Anytime Index-Based Search Method for Large-Scale Simultaneous Coalition Structure Generation and Assignment

Redha Taguelmimt, Samir Aknine, Djamila Boukredera, Narayan Changder

Anytime Index-Based Search Method for Large-Scale Simultaneous Coalition Structure Generation and Assignment

Redha Taguelmimt[a;*], **Samir Aknine**[a], **Djamila Boukredera**[b] **and Narayan Changder**[c]

[a]Univ Lyon, UCBL, CNRS, INSA Lyon, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Lyon, France
[b]Laboratory of Applied Mathematics, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria
[c]TCG Centres for Research and Education in Science and Technology, Kolkata, India

Abstract. Organizing agents into disjoint groups is a crucial challenge in artificial intelligence, with many applications where quick runtime is essential. The Simultaneous Coalition Structure Generation and Assignment (SCSGA) problem involves partitioning a set of agents into coalitions and assigning each coalition to a task, with the goal of maximizing social welfare. However, this is an NP-complete problem, and only a few algorithms have been proposed to address it for both small and large-scale problems. In this paper, we address this challenge by presenting a novel algorithm that can efficiently solve both small and large instances of this problem. Our method is based on a new search space representation, where each coalition is codified by an index. We have developed an algorithm that can explore this solution space effectively by generating index vectors that represent coalition structures. The resulting algorithm is anytime and can scale to large problems with hundreds or thousands of agents. We evaluated our algorithm on a range of value distributions and compared its performance against state-of-the-art algorithms. Our experimental results demonstrate that our algorithm outperforms existing methods in solving the SCSGA problem, providing high-quality solutions for a wide range of problem instances.

1 Introduction

An important research in artificial intelligence and game theory is how to partition a set of agents into disjoint exhaustive coalitions to maximize social welfare. Several paradigms have been explored in this context. Coalition formation is a coordination paradigm that has received extensive coverage in the last three decades in the literature. This involves forming coalitions and finding the optimal set of coalitions among agents through Coalition Structure Generation (CSG), with potential applications in several domains, including transportation [16], disaster response [22], distributed sensor networks [3], and e-commerce [20]. A CSG problem is defined on a set of n agents and a characteristic function v that assigns a value to each coalition. It is NP-Complete [15], and several algorithms have been proposed to solve it either optimally or approximately, ranging from dynamic programming [23, 12, 7, 1] to branch-and-bound and tree search [14] to hybrid algorithms [7, 2] to heuristic methods [17, 5, 4]. For this problem and these methods, the value of a coalition depends only on its members. However, for cases where a coalition of agents is not only evaluated with its members, but also by the task/goal it is assigned to, the CSG process may fail to produce good-enough solutions to the problems.

To illustrate this, consider the following example. Suppose we aim to allocate a set of electric vehicles, needing to charge, to charging stations. Notice that charging an electric vehicle takes significantly more time than refueling a non-electric vehicle. Thus, we need to minimize the amount of time electric vehicles wait to be charged. Since charging stations could be in different locations and not have the same power or capacity, assigning a group of electric vehicles to one station or another will not have the same impact. Therefore, if we do not consider the charging stations, we may form groups of vehicles that are not good enough. Hence, we need to match each group of vehicles with the charging station that maximizes the utility of the system.

With this in mind, the goal of our present work is to investigate how to solve the CSG problem in which the task/goal assigned to a coalition affects its value. This problem is the Simultaneous Coalition Structure Generation and Assignment (SCSGA) problem, in which the coalitions are assigned to different tasks and have different values/utilities given the task at hand. As discussed by [11], there is just a handful of recent research works tackling SCSGA in a few application domains. Among them, [9] introduces a search space representation based on multiset permutations of integer partitions [13] to prune large portions of the search space. Another work [10] proposes a dynamic programming algorithm for optimally solving the SCSGA problem. These methods are optimal and outperform CPLEX—a commercial state-of-the-art optimization software—when it comes to finding optimal solutions for small-scale problems. However, they can only be run with small numbers of agents, which limits their applicability to large-scale problems. Despite the interest, to the best of our knowledge, [11] is the only work that investigates large-scale SCSGA and compares the results of different heuristic algorithms, including monte carlo tree search, simulated annealing and local search. Hence, very few scalable solutions to the SCSGA problem exist and scalability remains a challenging issue for this problem.

In light of this, we propose a new method for the SCSGA problem. The following summarizes the contributions of this paper:

1. We design a new search space representation for the SCSGA problem. Specifically, we borrow ideas from the research [18], which has been successful in quickly solving the coalition structure gen-

* Corresponding Author. Email: redha.taguelmimt@gmail.com.

eration problem, and represent coalitions of agents as indexes that indicate both the coalition to which agents belong and the task assigned to them. Additionally, we partition the solution space into smaller subspaces. This representation allows anytime problem solving of SCSGA.
2. We develop a new search method for exploring coalition structures, which relies on our search space representation. Our algorithm is capable of running on large-scale problem instances with hundreds of agents, and it is anytime, meaning it can be stopped at any point to return the best solution found thus far.
3. We empirically demonstrate that our method outperforms the state-of-the-art algorithms when generating high quality solutions for solving both small and large problems. In addition to this, we propose a formulation of the electric vehicle allocation problem that motivated our work and design a value distribution for this problem.

