizes and not to solve CSG problems. The pseudocode of the algorithm used by the preprocessor is in the appendix.

Now, the search process in RDP is carried out on the coalitions of sizes that belong to the optimal set of sizes $\mathcal{S} = \{s_1, s_2, ..., s_k\}$ obtained by the preprocessor. RDP starts by evaluating the coalitions of size s_1, and for every coalition $\mathcal{C}_{s_1}$ of size s_1 that the algorithm encounters, it evaluates all possible ways of splitting $\mathcal{C}_{s_1}$ into two coalitions and checks whether it is beneficial to split $\mathcal{C}_{s_1}$ or not. The best results are then stored in $\mathcal{P}_t(\mathcal{C}_{s_1})$ and $\mathcal{V}_t(\mathcal{C}_{s_1})$. Similarly, RDP evaluates the coalitions of size s_2, and so on. This process is repeated until the last size s_k. This way, by the time RDP evaluates the coalitions of size s_k, which is always equal to n (because we always need to compute the best partition for the coalition of size n to find the optimal solution), the best partition of each coalition is obtained, and also that of the grand coalition that corresponds to the optimal solution. The pseudocode of RDP is in the appendix.

4.3 Hybridization: ELIXIR

In ELIXIR, we combine the CSSA and RDP algorithms to assist one another during the search process. The two algorithms work in parallel and share a CSS graph. CSSA starts by sorting the subspace groups and the subspaces within each group by their upper bounds and begins the search with the group that has the highest upper bound. As it explores the subspaces, it prunes out those that do not have a better upper bound than the best solution found so far. At the same time, RDP evaluates the coalitions starting with the smallest ones. The two algorithms share a CSS graph and work together as follows: once RDP finishes evaluating all the coalitions of size s_i, the edges resulting from splitting s_i into two are added to the CSS graph. The following theorem shows the relationship between the two algorithms (the proof is similar to that of Theorem 3, and can be found in the appendix):

Theorem 4. *If any node in the CSS graph is connected to the bottom node with a series of edges, it is fully searched by RDP.*

The newly added edges by RDP make certain subspaces reachable from the bottom node, and hence, they are fully searched by RDP, which finds the optimal coalition structure among them. This means that CSSA does not need to search them anymore, thereby pruning them out. As the size s_i increases, more subspaces become reachable from the bottom node and are pruned out. Once all subspaces have been searched or pruned out by either algorithm, ELIXIR finishes and returns the optimal solution.

Updating the Group Upper Bounds

We have shown that RDP can help CSSA prune out subspaces that are already searched by RDP. We now show how this can also help improve the CSSA search process. As stated earlier, CSSA computes the group upper bound of each subspace group and then searches the groups considering this bound. The group upper bound of a subspace group corresponds to the upper bound of the highest upper bound subspace of the group. In case the highest upper bound subspace of a group is pruned by RDP and the group still has a chance of containing an optimal solution, the group upper bound of this subspace group is recalculated as the upper bound of the subspace with the highest upper bound among those remaining. Hence, the bound is lowered. Thus, the order of the groups that are yet to be searched can change during the execution, thus favoring the subspaces with the highest chances of containing an optimal solution.

The pseudocode of ELIXIR is in the appendix.

Theorem 5. *ELIXIR always finds an optimal solution.*

Proof. Each node in the CSS graph is searched using ELIXIR by CSSA or RDP. A node represents a subspace of solutions, which contains a number of coalition structures that match the parts of the subspace. Fix the node P that contains the optimal solution. The optimal solution is found if RDP reaches P from the bottom node or if CSSA finishes searching the subspace group that contains P. ELIXIR finishes when all subspaces are searched by RDP or CSSA. Hence, the subspace P that contains the optimal solution is always searched by ELIXIR using RDP or CSSA. □

In Theorems 6 and 7, we show the time complexity of the ELIXIR algorithm and the preprocessor. Recall that the preprocessor is run only once for each problem size n to configure the RDP algorithm of ELIXIR. Hence, ELIXIR is the only algorithm we run to search for the optimal solutions. The proof of theorem 7 can be found in the appendix.

