Code-based Algorithm for Coalition Structure Generation

Paper ID: 1882

Abstract

Finding the optimal coalition structure is an NP-complete problem that is computationally challenging even under quite restrictive assumptions. In this paper, we propose a new algorithm for the Coalition Structure Generation (CSG) problem, which provides good enough quality solutions and that can be run with hundreds of agents. The Agent Code-based coalition structure Search algorithm (ACS) uses a novel representation of the search space of coalition structures and a new code-based search technique. We devise an effective heuristic search method to efficiently explore the space of coalition structures using our code-based technique. Results show that our method outperforms existing state-of-the-art algorithms by multiple orders of magnitude while providing high quality solutions.

1 Introduction

Usefulness of coalition formation has been proved in many applications such as e-commerce [Tsvetovat and Sycara, 2000], distributed sensor networks [Dang *et al.*, 2006], etc. In this paper, we propose a new algorithm for the coalition structure generation problem that can be run with hundreds of agents. The fastest exact algorithms to date for solving the coalition structure generation problem are hybrid solutions called ODP-IP [Michalak *et al.*, 2016] and ODSS [Changder *et al.*, 2020] that combine IDP [Rahwan and Jennings, 2008] and IP [Rahwan *et al.*, 2009]. ODP-IP and ODSS are very efficient for solving some problem instances when the number of agents is fewer than 30. In cases where there is enough time to perform the calculations required to find the exact result, exact algorithms are still practical. However, in some cases, when agents do not have sufficient time, an approach that gives good enough quality solutions, within a reasonable time, is more valuable [Wu and Ramchurn, 2020; Farinelli *et al.*, 2013]. Moreover, exact algorithms, such as ODP-IP and ODSS, are difficult to use on large-scale instances with hundreds of agents. [Wu and Ramchurn, 2020] showed that their proposed algorithm CSG-UCT outperforms C-Link [Farinelli *et al.*, 2013]. Subsequently, we compare the ACS algorithm with CSG-UCT and show their results. The contributions of this paper are:

- We propose a novel search method of coalition structures. It consists of an innovative heuristic algorithm ACS and a novel representation of the search space.
- ACS can be run with hundreds of agents, where the limit of exact algorithms is a few dozens of agents [1]. We show that our method is faster than the state-of-the-art.

Section 2 of this paper presents our novel representation of the search space, while Section 3 details our algorithm (ACS). Section 4 analyzes our algorithm. Section 5 provides the results of the empirical evaluation. Finally, Section 6 concludes the paper.

2 Novel Search Space Representation

The coalition structure generation problem is defined by a set $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ of n agents where the efficiency of any subset of $\mathcal{A}$ is determined by a characteristic function v that assigns a real value to every subset of $\mathcal{A}$ called a coalition. A coalition structure $\mathcal{CS}$ is a partition of the set of agents $\mathcal{A}$ into disjoint coalitions $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$ and $\forall \mathcal{C}_i \in \mathcal{CS}$, $\mathcal{C}_i \neq \emptyset$ and $\mathcal{C}_i \in p(\mathcal{A})$, a set of all the possible coalitions in $\mathcal{A}$. $\mathcal{CS}$ satisfies the following constraints: $\bigcup_{j=1}^{k} \mathcal{C}_i = \mathcal{A}$ and for all $i, j \in \{1, 2, ..., k\}$ where $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$. The set of all coalition structures is $\Pi(\mathcal{A})$. The value of a coalition structure $\mathcal{CS}$ is assessed as $V(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$. The coalition structure generation problem takes as input a set of agents $\mathcal{A}$ and aims at finding the optimal coalition structure $\mathcal{CS}^* \in \Pi(\mathcal{A})$, where $\mathcal{CS}^* = \arg\max_{\mathcal{CS} \in \Pi(\mathcal{A})} V(\mathcal{CS})$.

Before going any further, let us recall the principle of the integer partition (IP) graph [Rahwan *et al.*, 2007], which divides the space of all the coalition structures into subspaces that are each represented by an integer partition of n. For instance, for $n = 4$ agents, the set of integer partitions is: $\{[4], [1, 3], [2, 2], [1, 1, 2], [1, 1, 1, 1]\}$. Each partition of n sums to n. The size of a partition $|\mathcal{P}|$ denotes the number of parts it contains. For example, the size of the partition $[1, 1, 2]$ is 3 because it contains 3 parts. In the IP graph, each partition of n is represented by a node, where two adjacent nodes are connected if and only if the partition in level l can be reached

[1] The code of ACS and the datasets are provided in the supplementary material.