2 SCSGA Problem Formulation

We investigate the simultaneous coalition structure generation and assignment problem (SCSGA). In this problem, we are given a set $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ of n agents, a set $\mathcal{T} = \{t_1, t_2, ..., t_k\}$ of k tasks or goals, and a value $v(\mathcal{C}, t)$ for each coalition-task pair, which denotes the efficiency of the coalition when assigned to the task. A coalition $\mathcal{C}$ is any subset of $\mathcal{A}$, including the empty set. There are 2^n possible coalitions for each task, resulting in a total of $k2^n$ coalition values. An ordered coalition structure is a partition of the set of agents into exactly k disjoint coalitions. Formally, given a set of exactly k coalitions $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, an ordered coalition structure is a collection of k ordered coalitions $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$ that satisfies the following constraints: $\bigcup_{j=1}^{k} \mathcal{C}_i = \mathcal{A}$ and for all $i, j \in \{1, 2, ..., k\}$ where $i \neq j, \mathcal{C}_i \cap \mathcal{C}_j = \emptyset$. It is possible to pair the coalitions to the tasks differently, but for simplicity in this paper, we consider that the tasks are ordered, meaning that the coalitions in positions $1, 2, \ldots, k$ are assigned to tasks $t_1, t_2, \ldots, t_k$, respectively. For example, with 5 agents $\{a_1, a_2, a_3, a_4, a_5\}$ and $k = 3$, in the ordered coalition structure $\{\{a_2, a_3, a_5\}, \emptyset, \{a_1, a_4\}\}$, the coalition $\{a_2, a_3, a_5\}$ is assigned to task t_1, the empty coalition $\emptyset$ is assigned to task t_2, and the coalition $\{a_1, a_4\}$ is assigned to task t_3. Notice that ordered coalition structures such as $\{\{a_2, a_3, a_5\}, \emptyset, \{a_1, a_4\}\}$ and $\{\emptyset, \{a_2, a_3, a_5\}, \{a_1, a_4\}\}$ are not equivalent, even if the coalitions are composed by the same agents, because they are not assigned to the same task (the coalition $\{a_2, a_3, a_5\}$ is assigned to task t_1 in the first ordered coalition structure and to task t_2 in the second one).

The value of an ordered coalition structure $\mathcal{CS}$ is assessed as the sum of the values of the coalitions that comprise it: $V(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C}, t(\mathcal{C}))$, where $t(\mathcal{C})$ is the task assigned to the coalition $\mathcal{C}$. The goal of the SCSGA problem is to find the optimal solution, which is the highest-valued ordered coalition structure. Hence, for the remainder of this paper, we use the terms ordered coalition structure and solution interchangeably.

3 Related Work

Very few algorithms [11, 9, 10] have been developed for the SCSGA problem. On the other, the closely related problem CSG has been extensively studied, and many algorithms have been developed to solve it, which are discussed in Section 1. However, without redesigning them, these methods are unsuitable for SCSGA and are specifically designed to solve the CSG problem without tasks/goals, for which they consider coalition structures of any size, whereas in SCSGA we only consider ordered coalition structures of size k. Moreover, empty coalition structures are not considered in CSG because they have no impact since they correspond to the absence of coalition formation. In contrast, in SCSGA, when a coalition assigned to a task/goal is of size 0, it means that no agent is assigned to the task/goal. Additionally, the order of coalitions in CSG is not important because there are no tasks/goals to assign, whereas in SCSGA, the order of coalitions is crucial since it determines which coalition is assigned to each task. Thus, it is important to develop new algorithms specifically tailored to the SCSGA problem.

4 Representing Ordered Coalition Structures

We build upon the representation of the search space proposed by [13] and adopt a similar approach to that used by [9] to address the specific challenge of considering coalition structures that contain exactly k coalitions. To represent the coalitions and structures with integers, we add an integer layer on top of this representation. Recall that an integer partition of n is a vector of positive integers that sum to n. For instance, for $n = 4$ agents, the integer partitions are: $[4], [1, 3], [2, 2], [1, 1, 2], [1, 1, 1, 1]$. Specifically, we generate the set I_1 of integer partitions of n of size at most k. For example, for $n = 5$ and $k = 3$, $I_1 = \{[5], [1, 4], [2, 3], [1, 1, 3], [1, 2, 2]\}$. Then, we generate the set I_2 of configurations by appending zeros to the partitions of size less than k. For example, for $n = 5$ and $k = 3$, $I_2 = \{[5, 0, 0], [1, 4, 0], [2, 3, 0], [1, 1, 3], [1, 2, 2]\}$. Finally, we generate the set I_3 of all the combinations of the configurations of I_2. This can efficiently be done using the algorithm based on tree-traversal proposed in [19], or the algorithm based on loopless generation proposed in [21]. For example, for $n = 5$ and $k = 3$, $I_3 = \{[5, 0, 0], [0, 5, 0], [0, 0, 5],$
$[1, 4, 0], [1, 0, 4], [4, 1, 0], [4, 0, 1], [0, 1, 4], [0, 4, 1],$
$[2, 3, 0], [2, 0, 3], [3, 2, 0], [3, 0, 2], [0, 2, 3], [0, 3, 2],$
$[1, 1, 3], [1, 3, 1], [3, 1, 1], [1, 2, 2], [2, 1, 2], [2, 2, 1]\}$.

I_3 represents all subspaces of the search space, where each subspace (configuration) is represented by a filled integer partition of n of size k. Specifically, each filled partition has k parts that sum to n, and may contain 0 to $k - 1$ zeros. Each partition represents coalition structures that contain exactly k coalitions where the sizes and the order of the coalitions matches its parts. For example, $[2, 1, 2]$ represents all coalition structures that contain one coalition of size 2 assigned to the first task, one coalition of size 1 assigned to the second task, and one coalition of size 2 assigned to the third task.

To account for cases where no coalition is assigned to a task, we consider the empty set as a coalition of size 0. Therefore, when a task has no assigned coalition, it is represented by a coalition of size 0.

4.1 Coalition Representation

Each configuration in I_3 represents a set of coalition structures that satisfy the criteria specified by the configuration. For example, $[0, 4, 1]$ represents all coalition structures that contain one coalition $\mathcal{C}_0$ of size 0 assigned to the first task, one coalition $\mathcal{C}_1$ of size 4 assigned to the second task, and one coalition $\mathcal{C}_2$ of size 1 assigned to the third task.

We adapt the representation presented in [18] to the configurations of I_3. For each configuration, we represent each coalition with its index. For instance, C_0 is represented with index 0, C_1 with index 1, and so on. These indexes serve two purposes: they identify the coalitions and determine their assigned tasks. For instance, index 0 identifies coalition C_0 and specifies that it is assigned to task 0.

4.2 *Ordered Coalition Structure Representation*

Having described how we represent a coalition given a configuration, we now focus on how we represent the coalition structures of the configuration.