Theorem 6. *Given n agents, ELIXIR runs in $O(3^n)$ time.*

Proof. In ELIXIR, CSSA and RDP run in parallel in two processes, and when any of them finds the optimal solution, ELIXIR terminates. The worst-case run time of dynamic programming, and hence of RDP, on this problem is $\mathcal{O}(3^n)$ [Yeh, 1986]. CSSA, which requires in the worst case searching all coalition structures, has run time $\mathcal{O}(n^n)$. Thus, the time complexity of ELIXIR is $\min(\mathcal{O}(3^n), \mathcal{O}(n^n)) = \mathcal{O}(3^n)$. □

Theorem 7. *For all CSG problem sizes n, the preprocessor of RDP computes the optimal size set in $\mathcal{O}(2^n \times n^2 \times \frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n})$.*

5 Experiments

The main goals of our experiments are to investigate how different input sizes and value distributions affect our search method, and how our algorithm compares to the prior state of the art. In accordance with established practices in coalition structure generation, we benchmark on the following coalition value distributions: Normal [Rahwan *et al.*, 2007], Modified Normal [Rahwan *et al.*, 2012], Uniform [Larson and

Sandholm, 2000], Modified Uniform [Service and Adams, 2010], Beta, Exponential, Gamma [Michalak *et al.*, 2016], Pascal and Weibull [Changder *et al.*, 2020]. We implemented ELIXIR in Java and used the codes provided by the authors of ODP-IP [Michalak *et al.*, 2016] and BOSS [Changder *et al.*, 2021], which are also written in Java. The experiments were conducted on an Intel Xeon 2.30GHz E5-2650 CPU with 256GB of RAM.

ELIXIR uses a hyperparameter for the number of variables that it considers in CSSA. This hyperparameter is optimized using 9000 problem instances by varying the number of variables. Our hyperparameter search and the final hyperparameter are presented in the appendix.

Figure 4 shows how our algorithm performs against the prior state-of-the-art algorithms ODP-IP and BOSS. The algorithms behave differently depending on the value distributions. Each result was produced by computing the average result from 50 generated problem instances per value distribution. Overall, ELIXIR outperforms ODP-IP and BOSS on some value distributions (Normal, Uniform, Modified Uniform, Modified Normal, Weibull and Pascal) and on par on the other value distributions (Gamma and Exponential). A notable exception is the Beta distribution, for which there is essentially a perfect tie. There are also some points in Figure 4 where ODP-IP and BOSS outperform ELIXIR (e.g., for 23 agents on the Normal distribution). A potential reason for this is that the sets of sizes that RDP uses do not help the algorithm prune the subspaces faster than the other algorithms for these points. An important observation is that there are only three points in Figure 4 where the other algorithms are faster than ELIXIR and there is no distribution for which ELIXIR performs meaningfully worse than the other algorithms. This indicates that our algorithm is suitable for many different problem instance distributions, including easy ones such as Pascal and Modified Uniform, and hard ones such as Normal and Gamma.

To see how the preprocessing phase affects the performance of our algorithm, we benchmark RDP against the dynamic programming algorithm IDP [Rahwan and Jennings, 2008] used in both ODP-IP and BOSS. In these experiments, we show the run time of RDP and IDP, which is not influenced by the value distribution, but only by the number of agents. We also show the time gain achieved by RDP compared to IDP. Table 1 clearly shows that RDP provides optimal solutions faster than IDP. For example, after 388 seconds, for 24 agents, RDP returns optimal solutions roughly 139 seconds faster than IDP.

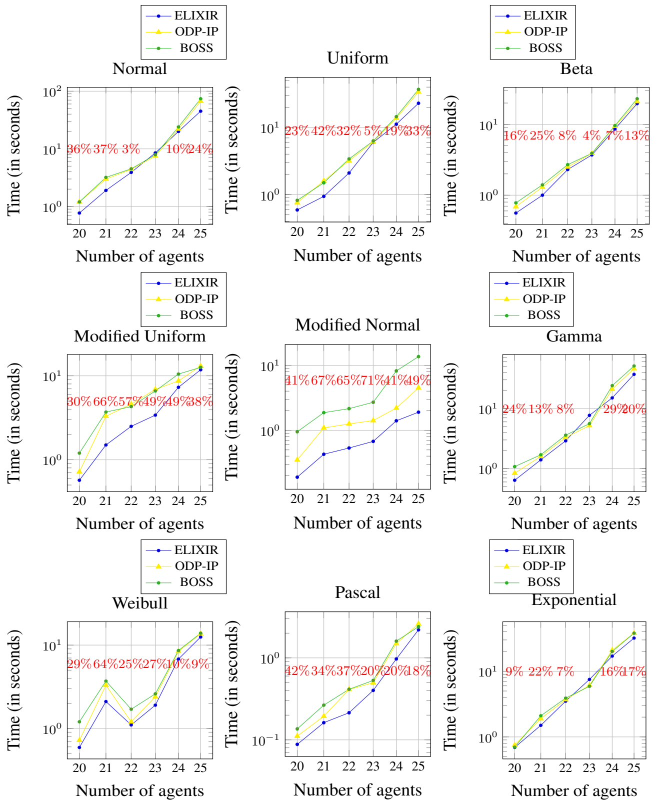

Figure 4: Run time of ELIXIR, ODP-IP, and BOSS in log scale. The percentages represent the time gain achieved by ELIXIR compared to the best performing algorithm between ODP-IP and BOSS. They are computed by dividing the run time difference by the run time of best performing algorithm between ODP-IP and BOSS.