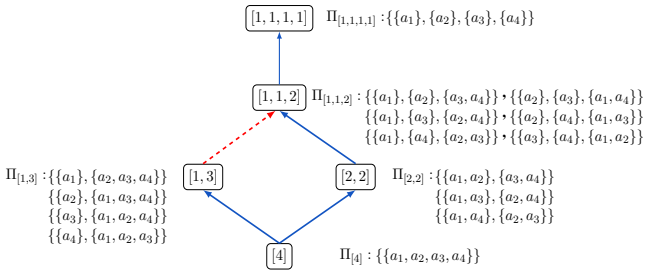

Figure 1: A four-agent example of the integer partition graph. The bottom node $[4]$ represents the coalition structure formed by the grand coalition $\{a_1, a_2, a_3, a_4\}$. The top node $[1, 1, 1, 1]$ represents the coalition structure formed by all the singleton coalitions $\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}$.

from another partition in level $l-1$ by splitting only an integer. In each level $l \in \{1, 2, .., n\}$, each node represents coalition structures that contain exactly $l \in \{1, 2, .., n\}$ coalitions. For example, in Figure 1, the node $[1, 1, 2]$ represents all the coalition structures that contain two coalitions of size 1 and one coalition of size 2.

In our method, we codify each coalition $\mathcal{C}_i$ with a code (i.e., an index-based code). As a result, each agent of a given coalition $\mathcal{C}_i$ will be encoded with a code i, which corresponds to the index of the coalition to which it belongs. By doing so, each coalition structure is defined as a vector of size n where n represents the number of agents. Each position p in the vector refers to the agent $p_{/p=1..n}$, while the code in position p of the vector identifies the coalition to which the agent p belongs. Hence, a coalition structure will be encoded by a set of positive numbers. Let us consider the following example. Let $\mathcal{A}$ be a set of $n = 10$ agents. Consider the coalition structure $\mathcal{CS} = \{\{a_1\}, \{a_2, a_3, a_4\}, \{a_5, a_6, a_7, a_8, a_9, a_{10}\}\}$ containing three coalitions: $\mathcal{C}_0 = \{a_1\}$, $\mathcal{C}_1 = \{a_2, a_3, a_4\}$ and $\mathcal{C}_2 = \{a_5, a_6, a_7, a_8, a_9, a_{10}\}$. The coalitions $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_2$ are encoded with codes 0, 1 and 2, respectively. $\mathcal{CS}$ will be encoded by the vector of codes $[x_1\ x_2\ ...\ x_{10}]$, where $x_{i/i\in\{1,..,10\}} = 0$ iff $a_i \in \mathcal{C}_0$, $x_{i/i\in\{1,..,10\}} = 1$ iff $a_i \in \mathcal{C}_1$ and, $x_{i/i\in\{1,..,10\}} = 2$ iff $a_i \in \mathcal{C}_2$. The coalition structure $\mathcal{CS}$ is then encoded using this representation with the vector of codes [0 1 1 1 2 2 2 2 2 2], where the size of the vector equals n. Agent a_1 is in the coalition $\mathcal{C}_0$, which is why the number associated with a_1 in $\mathcal{CS}$ is 0. Agents a_2, a_3 and a_4 are in $\mathcal{C}_1$ and hence, their number is 1. The rest of the agents form $\mathcal{C}_2$.

Any permutation of these numbers provides a different coalition structure, as shown in Figure 2. In this example, the first coalition is formed by agent a_3, the second coalition by a_2, a_4, a_6, while the last coalition is formed by $a_1, a_5, a_7, a_8, a_9, a_{10}$. Thus, the coalition structure is represented with the vector of codes [2 1 0 1 2 1 2 2 2 2].

Generating all the combinations of these numbers guarantees exploration of all the coalition structures represented by each node. Let us now generalize this representation for the entire integer partition graph. For each node in level l, we have l coalitions and we then codify each coalition $\mathcal{C}_i$ with a code (i.e., a representative number) $i \in \mathcal{N} = \{0, 1, .., l-1\}$. Then, to represent with a vector of codes each coalition structure $\mathcal{CS}$ of this node, each coalition $\mathcal{C}_i$ of size k in $\mathcal{CS}$ is depicted k times by the code i. For example, the node $[1, 3, 6]$, to which the coalition structure presented in Figure 2 belongs, contains three coalitions where $|\mathcal{C}_0| = 1$, $|\mathcal{C}_1| = 3$ and $|\mathcal{C}_2| = 6$. Each vector representing a coalition structure of this node will contain one 0 for the coalition of size 1, three 1 for the coalition of size 3, and six 2 for the coalition of size 6. Figure 3 shows the vectors which represent all the coalition structures for a set of $n = 4$ agents. Each vector in Figure 3 corresponds to a coalition structure in Figure 1. This new representation is based on coalition indexes, unlike the original version in [Rahwan *et al.*, 2007]. This makes it possible to work with integers while searching in the space of coalition structures.