Similar to [18], we represent each coalition structure with a vector of indexes of size $n = |\mathcal{A}|$: $[x_1, x_2, .., x_p, .., x_n]$. The indexes in positions $1, 2, \ldots, n$ represent the coalitions to which the agents $a_1, a_2, \ldots, a_n$ belong, respectively. For instance, let $\mathcal{A}$ be a set of $n = 5$ agents. Consider the coalition structure $\mathcal{CS}_1 = \{\emptyset, \{a_2, a_3, a_5\}, \{a_1, a_4\}\}$ of the subspace $[0, 3, 2]$ containing three coalitions: $\mathcal{C}_0 = \emptyset$, $\mathcal{C}_1 = \{a_2, a_3, a_5\}$ and $\mathcal{C}_2 = \{a_1, a_4\}$. $\mathcal{C}_0$, $\mathcal{C}_1$ and $\mathcal{C}_2$ are represented with indexes 0, 1 and 2, respectively. $\mathcal{CS}_1$ is encoded by the vector of indexes $[x_1\ x_2\ ..\ x_p\ ..\ x_6]$, where $x_{p/p=1..6} = j \Leftrightarrow a_p \in \mathcal{C}_j$. Note that as the coalitions of size 0 have no agent and the vector represents the coalitions of the agents, then the indexes that represent the coalitions of size 0 do not appear in the vectors that represent the coalition structures. Figure 1.a shows the representation of the coalition structure $\mathcal{CS}_1$ using the vector of indexes [2 1 1 2 1] of size $n = 5$, where the number of different indexes in the vector equals the number of coalitions of size more than 0 forming $\mathcal{CS}_1$. The index associated to a_1 and a_4 in the vector is 2 because a_1 and a_4 belong to $\mathcal{C}_2$ in $\mathcal{CS}_1$, while the index associated to a_2, a_3 and a_5 in the vector is 1 because a_2, a_3 and a_5 belong to $\mathcal{C}_1$ in $\mathcal{CS}_1$. However, index 0 is not associated with any agent because the coalition of index 0 is empty. Nevertheless, it is important to have the index 0 as this allows to identify the tasks of the other non-empty coalitions. Note that any permutation of these indexes provides a different coalition structure. For example, the vector of indexes [2 2 2 1 1] represents the coalition structure $\mathcal{CS}_2 = \{\emptyset, \{a_4, a_5\}, \{a_1, a_2, a_3\}\}$ (see Figure 1.b).

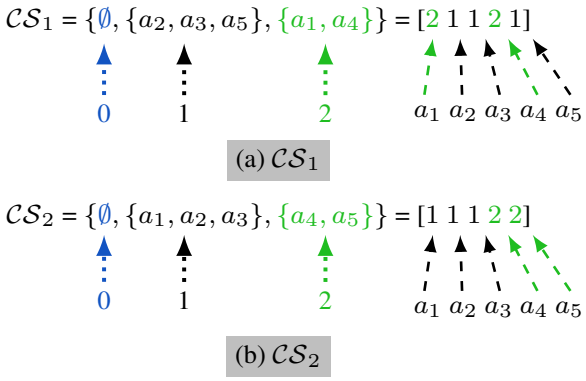

Figure 1: Representation of the coalition structures $\mathcal{CS}_1 = \{\emptyset, \{a_2, a_3, a_5\}, \{a_1, a_4\}\}$ and $\mathcal{CS}_2 = \{\emptyset, \{a_1, a_2, a_3\}, \{a_4, a_5\}\}$ using our approach. The indexes 0,1 and 2 represent the coalitions of the agents and the task assigned to each coalition.

4.3 *Generalization*

We now extend this representation to any configuration in I_3. All configurations of I_3 have k parts. We assign an index $j \in 0, 1, ..., k-1$ to each coalition $\mathcal{C}_j$. To represent a coalition structure $\mathcal{CS}$ gathered in a configuration, we create a vector of size $n = |\mathcal{A}|$, where each non-empty coalition $\mathcal{C}_j$ is represented l times in the vector, where $l = |\mathcal{C}_j|$. The indexes associated with empty coalitions are not included in the vector because $l = 0$. For example, consider the configuration $[0, 3, 2]$, which has three coalitions with sizes 0, 3, and 2 respectively. Let $\mathcal{C}0, \mathcal{C}1, \mathcal{C}2$ represent the empty, second, and third coalition respectively. The vector representing a coalition structure of this configuration will have no occurrences of 0 because $\mathcal{C}0$ is empty, three occurrences of 1 because $\mathcal{C}1$ has three agents, and two occurrences of 2 because $\mathcal{C}2$ has two agents (see Figure 1). Figure 2 shows the vectors representing all the coalition structures for a problem with $n = 5$ agents and $k = 2$.

Theorem 1. *The entire solution space can be represented with our ordered coalition structure representation.*

Proof. We stated that each configuration in $\mathcal{I}_3$ represents a set of ordered coalition structures meeting the criteria of the configuration. Assume that there exists an ordered coalition structure that is not represented with our method, meaning that there is no vector of indexes that represents this coalition structure. Now, as all of the ordered coalition structures of each configuration are represented with our method (as defined in the subsection 3.3), this means that this coalition structure is not represented within the configurations in $\mathcal{I}_3$. However, as proved by [9], every ordered coalition structure belongs to one of the configurations, so Theorem 1 holds. □

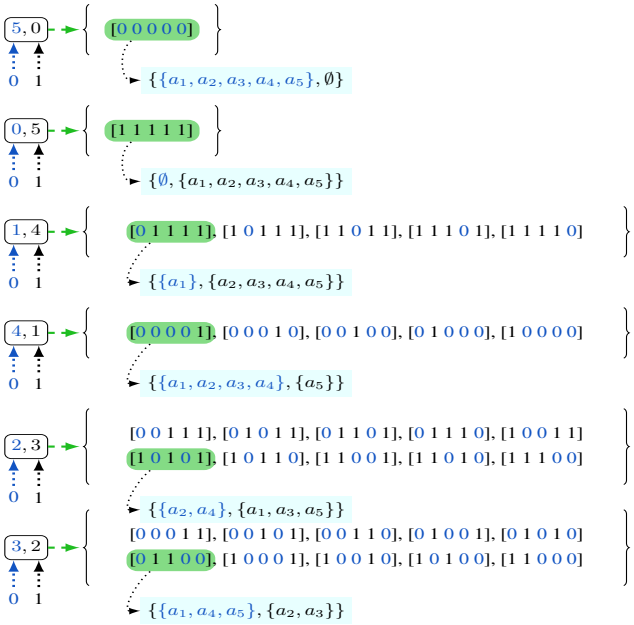

Figure 2: An example of the representation of all the coalition structures for a problem with five agents and $k = 2$. Our representation complements that of [9]. Each vector in this figure corresponds to a coalition structure. For example, the coalition structures highlighted in the blue rectangles correspond to the vectors highlighted in the green rectangles.