Number of Agents	**Execution Time**		**Time Difference**
	RDP	**IDP**	
20	3.2	3.7	0.5
21	12.8	15.9	3.1
22	22	24.7	2.7
23	108	131	23
24	368	507	139
25	847	887	40
26	3515	3659	144
27	5612	7078	1466

Table 1: Run time in seconds of RDP compared to the run time of IDP. The 'Time Difference' column shows the time gain achieved by RDP compared to IDP.

6 Conclusions

In this paper, we presented a new method for optimally solving the coalition structure generation problem—a well-studied challenge in AI. We presented a novel compact representation of the search space that gathers solution subspaces in a new way and we developed a new method for finding the optimal solution to the problem based on this representation. The algorithm combines two new techniques: one that uses the new representation along with a branch-and-bound technique, and one that uses a preprocessor to select the best sets of coalitions to evaluate. We analyzed how parts of the solution space can be gathered into groups to avoid their redundant re-evaluation. We also investigated the computational gain achieved by avoiding this redundancy. We conducted experiments on a variety of common benchmark value distributions. The experiments show that our algorithm performs better than, or on par with, the prior state-of-the-art algorithms on all the value distributions and all numbers of agents.

References

[Changder *et al.*, 2020] Narayan Changder, Samir Aknine, Sarvapali D Ramchurn, and Animesh Dutta. Odss: Efficient hybridization for optimal coalition structure generation. In *Proc. of AAAI*, pages 7079–7086, 2020.

[Changder *et al.*, 2021] Narayan Changder, Samir Aknine, Sarvapali D. Ramchurn, and Animesh Dutta. Boss: A bidirectional search technique for optimal coalition structure generation with minimal overlapping (student abstract). In *Proc. of AAAI*, volume 35, pages 15765–15766, May 2021.

[Dang and Jennings, 2004] Viet Dung Dang and Nicholas R Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 564–571. IEEE Computer Society, 2004.

[Dang *et al.*, 2006] Viet Dung Dang, Rajdeep K Dash, Alex Rogers, and Nicholas R Jennings. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In *Proc. of AAAI*, volume 6, pages 635–640, 2006.

[Di Mauro *et al.*, 2010] Nicola Di Mauro, Teresa MA Basile, Stefano Ferilli, and Floriana Esposito. Coalition structure generation with grasp. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 111–120. Springer, 2010.

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

[Larson and Sandholm, 2000] Kate S Larson and Tuomas W Sandholm. Anytime coalition structure generation: an average case study. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1):23–42, 2000.

[Michalak *et al.*, 2016] Tomasz Michalak, Talal Rahwan, Edith Elkind, Michael Wooldridge, and Nicholas R Jennings. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230:14–50, 2016.

[Rahwan and Jennings, 2008] Talal Rahwan and Nicholas R Jennings. An improved dynamic programming algorithm for coalition structure generation. In *Proc. of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[Rahwan *et al.*, 2007] Talal Rahwan, Sarvapali D Ramchurn, Viet Dung Dang, and Nicholas R Jennings. Near-optimal anytime coalition structure generation. In *Proc. of IJCAI*, volume 7, pages 2365–2371, 2007.

[Rahwan *et al.*, 2009] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, and Andrea Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of artificial intelligence research*, 34:521–567, 2009.

[Rahwan *et al.*, 2012] Talal Rahwan, Tomasz Michalak, and Nicholas R Jennings. A hybrid algorithm for coalition structure generation. In *Proc. of AAAI*, pages 1443–1449, 2012.

[Sandholm and Lesser, 1997] Tuomas Sandholm and Victor R Lesser. Coalitions among computationally bounded agents. *Artificial intelligence*, 94(1):99–138, 1997.

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

[Sen and Dutta, 2000] Sandip Sen and Partha Sarathi Dutta. Searching for optimal coalition structures. In *Proceedings Fourth International Conference on MultiAgent Systems*, pages 287–292. IEEE, 2000.

[Service and Adams, 2010] Travis Service and Julie Adams. Approximate coalition structure generation. In *Proc. of AAAI*, pages 854–859, 2010.

[Taguelmimt *et al.*, 2021] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Code-based algorithm for coalition structure generation. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1075–1082, 2021.

[Taguelmimt *et al.*, 2022a] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Pics: Parallel index-based search algorithm for coalition structure generation. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 739–746, 2022.

[Taguelmimt *et al.*, 2022b] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Subspace-focused search method for optimal coalition structure generation. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1435–1440, 2022.

[Wilf, 2000] Herbert Wilf. Lectures on integer partitions. University of Pennsylvania, 09 2000.

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