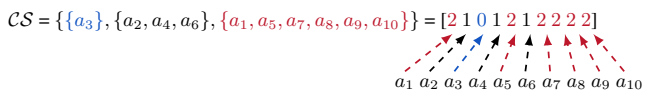

Figure 2: Representation of the coalition structure $\mathcal{CS} = \{\{a_3\}, \{a_2, a_4, a_6\}, \{a_1, a_5, a_7, a_8, a_9, a_{10}\}\}$ in the new method with the vector [2 1 0 1 2 1 2 2 2 2]. Agent a_3 is encoded with the code 0 because $a_3 \in \mathcal{C}_0$. a_1, a_5, a_7, a_8, a_9 and a_{10} are encoded with the code 2 representing the coalition $\mathcal{C}_2$. In the IP graph of 10 agents, this coalition structure $\mathcal{CS}$ is represented in the node $[1, 3, 6]$.

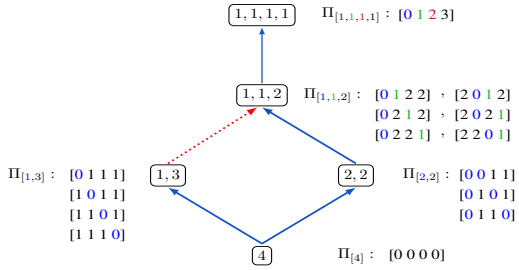

Figure 3: An example of the IP graph with four agents. The node $[1, 3]$ holds 4 coalition structures, each formed by two coalitions of sizes 1 and 3. Two numbers codify these coalitions. 0 codifies the coalition of size 1, and 1 codifies the coalition of size 3. Consequently, all the coalition structures are represented with the vectors [0 1 1 1], [1 0 1 1], [1 1 0 1] and [1 1 1 0].

3 Code-based CSG Algorithm

ACS partially enumerates the coalition structures of each node of the integer partition graph. Specifically, we start by generating all the integer partitions of n, then ACS explores each node of this graph. Once all the possible partitions of the integer n are generated, the ACS algorithm proceeds in two main phases. The first phase involves generating, for each possible partition, the codes that represent any coalition structure that belongs to it. These codes are then used in the second phase, which involves generating the different permutations of the codes obtained at the end of phase 1. These permutations are then used to compute the coalition structures.

3.1 ACS Algorithm Phases

Phase 1: Code Generation for Each Partition

As shown in Figure 2, the set of codes that we use to compute the combinations contains repeated numbers. Thus, using a naive method for calculating combinations is time-consuming, especially for a large number of agents. Therefore, ACS partially generates the permutations. Consider the node $[1, 3, 6]$ (see Figure 2). To enumerate the coalition structures represented by this node, we use the codes $0, 1$ and 2. Rather than computing all the combinations of the 10 codes (that represent the coalition to which each agent of the coalition structure belongs), ACS calculates the combinations of the three codes $\{0, 1, 2\}$ that represent the coalitions. We then obtain these combinations: $\vartheta = \{\{0, 1, 2\}, \{0, 2, 1\}, \{1, 0, 2\}, \{1, 2, 0\}, \{2, 0, 1\}, \{2, 1, 0\}\}$. For each combination in ϑ, ACS calculates a first coalition structure of the combination called Initialization. Each code in the combination will then be repeated as many times as the size of the coalition it represents. This means that in each Initialization, the agents are assigned to the coalitions by respecting the order of the coalitions. For example, the Initialization of $\{0, 1, 2\}$ is [0 1 1 1 2 2 2 2 2 2] because the coalition encoded with 0 is of size 1, the coalition encoded with 1 is of size 3, and the coalition encoded with 2 is of size 6. Likewise, the Initialization of $\{0, 2, 1\}$ is [0 2 2 2 2 2 2 1 1 1] because we start by completing the coalition encoded with 0 by one agent (which corresponds to its size of 1), then we complete the coalition encoded with 2 by six agents, and finally, the coalition encoded with 1 by the last three agents. Similarly, the Initialization of $\{2, 0, 1\}$ is [2 2 2 2 2 2 0 1 1 1]. The different Initializations thus obtained will then be used in the second phase to generate the different permutations.