5 PISA Algorithm

In this section, we present our PISA (**P**ermutation-based **I**ndex **S**earch and **A**ssignment) algorithm. Our starting point is the set $\mathcal{I}_3$ of configurations. Given this set, PISA partially generates the ordered coalition structures of each configuration.

5.1 Step 1: Initial Vectors Generation

PISA generates the vectors that represent the ordered coalition structures by permuting the indexes of selected initial vectors. This process begins with the generation of a set of indexes $\mathcal{X} = \{0, 1, \ldots, k\}$ and a set $\mathcal{Y} = \{k, k-1, \ldots, 0\}$ that is the reverse of $\mathcal{X}$. Next, PISA generates the set $\mathcal{Z}$, which contains permutations of the sets $\mathcal{X}$ and $\mathcal{Y}$, as well as the sets $\mathcal{X}$ and $\mathcal{Y}$ themselves. The sets $\mathcal{X}$ and $\mathcal{Y}$ are used because they are easily identifiable and can be used to generate a large number of different permutations in $\mathcal{Z}$. To construct $\mathcal{Z}$, PISA starts with the first index and permutes it with each of the other indexes. Then, PISA moves to the next index and applies the same permutation operations to it, until it reaches the last index. For example if $k = 3$, then $\mathcal{X} = \{0, 1, 2\}$ and $\mathcal{Y} = \{2, 1, 0\}$. The set $\mathcal{Z}$ would contain the sets of indexes that result from the permutations performed on $\mathcal{X}$ (which are: $\{1, 0, 2\}$, $\{2, 1, 0\}$, $\{0, 2, 1\}$), and those performed on $\mathcal{Y}$ (which are: $\{1, 2, 0\}$, $\{0, 1, 2\}$, $\{2, 0, 1\}$). Hence, $\mathcal{Z}$ = { $\{0, 1, 2\}$, $\{2, 1, 0\}$, $\{1, 0, 2\}$, $\{0, 2, 1\}$, $\{1, 2, 0\}$, $\{2, 0, 1\}$ }, by removing the duplicates. The sets $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ are the same for all configurations as they all contain k coalitions.

Now, for each configuration, PISA generates some pre-initial vectors that are used to generate the initial vectors. To generate the pre-initial vectors, PISA takes each set of indexes in $\mathcal{Z}$ and generates the corresponding pre-initial vector by positioning the indexes according to the order of the indexes in the set, and repeating each index as many times as the size of the coalition it represents. For example, with 5 agents and the subspace $[0, 3, 2]$, the pre-initial vector of the set of indexes $\{0, 1, 2\}$ is [1 1 1 2 2] because the coalitions represented with 0, 1, and 2 are of sizes 0, 3, and 2, respectively. To generate the initial vectors, PISA permutes the indexes of each pre-initial vector. For each pre-initial vector, PISA starts with the first index and permutes it with each of the other indexes. Then, PISA moves to the next index and applies the same permutation operations to it, until it reaches the last index. For example, with the pre-initial vector [1 1 1 2 2], PISA generates these initial vectors: [2 1 1 1 2], [2 1 1 2 1], [1 2 1 1 2], [1 2 1 2 1], [1 1 2 1 2], [1 1 2 2 1]. An illustration example of this phase is shown in Figure 3.

5.2 Step 2: Ordered Coalition Structure Generation

In phase 2, PISA generates the ordered coalition structures using the initial vectors obtained in phase 1. To do this, PISA permutes the indexes of these vectors, and after each permutation, it obtains a new vector that represents a new ordered coalition structure. To generate the ordered coalition structures, PISA starts by taking one initial vector at a time. It then generates the permutations as follows: first, PISA starts with the first index of the initial vector and permutes it with each of the other indexes. Then, PISA moves to the next index and applies the same permutation operations to it, until it reaches the last index. This generates a new vector, which represents a different ordered coalition structure. In each generated vector, the indexes determine which agents belong to each coalition, and the index of each coalition determines the task assigned to the coalition. If no agent belongs to a coalition, meaning that no index of the coalition appears in the vector, then that coalition is empty. For example, the vector [1 1 1 2 2] does not contain the index 0, meaning that the coalition of index 0, which is assigned to the task of index 0, is empty. Figure 4 shows an illustration example of this phase.

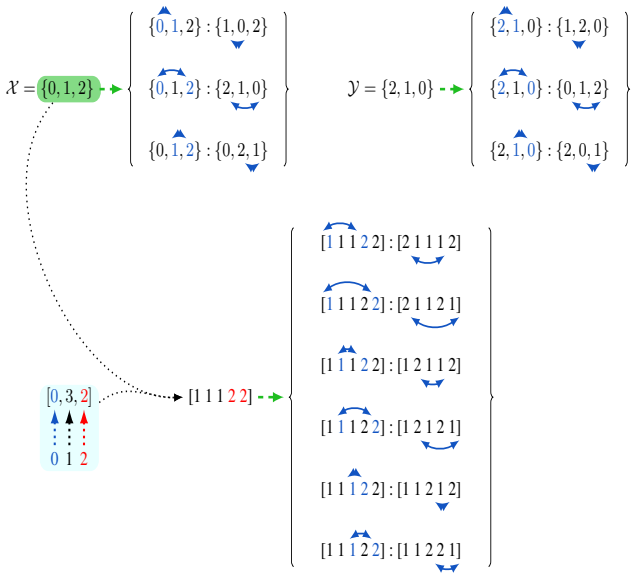

Figure 3: An illustration example of the initial vector generation for the configuration $[0, 3, 2]$. We first generate the set $\mathcal{Z}$ by permuting the indexes of the sets $\mathcal{X}$ and $\mathcal{Y}$. Then we generate the pre-initial vectors that we use to generate the initial vectors. In this example, we only show the initial vectors generated using one pre-initial vector [1 1 1 2 2]. Note that when a coalition is of size 0, the vectors do not contain the index of that coalition as no agent belongs to it.