Phase 2: Coalition Structure Generation

The purpose of this second phase is to generate the permutations of the codes composing the vectors obtained in phase 1. Each newly generated vector of codes, after each permutation, will represent a different coalition structure. For each Initialization vector, ACS starts with the first agent and permutes its code with each of the codes of the other agents. Then, ACS moves to the next agent and applies the same permutation operations to its code. This process is then iterated until ACS reaches the last code of the vector. A four-agent example is shown in Figure 4. This example concerns the node $[1, 1, 2]$ of level 3 (see Figure 3) that represents the coalition structures with two coalitions of size 1 and one coalition of size 2. The first coalition structure that ACS assigns to this node is represented by the vector [0 1 2 2] (see line 1 of Figure 4), which is the Initialization of the combination $\{0, 1, 2\}$ (we have one 0 for the coalition of size 1 and one 1 for the second coalition of size 1 followed up by two codes 2 for the coalition of size 2). In the second line of Figure 5, we swap the first two codes of the vector [0 1 2 2] and get the vector [1 0 2 2]. This vector represents the coalition structure $\{\{a_2\}, \{a_1\}, \{a_3, a_4\}\}$. We do the same for lines 3 and 4. In line 5, we move to the second agent and swap the second code with the third one to get the coalition structure $\{\{a_1\}, \{a_3\}, \{a_2, a_4\}\}$ represented by the vector [0 2 1 2]. Line 6 reflects the last possible permutation of this Initialization. Notice that with 4 agents, ACS generates all possible coalition structures, and hence the optimal one. However, for sets with more than 6 agents, the two phases of ACS skip some coalition structures because they may not be generated with this method. For example, the coalition structure $\mathcal{CS}$ in Figure 2 (with 10 agents) is skipped because no permutation of the Initializations (e.g. [0 1 1 1 2 2 2 2 2 2] or [2 2 2 2 2 2 0 1 1 1]) yields it and hence, ACS does not guarantee finding the optimal solution.

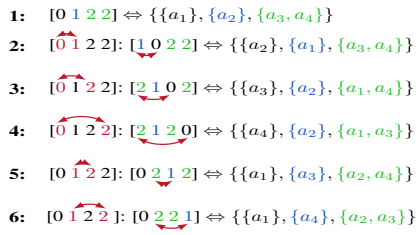

Figure 4: An illustration of ACS when searching the node $[1, 1, 2]$ of the IP graph in Figure 3 given the Initialization [0 1 2 2].

3.2 ACS Algorithm Optimizations

Pruning Combinations

The two phases of ACS may generate redundant coalition structures such as $\{\{a_1, a_3\}, \{a_2, a_4\}\}$ and $\{\{a_2, a_4\}, \{a_1, a_3\}\}$. These redundancies arise when repeated coalition sizes appear in a node (e.g. $[2, 2]$ or $[1, 1, 2]$). In such cases, ACS considers only Initializations that result from combinations where the order of the corresponding coalition sizes is unique. For example, with the node $[2, 2]$ (see Figure 5) where $|\mathcal{C}_0| = |\mathcal{C}_1| = 2$, the two combinations $\{0, 1\}$ (initialized to [0 0 1 1]) and $\{1, 0\}$ (initialized to [1 1 0 0]) yield the same order of coalition sizes: $[|\mathcal{C}_0|, |\mathcal{C}_1|] = [|\mathcal{C}_1|, |\mathcal{C}_0|] = [2, 2]$ and hence, ACS only considers one of these combinations.

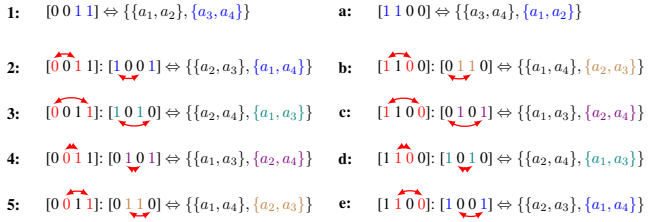

Figure 5: An example of repeated generation of coalition structures when searching the node [2,2] of the IP graph. ACS finds that both Initializations $\{0, 1\}$ and $\{1, 0\}$ compute the same coalition structures (each combination orders the coalition codes differently but the size of the corresponding coalitions remains the same). Thus, ACS computes the permutations for only one Initialization.

Limiting the Permutations

In phase 1, generation of the code combinations applies only for nodes in levels 1 to 5 of the IP graph. For nodes on other levels, ACS generates only two Initializations without computing all the combinations. These Initializations are those that order the indexes in an ascending order or in a descending order, thus differentiating these two Initializations, i.e. the

order of coalition sizes in the two combinations would never be the same (except when all the coalitions have the same size). Limiting permutations reduces computation time, as will be shown in Section 5.

Bounding the Search

In the IP graph, some nodes have no chance of improving the solution quality. To identify such nodes, ACS computes before processing a node, its upper bound, i.e. the highest value a coalition structure of this node can possibly reach. This upper bound $\mathcal{UB}$ of a node $\mathcal{N}$ is calculated as $\mathcal{UB}(\mathcal{N}) = \sum_{i \in Integers(\mathcal{N})} Max_i$, where Max_i is the maximum value a coalition of size i can take and $Integers(\mathcal{N})$ is the set of integers that form the integer partition of the node $\mathcal{N}$. For instance, for $\mathcal{N} = [1, 3, 6]$, $Integers(\mathcal{N}) = \{1, 3, 6\}$ and $\mathcal{UB}(\mathcal{N}) = \sum_{i \in \{1,3,6\}} Max_i = Max_1 + Max_3 + Max_6$. By comparing the upper bounds of the different nodes, ACS eliminates the nodes that do not have a better upper bound than the last best solution found.