5.3 Ordering the Configurations

As described in the previous subsections, PISA considers the configurations one by one and performs the index permutations on the initial vectors and computes the initial vectors for each configuration. The order in which the configurations are considered can affect the quality of the anytime solutions found during the execution, and some configurations may be unlikely to improve the current best solution. To order the configurations and to enable pruning out the configurations that have no chance of containing an optimal solution, PISA computes the upper bound of each configuration and sorts them using these values. Let $Max_{s,i}$ denote the highest value of the coalitions of size s assigned to task i. By computing $Max_{s,i}$ for every coalition size s and task i, it is possible to compute the upper bound of a configuration $\mathcal{S}$: $UB_{\mathcal{S}} = \sum_{i=0}^{k-1} Max_{Integers(\mathcal{S})[i],i}$, where $Integers(\mathcal{S})$ is the set of integers that form the configuration $\mathcal{S}$. Note that the maximum value is 0 for empty coalitions of size 0 (i.e. $Max_{0,i} = 0$). For example, consider the configuration $\mathcal{S} = [0, 3, 2]$. Here, $Integers(\mathcal{S}) = \{0, 3, 2\}$, and we calculate the upper bound of $\mathcal{S}$ as follows: $UB_{[0,3,2]} = Max_{0,0} + Max_{3,1} + Max_{2,2}$. PISA uses the upper bounds of the configurations to order them and prune out those that have no chance of improving the current best solution. PISA repeats the subspace removal procedure each time it finishes evaluating a configuration that improves the solution. This enables PISA to remove configurations that do not have a better upper bound than the last best solution found. By using the upper bound to order and prune configurations, PISA can focus its search on those that are most promising and improve the solution quality faster.

Algorithm 1 outlines the functioning of PISA. It initializes the best coalition structure to be the grand coalition assigned to the first task and then iteratively searches for better coalition structures as described in lines 6-19. More specifically, PISA processes one subspace (configuration) at a time. It generates the initial vectors for

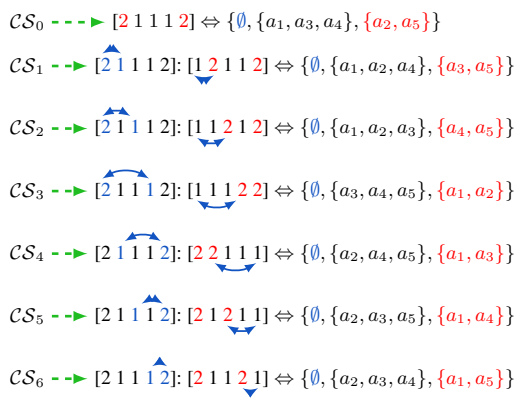

Figure 4: An illustration example of the permutations performed by PISA when searching the configuration $[0, 3, 2]$ with the initial vector [2 1 1 1 2]. With this initial vector, PISA performs 6 permutations and generates 7 coalition structures.

the subspace as described in Section 5.1. Then, PISA permutes the indexes of the initial vectors to obtain index vectors that represent ordered coalition structures. Each permutation, which corresponds to a new ordered coalition structure, is then evaluated to determine if it improves upon the value of the current best solution.

Algorithm 1: The PISA algorithm

Input: A set of n agents $\mathcal{A}$ and a set of k tasks $\mathcal{T}$. Set of all coalitions and the value $v(\mathcal{C}, t)$ of each coalition $\mathcal{C}$ and task t.
Output: The best structure found $\mathcal{CS}^+$ and its value $\mathcal{V}^+$.

Generate partitions of n of size at most k and store them in I_1
Add zeros to the partitions in I_1 of size less than k and generate the set I_2
Generate the set I_3 of combinations of the configurations in I_2
$\mathcal{CS}^+ \leftarrow \mathcal{A}, t_1 \triangleright$ *initialization with the grand coalition assigned to the first task*
$\mathcal{V}^+ \leftarrow v(\mathcal{A}, t_1)$
foreach *Subspace* $\mathcal{S} \in I_3$ **do**
  Generate the initial vectors and store them in ϑ
  foreach $\varsigma \in \vartheta$ **do**
    permute the indexes of ς and store them in the set $\mathcal{PM}$
    foreach $x \in \mathcal{PM}$ **do**
      $\mathcal{CS} \leftarrow g(x) \triangleright g$ is the conversion function from the vector x to a coalition structure $\mathcal{CS}$
      $\mathcal{V}(\mathcal{CS}) = \sum_{j=0}^{l-1} v(\mathcal{C}_j, t_j)$, where $(\mathcal{C}_j, t_j) \in \mathcal{CS}$
      if $\mathcal{V}(\mathcal{CS}) > \mathcal{V}^+$ **then**
        $\mathcal{V}^+ \leftarrow \mathcal{V}(\mathcal{CS})$
        $\mathcal{CS}^+ \leftarrow \mathcal{CS}$
      end
    end
  end
end
Return $\mathcal{CS}^+$, $\mathcal{V}^+$

In Theorem 2, we prove that the proposed algorithm is anytime, meaning that it can provide a solution at any time during its execution, and the solution quality improves over time. This property is desirable in many real-world scenarios where the available computation time may be limited or uncertain. This feature also makes PISA a robust and flexible algorithm that can handle various problem instances effectively.

Theorem 2. *The PISA algorithm is an anytime algorithm.*

Proof. The PISA algorithm starts with an initial coalition structure and then repeatedly improves it over time by performing several permutation operations. At any point during the execution, the algorithm can return the currently best solution found so far. Moreover, since the algorithm continues to improve the solution by iteratively exploring the search space and refining the coalition structure, the quality of the solution will continue to improve as long as the algorithm is allowed to run. Therefore, we conclude that the PISA algorithm is an anytime algorithm □

In the following theorem, we show the time needed to search a subspace by PISA.