Algorithm 1 details the coalition structure generation. ACS iterates over the nodes of the integer partition graph that covers all the partitions of integer n, given n agents. Algorithm 1 starts by storing the value of each coalition (see the loop in lines 1-5). The optimal coalition structure is first set to be the grand coalition, before its value improves during the search as described in lines 9-30. In more detail, ACS processes one node of the IP graph at a time as follows: after generating the set of codes for the given node (line 11), it computes the combinations and then filters them (see lines 12-17) to eliminate those that are redundant. For each combination, ACS generates the Initialization vector (line 19) and computes the permutations (line 20). Each permutation, which represents a new coalition structure, is then considered to, possibly, determine the new best one, as described by lines 21-27.

Algorithm 1: The ACS algorithm

Input: A Set of agents $\mathcal{A}$ of size n.
Output: The best structure $\mathcal{CS}^+$ and its value $\mathcal{V}^+$.

1 **for** $i = 1$ *to* n **do**
2 **for** $\mathcal{C} \subseteq \mathcal{A}$, *where* $|\mathcal{C}| = i$ **do**
3 $\mathcal{V}(\mathcal{C}) \leftarrow v(\mathcal{C})$
4 **end**
5 **end**
6 Generate partitions of n and store them in φ
7 $\mathcal{CS}^+ \leftarrow \mathcal{A}$ ▷ *initialization with the grand coalition*
8 $\mathcal{V}^+ \leftarrow \mathcal{V}(\mathcal{A})$
9 **for** $l = 1$ *to* n **do**
10 **for** $p \in \varphi$, *where* $|p| = l$ **do**
11 $\mathcal{N} \leftarrow \{0, 1, .., l-1\}$ ▷ $\mathcal{N}$ *is the set of codes corresponding to the coalitions in partition* p
12 **if** $l < 6$ **then**
13 Generate all combinations of $\mathcal{N}$ and store them in the set ϑ
14 **else**
15 Generate 2 combinations of $\mathcal{N}$ and store them in the set ϑ ▷ *these two combinations are detailed in Section 3.2*
16 **end**
17 $\vartheta \leftarrow$ Combination_pruning(ϑ) ▷ *call of Algorithm 3 detailed in the Appendix*
18 **foreach** $\varsigma \in \vartheta$ **do**
19 Init $\leftarrow$ Constructing_an_Initialization(ς) ▷ *call of the first Algorithm in Appendix*
20 $\mathcal{PM} \leftarrow$ permuting_the_codes(Init) ▷ *call of Algorithm 2 detailed in the Appendix*, $\mathcal{PM}$ *is a set of code vectors*
21 **foreach** $\mathcal{CS} \in \mathcal{PM}$ **do** ▷ *each permutation represents a coalition structure* $\mathcal{CS}$
22 $\mathcal{V}(\mathcal{CS}) = \sum_{j=0}^{l-1} \mathcal{V}(\mathcal{C}_j)$, where $\mathcal{C}_j \in \mathcal{CS}$
23 **if** $\mathcal{V}(\mathcal{CS}) > \mathcal{V}^+$ **then**
24 $\mathcal{V}^+ \leftarrow \mathcal{V}(\mathcal{CS})$
25 $\mathcal{CS}^+ \leftarrow \mathcal{CS}$
26 **end**
27 **end**
28 **end**
29 **end**
30 **end**
31 Return $\mathcal{CS}^+$, $\mathcal{V}^+$

4 Analysis of ACS

Theorem 1. *The ACS algorithm is anytime.*

Proof. To compute the coalition structures, ACS starts with an initial coalition structure and then improves it over time. When execution is stopped, it can still return the currently best solution that is better than the previous best ones. Therefore, ACS is anytime. □

Theorem 2. *Given any partition $\mathcal{P}$, ACS guarantees that for each agent a_i and for each coalition size k in $\mathcal{P}$, ACS examines at least $2 \times k$ coalition structures in which a_i appears in one coalition of size k.*

Proof. For each node $\mathcal{P}$ in the IP graph, ACS computes the permutations for at least two Initializations. Thus, it suffices to prove that for each Initialization, for each agent a_i and for each coalition size k in $\mathcal{P}$, there are at least k coalition structures in which a_i appears in a coalition of size k. Consider any coalition $\mathcal{C}$ of size k. The result is immediate when the agent belongs to $\mathcal{C}$. Thus, for the remainder of the proof, we assume that the agent does not belong to $\mathcal{C}$. In the previous section, we stated that the code of any agent a_i is permuted with the code of all the other agents and hence with each agent in the coalition $\mathcal{C}$. Thus, any agent a_i will have the code that represents the coalition $\mathcal{C}$, k times, which proves that a_i will belong to a coalition $\mathcal{C}$ of size k at least k times. □

Theorem 3. *The time complexity of ACS is* $\mathcal{O}(n^2 \times \frac{e^{\pi\sqrt{2n/3}}}{n})$.

Proof. For each node in level $l < 6$ of the IP graph, ACS computes the permutations on $l!$ Initializations (line 13 in Algorithm 1). On the other hand, for each node in level $l \geq 6$, ACS considers only 2 Initializations (line 15 in Algorithm 1). We denote by $p(n, l)$ the number of partitions (i.e. nodes) of

n with l parts. The number of permutations for a single Initialization is $\mathcal{O}(\frac{n^2}{2}) = \mathcal{O}(n^2)$. With this, we can compute the time complexity of ACS. Total operations performed $\mathcal{T}(n)$ is as follows:

$\mathcal{T}(n) = \sum_{l=2}^{n-1} \mathcal{O}(n^2) \times i(l) \times p(n,l)$, where $\mathcal{O}(n^2)$ is the number of permutations, $i(l)$ is the number of Initializations and $p(n,l)$ is the number of nodes in level l. We do not consider the levels 1 and n because they only contain one coalition structure and hence no permutation is needed. For $l < 6$, $i(l) = l!$ while $i(l) = 2$ for $l \geq 6$.

$\mathcal{T}(n) = \sum_{l=2}^{5} \mathcal{O}(n^2) \times l! \times p(n,l) + \sum_{l=6}^{n-1} \mathcal{O}(n^2) \times 2 \times p(n,l)$

$\mathcal{T}(n) = \mathcal{O}(n^2) \times \sum_{l=2}^{5} l! \times p(n,l) + \mathcal{O}(n^2) \times 2 \times \sum_{l=6}^{n-1} p(n,l)$

$\mathcal{T}(n) \leq \mathcal{O}(n^2) \times \sum_{l=2}^{5} 5! \times p(n,l) + \mathcal{O}(n^2) \times 2 \times \sum_{l=6}^{n-1} p(n,l)$

$\mathcal{T}(n) \leq \mathcal{O}(n^2) \times 4 \times 5! \times \sum_{l=2}^{5} p(n,l) + \mathcal{O}(n^2) \times 2 \times \sum_{l=6}^{n-1} p(n,l)$

However, the growth rate of the number of nodes in the IP graph, which is the same as the growth rate of the integer partition of n, is known to be $\mathcal{O}(\frac{e^{\pi\sqrt{2n/3}}}{n})$ [Wilf, 2000]. Hence:

$\mathcal{T}(n) \leq \mathcal{O}(n^2) \times 4 \times 5! \times \mathcal{O}(\frac{e^{\pi\sqrt{2n/3}}}{n}) + \mathcal{O}(n^2) \times 2 \times \mathcal{O}(\frac{e^{\pi\sqrt{2n/3}}}{n})$

As a result, the time complexity of ACS is:

$\mathcal{O}(n^2 \times \frac{e^{\pi\sqrt{2n/3}}}{n})$ □

5 Empirical Evaluation

We evaluated the performance of our algorithm on several value distributions and compared it with ODP-IP [Michalak *et al.*, 2016], and CSG-UCT [Wu and Ramchurn, 2020]. For ODP-IP and CSG-UCT, we used the code provided by their authors. All the evaluations were performed on a DELL PowerEdge R740, Intel(R) Xeon(R) Gold 5120 CPU at 2.20 GHz with 384 GB of RAM at 2666 MHz. We considered two studies. The first, for sets of agents up to 27 in order to compare ACS to ODP-IP and CSG-UCT, while the second focused on large-scale problems with hundreds of agents. We only compared with CSG-UCT because ODP-IP cannot be run with more than a few dozen agents. In the first series of experiments, we used these two performance metrics: the **execution time** required to find an optimal or a near-optimal solution in the case of a heuristic; the **solution quality** $= \frac{v(\mathcal{CS}^+)*100}{v(\mathcal{CS}^*)}\%$, which is computed as the ratio between the value of the best coalition structure $\mathcal{CS}^+$ found by an algorithm after completion and the value of the optimal solution $\mathcal{CS}^*$ found by ODP-IP. We considered different value distributions. For each distribution, we randomly generated multiple problem instances varying the number of agents. We considered the following distributions: Agent-based Uniform, Modified Normal [Rahwan *et al.*, 2012], Agent-based Normal, Beta, Exponential, Gamma [Michalak *et al.*, 2016], Normal [Rahwan *et al.*, 2007], Uniform [Larson and Sandholm, 2000], Modified Uniform [Service and Adams, 2010], Normally Distributed Coalition Structures (NDCS) [Rahwan *et al.*, 2009], Pascal, Zipf and F [Changder *et al.*, 2020]. We also considered a Disaster Response distribution as introduced in [Wu and Ramchurn, 2020].