Theorem 3. *Given n agents and k tasks, a subspace is searched in PISA in $\mathcal{O}(n^4 \times k^2)$ time.*

Proof. For any subspace (configuration), the number of permutations for a single initial vector is $\mathcal{O}(\frac{n^2}{2}) = \mathcal{O}(n^2)$. With this, the total operations performed $\mathcal{T}(n, k)$ is as follows:
$\mathcal{T}(n, k) = \mathcal{O}(n^2) \times i(k)$, where $\mathcal{O}(n^2)$ is the number of permutations and $i(k)$ is the number of initial vectors. We will now compute the number of initial vectors. PISA starts by generating two sets $\mathcal{X}$ and $\mathcal{Y}$ of size k, and the set $\mathcal{Z}$ by permuting the sets $\mathcal{X}$ and $\mathcal{Y}$. The number of sets in $\mathcal{Z}$, and hence of pre-initial vectors, is $\mathcal{O}(k^2)$. Now, given each pre-initial vector, PISA generates $\mathcal{O}(n^2)$ initial vectors. Hence, the total number of initial vectors is $i(k) = \mathcal{O}(k^2) \times \mathcal{O}(n^2)$. As a result, the number of operations performed by PISA is:
$\mathcal{T}(n, k) = \mathcal{O}(n^2) \times \mathcal{O}(k^2) \times \mathcal{O}(n^2)$
$\mathcal{T}(n, k) = \mathcal{O}(n^4 \times k^2)$. □

6 Empirical Evaluation

We experimentally compare the PISA algorithm against representative state-of-the-art SCSGA algorithms for both small and large-scale problems. We compare both the solution quality (for small-scale problems) and the gain rate (for large-scale problems). We implemented our algorithm in Java and in the comparisons we used the codes provided by the authors of [11]. We compared our algorithms to the random-restart hill climb algorithm (RHC) and the random-restart hybrid algorithm (RHY) that combines the greedy algorithm with the hill climb algorithm, which have been shown to achieve state-of-the-art performance for large-scale SCSGA [11]. The algorithms were run on an Intel Xeon 2.30GHz E5-2650 CPU with 256GB of RAM. To generate the problem instances, we considered the following value distributions: NPD (normal probability distribution) [13], UPD (uniform probability distribution) [6], SNPD (sparse UPD) and SUPD (sparse UPD) [11].

- **NPD:** Each value of a coalition is obtained from: $v(\mathcal{C}, t) \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = 10 \times |\mathcal{C}|$ and $\sigma = 0.1$ [13].
- **UPD:** $v(\mathcal{C}, t) \sim \mathcal{U}(0, a)$, with $a = |\mathcal{C}|$ [6].
- **SNPD:** $v(\mathcal{C}, t) \sim \mathcal{N}(\mu_1, \sigma^2)$ with probability 0.01, else $v(\mathcal{C}, t) \sim \mathcal{N}(\mu_2, \sigma^2)$, with $\mu_1 = 1$, $\mu_2 = 0.1$, $\sigma = 0.1$ [11].
- **SUPD:** $v(\mathcal{C}, t) \sim \mathcal{U}(0, a_1)$, where $a_1 = 1$, with probability 0.01, else $v(\mathcal{C}, t) \sim \mathcal{U}(0, a_2)$, where $a_2 = 0.1$ [11].

The result for each value distribution was produced by computing the average result from 50 generated problem instances per value distribution.

We also address the Electric Vehicle Allocation problem, which involves assigning a set of electric vehicles to charging stations with the aim of minimizing waiting time, given that charging an electric vehicle takes considerably more time than refueling a non-electric vehicle. For clarity, bear in mind that a charging station is a specific physical location that has one or more charging posts. A charging post itself may have one or more ports, where each port can charge a single electric vehicle. A set of electric vehicles $\mathcal{C}$ is allocated to a single charging station, and the utility function $v(\mathcal{C}, t)$ required by coalition formation algorithms can be defined in various ways. In this paper, we consider a simple model in which a set of n electric vehicles is to be allocated to a set of k stations to minimize waiting time.

We denote the power and capacity of each charging station c_j as $\phi(c_j)$ and $p(c_j)$, respectively. The more power delivered, the faster the electric vehicle charges. The capacity of each station varies between 1 and 10, and each electric vehicle requires a specific amount of charge time, depending on its battery capacity, the charging station's power, and the state of the battery. To simplify the model, we assume that each vehicle a_i requires a charging time $r_{ij} = \frac{\mathcal{S}_i}{min(\phi(c_j),\Phi_i)} \times 60$ in each station c_j, where $\mathcal{S}i$ is the battery capacity and Φi is the maximum charging power the electric vehicle can handle.

The utility of a certain set $\mathcal{C}$ and a station c_j is defined as $v(\mathcal{C}) = \sum_{i \in \mathcal{C}} r_{ij}/p(c_j)$. This function can be customized based on the specific requirements of the application and designed by a domain expert.

Each coalition structure $\mathcal{CS}$ is multivalued and has two values that need to be minimized. The first value, $V_1(\mathcal{CS}) = \max_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$, is equal to the value of its highest-valued coalition and is used to minimize the time to finish charging the last vehicle. The second value, $V_2(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$, is evaluated as the sum of the values of its composing coalitions and is used to obtain the best overall performance. To solve this SCSGA problem, both values need to be minimized.

6.1 *Small-Scale Benchmarks*

In this subsection, we present a comparative analysis of our algorithm's performance against state-of-the-art methods on small-scale problems involving a limited number of agents. To evaluate the performance of the algorithms, we conducted experiments on five different value distributions and measured the solution quality achieved by each algorithm. It should be noted that the algorithms exhibit varying behaviors depending on the specific value distributions used in the experiments.

To establish a benchmark for comparison, we also computed the optimal solutions using CPLEX [8]. However, for problems with a large number of agents, CPLEX is not able to generate feasible solutions in a reasonable amount of time. As a result, we focused on benchmarking the algorithms on problems with a limited number of agents.

We present the solution quality obtained by each algorithm in comparison to the optimal solutions computed by CPLEX in Figure 5. It is evident from the figure that all the algorithms, including our proposed algorithm PISA, yield near-optimal solutions much faster than CPLEX. Specifically, even for the most challenging problem set, PISA achieves solutions that are approximately 99% of the optimal within just 1 second. Our algorithm outperforms the current state-of-the-art and the other methods in terms of speed and solution quality. Figure 6 shows the solution quality of the algorithms when varying the number of agents. As can be seen, all algorithms quickly produce high-quality solutions, and there is a clear general trend that the PISA and RHC outperform RHY.

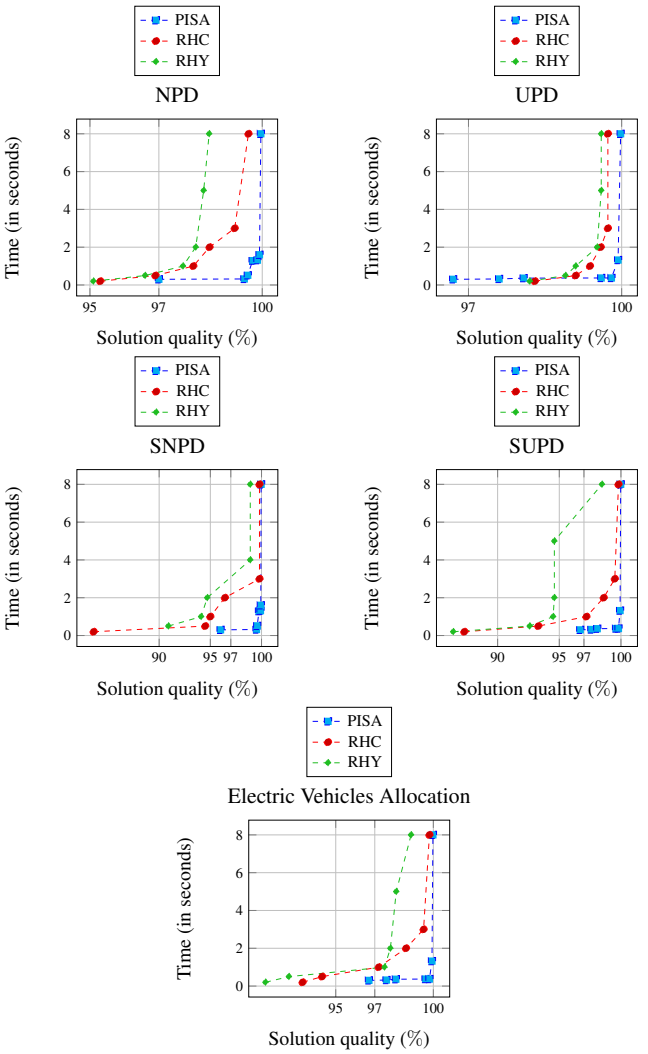

Figure 5: Average solution quality obtained by PISA, RHC and RHY for NPD, UPD, SNPD and SUPD value distributions with 15 agents and 5 tasks. The optimal solutions are computed using CPLEX, which always provides the optimal solution.

6.2 *Large-Scale Benchmarks*

CPLEX and other optimal algorithms have limited scalability and are only able to handle problems with a small number of agents. In contrast, our proposed algorithm can efficiently handle large-scale problems involving hundreds or thousands of agents. While algorithms such as RHC and RHY can also handle large-scale problems, it is infeasible to guarantee to find an optimal solution in a feasible time due to the exponential nature of the solution space. Therefore, it is not possible to compute the solution quality by comparing the solutions obtained to the optimal solutions. To compare the performance

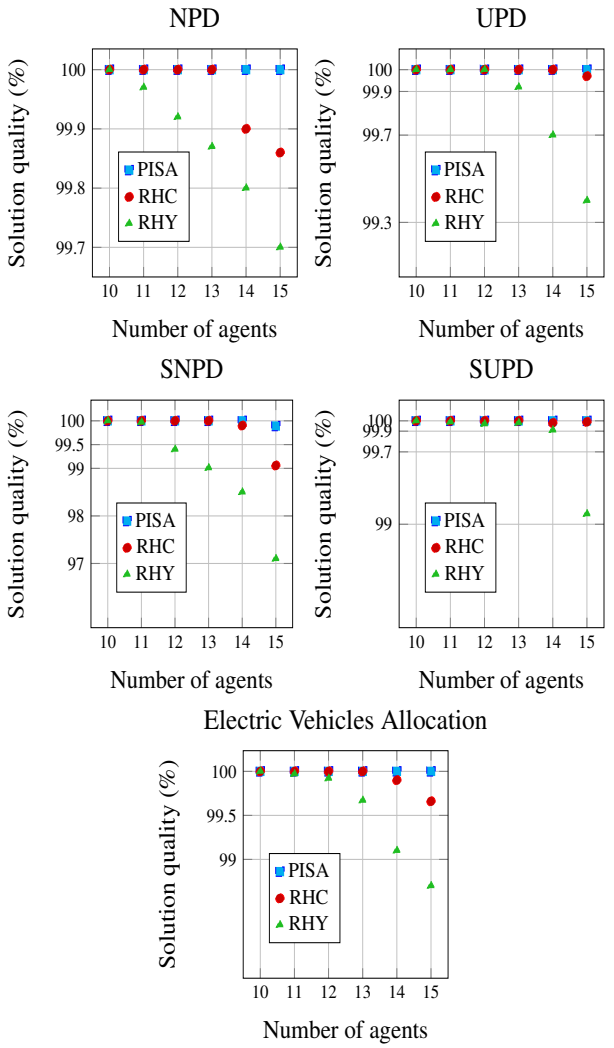

Figure 6: Solution quality of PISA, RHC and RHY obtained after 8 seconds for sets of agents between 10 and 15 and 5 tasks.

of the algorithms on large-scale problems, we use the gain rate as a metric, which measures the improvement of the solution achieved by each algorithm relative to the value of the singleton coalition structure. This approach allows us to evaluate the algorithms' performance without the need for an optimal solution as a benchmark, which is not feasible for large-scale problems. We define the *gain rate* as $\frac{\frac{v(\mathcal{CS})}{v(\mathcal{CS}_s)}}{max_i(\frac{v(\mathcal{CS}_i^+)}{v(\mathcal{CS}_s)})}$, where $v(\mathcal{CS}_s)$ denotes the value of the singleton coalition structure, which represents a partition into n coalitions, each containing a single agent. The term $v(\mathcal{CS})$ represents the value of the coalition structure obtained by the algorithm being evaluated, and $v(\mathcal{CS}_i^+)$ denotes the value of the best solution obtained by algorithm $i \in \{PISA, RHC, RHY\}$.

In Figure 7, we present the results of our large-scale benchmarks, comparing the performance of PISA, RHC, and RHY on instances of different value distributions. To ensure that the algorithms are competing on an equal search time, we used the same time limit for all the algorithms. Our results demonstrate that PISA, RHY, and RHC can quickly generate high-quality solutions for problems with immense search spaces and large input sizes. We observe that PISA consistently outperforms RHC and RHY on a majority of the tests, especially on problem instances of the Electric Vehicles Allocation and NPD distributions. Hence, our experiments demonstrate that our proposed algorithm, PISA, can efficiently solve large-scale problems and outperform the other algorithms on different value distributions.