We do not show the results for a number of agents between 2 and 20 because no significant difference was observed. Moreover, in order for CSG-UCT to produce one exact result, the complete search space needs to be searched, meaning that running it to termination requires a huge runtime. Thus, we compare our algorithm to CSG-UCT, which we run up to 100 iterations as suggested in [Wu and Ramchurn, 2020].

Figure 6 illustrates the execution time of ACS, ODP-IP and CSG-UCT for some of the above distributions. For each point on Figure 6 we conduct an average of 50 tests. As we can see, for all the value distributions, our algorithm outperforms all the considered algorithms. For example, with 25 agents for the Normal distribution, the ratio of ODP-IP time and our algorithm time is 75.9, meaning that our algorithm takes 0.013% of the time taken by ODP-IP to provide a good quality solution of 99%.

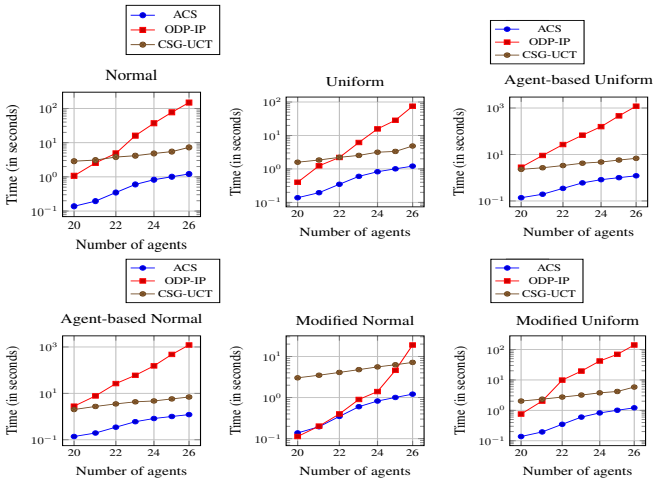

Figure 6: Time performance in seconds of ACS, ODP-IP and CSG-UCT for a number of agents between 20 and 26.

Table 1 reports the quality of the solutions provided by ACS and CSG-UCT given different value distributions. For each distribution, we took the average solution quality produced by the two algorithms for a number of agents between 20 and 26. As we can see, ACS outperforms CSG-UCT with better solution quality in many value distributions and remains competitive with other distributions, thus confirming the effectiveness of our algorithm.

Distribution	**Solution Quality**	
	ACS	**CSG-UCT**
Agent-based normal	99%	99%
Beta	99%	99%
Uniform	99%	98%
Pascal	99%	82%
Normal	98%	95%
Agent-based uniform	95%	89%
F Distribution	87%	85%
Exponential	80%	71%
Gamma	77%	65%

Table 1: Solution quality of ACS and CSG-UCT for sets of agents between 20 and 26. Here, the comparison is made with ODP-IP, which always provides the optimal solution when run to completion.

As mentioned above, ACS can be run with hundreds of agents. To demonstrate this, we conducted experiments on several value distributions and showed the **gain rate** produced by ACS and CSG-UCT given different numbers of agents (between 30 and 2000 agents). As no exact algorithm is run for these sets, the gain rate shown in Figure 7 is calculated as follows: $\frac{\frac{v(\mathcal{CS})}{v(\mathcal{CS}_s)}}{max(\frac{v(\mathcal{CS}^+_{ACS})}{v(\mathcal{CS}_s)}, \frac{v(\mathcal{CS}^+_{CSG-UCT})}{v(\mathcal{CS}_s)})}$, where $v(\mathcal{CS}^+_{ACS})$ and $v(\mathcal{CS}^+_{CSG-UCT})$ are the values of the best solutions provided by ACS and CSG-UCT, respectively. $v(\mathcal{CS}_s)$ is the value of the coalition structure that contains the singleton coalitions. This performance metric indicates the extent to which an algorithm has improved the solution compared to that of the singleton coalitions. We tested the two algorithms on multiple instances and recorded the solutions provided. As we can see from Figure 7, ACS produces better gain rates than CSG-UCT when run with hundreds of agents.

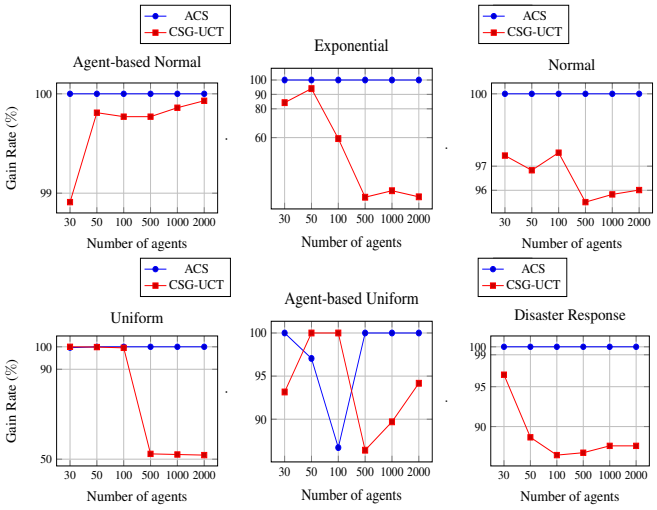

Figure 7: Gain rate of ACS and CSG-UCT when run with large numbers of agents.