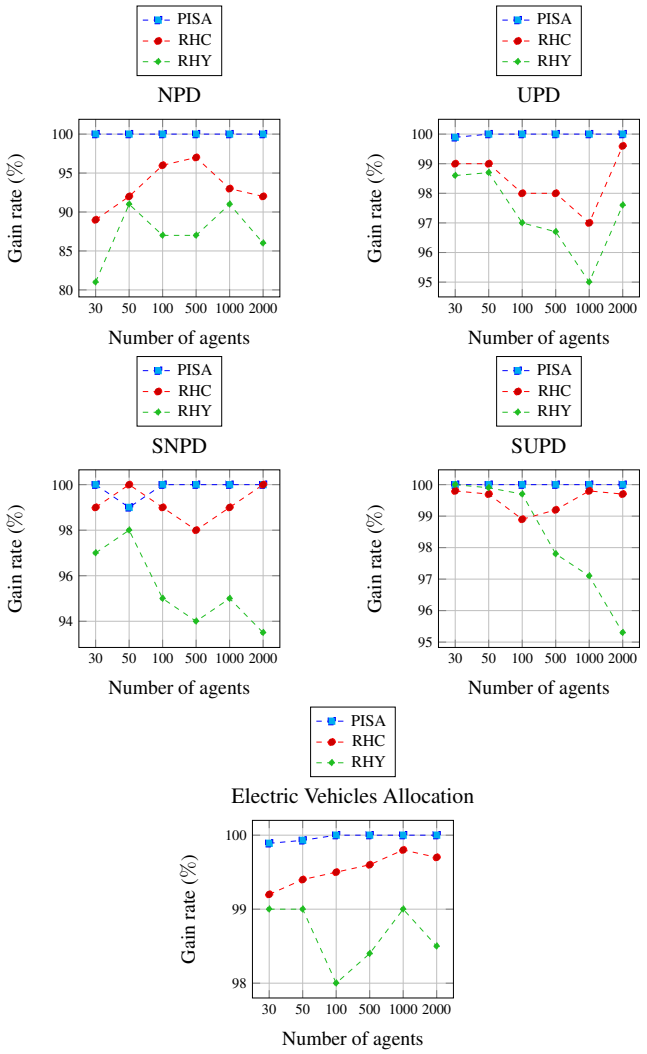

Figure 7: Gain rate obtained by PISA, RHC and RHY for NPD, UPD, SNPD and SUPD value distributions with 20 tasks.

7 Conclusion

This paper presents a novel algorithm for solving the simultaneous coalition structure generation and assignment (SCSGA) problem. Our approach involves a new search space representation, where an index codifies each coalition. Using this representation, we developed an algorithm that efficiently explores the solution space by generating index vectors that represent coalition structures. Our proposed algorithm is anytime and can handle large-scale problems. We evaluated our algorithm on a range of value distributions and compared its performance against state-of-the-art algorithms. Our experimental results demonstrate that our algorithm outperforms existing approaches in solving the SCSGA problem, providing high-quality solutions for a wide range of problem instances.

References

[1] Narayan Changder, Samir Aknine, and Animesh Dutta, 'An effective dynamic programming algorithm for optimal coalition structure generation', in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 721–727. IEEE, (2019).

[2] Narayan Changder, Samir Aknine, Sarvapali D. Ramchurn, and Animesh Dutta, 'Boss: A bi-directional search technique for optimal coalition structure generation with minimal overlapping (student abstract)', in *Proc. of AAAI*, volume 35, pp. 15765–15766, (May 2021).

[3] 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, pp. 635–640, (2006).

[4] 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*, pp. 111–120. Springer, (2010).

[5] Helena Keinänen, 'Simulated annealing for multi-agent coalition formation', in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, pp. 30–39. Springer, (2009).

[6] 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).

[7] 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).

[8] Fredrik Präntare and Fredrik Heintz, 'An anytime algorithm for simultaneous coalition structure generation and assignment', in *PRIMA 2018: Principles and Practice of Multi-Agent Systems*, eds., Tim Miller, Nir Oren, Yuko Sakurai, Itsuki Noda, Bastin Tony Roy Savarimuthu, and Tran Cao Son, pp. 158–174, Cham, (2018). Springer International Publishing.

[9] Fredrik Präntare and Fredrik Heintz, 'An anytime algorithm for optimal simultaneous coalition structure generation and assignment', *Autonomous Agents and Multi-Agent Systems*, **34**(1), 1–31, (2020).

[10] Fredrik Präntare and Fredrik Heintz, 'Hybrid dynamic programming for simultaneous coalition structure generation and assignment', in *PRIMA 2020: Principles and Practice of Multi-Agent Systems*, eds., Takahiro Uchiya, Quan Bai, and Iván Marsá Maestre, pp. 19–33, Cham, (2021). Springer International Publishing.

[11] Fredrik Präntare, Herman Appelgren, and Fredrik Heintz, 'Anytime heuristic and monte carlo methods for large-scale simultaneous coalition structure generation and assignment', *Proceedings of the AAAI Conference on Artificial Intelligence*, **35**(13), 11317–11324, (May 2021).

[12] 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*, pp. 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems, (2008).

[13] Talal Rahwan, Sarvapali D Ramchurn, Viet Dung Dang, and Nicholas R Jennings, 'Near-optimal anytime coalition structure generation', in *Proc. of IJCAI*, volume 7, pp. 2365–2371, (2007).

[14] 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).

[15] 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).

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

[17] Sandip Sen and Partha Sarathi Dutta, 'Searching for optimal coalition structures', in *Proceedings Fourth International Conference on MultiAgent Systems*, pp. 287–292. IEEE, (2000).

[18] 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)*, pp. 1075–1082, (2021).

[19] Tadao Takaoka, 'An o(1) time algorithm for generating multiset permutations', in *Algorithms and Computation*, pp. 237–246, Berlin, Heidelberg, (1999). Springer Berlin Heidelberg.

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

[21] Aaron Williams, 'Loopless generation of multiset permutations using a constant number of variables by prefix shifts', in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, p. 987–996, USA, (2009). Society for Industrial and Applied Mathematics.

[22] Feng Wu and Sarvapali D Ramchurn, 'Monte-carlo tree search for scalable coalition formation', in *Proc. of IJCAI*, pp. 407–413, (2020).

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