Next, we evaluate the anytime property of ACS. Figure 8 shows the time required to provide a certain solution quality for 26 agents. Here, time is measured in seconds. As can be seen, ACS still provides good quality solutions. For instance, for the Normal, Uniform and Beta distributions, solution quality exceeds 99% after about 10 milliseconds. This is particularly interesting in problems with limited time.

In ACS, we perform permutations on several Initializations when the node is on level $l < 6$ and only on two when $l \geq 6$. Based on the tests that we conducted, we found that exceeding level 6 with all possible Initializations is extremely time-consuming and that no significant improvement on solution quality is observed.

To demonstrate this, we fixed different numbers of agents and varied the level up to which ACS considers all possible Initializations. Figure 9 shows the difference in runtime of ACS when the permutations are computed on all possible Initializations up to level $i_{/i\in\{5,6,7,8\}}$. The solution qualities produced by the different variants of ACS on 4 different value distributions: Normal, Agent-based Normal, Uniform and Beta, are comparable and no significant differences were observed. In particular, for these distributions, the level 5 variant of ACS already produces near-optimal solutions, where the solution quality exceeds 99%.

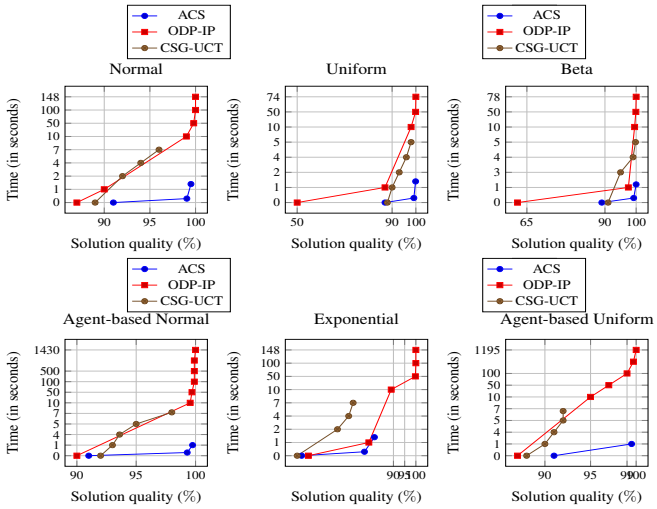

Figure 8: Time taken to produce a certain solution quality by ACS, ODP-IP and CSG-UCT for 26 agents. ODP-IP always provides the optimal solution when run to completion.

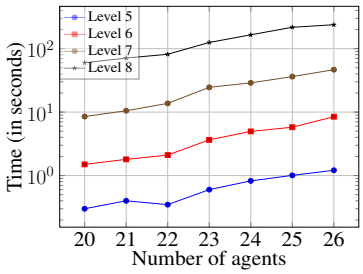

Figure 9: Time required in seconds to produce the final result when ACS computes all the combinations from phase 1 up to level i.

6 Conclusion

In this paper, we proposed a new algorithm for the CSG problem. The ACS algorithm can run with hundreds of agents. Our algorithm is based on an innovative heuristic algorithm and a novel representation of the search space, where each coalition is codified by an index-based code. The principle of ACS is hence to generate the coalition structures by permuting these codes. The main features of our algorithm are: (1) It is an anytime algorithm; (2) It finds a high quality solution compared to the state-of-the-art algorithms. Our empirical evaluation shows that our algorithm is effective and faster than the state-of-the-art algorithms by several orders of magnitude for different numbers of agents, while experimenting a wide range of value distributions.

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.

[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.

[Farinelli *et al.*, 2013] Alessandro Farinelli, Manuele Bicego, Sarvapali Ramchurn, and Marco Zuchelli. C-link: A hierarchical clustering approach to large-scale near-optimal coalition formation. In *Proc. of IJCAI*, pages 407–413, 2013.

[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.

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

[Tsvetovat and Sycara, 2000] Maksim Tsvetovat and Katia Sycara. Customer coalitions in the electronic marketplace. In *Proc. of the fourth international conference on Autonomous agents*, pages 263–264, 2000.

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

[Wu and Ramchurn, 2020] Feng Wu and Sarvapali D Ramchurn. Monte-carlo tree search for scalable coalition formation. In *Proc. of IJCAI*, pages 407–413, 2020.