Group Decision and Negotiation (2022) 31:747–768
https://doi.org/10.1007/s10726-022-09781-2

An Abortion Based Search Method for Optimal Coalition Structure Generation

Changder Narayan[1] · Aknine Samir[2] · Dutta Animesh[1]

Accepted: 20 March 2022 / Published online: 15 April 2022

Abstract
The Coalition Structure Generation (CSG) problem is a partitioning of a set of agents into exhaustive and disjoint subsets to maximize social welfare. This NP-complete problem arises in many practical scenarios. Prominent examples are included in the field of transportation, e-Commerce, distributed sensor networks, and others. The fastest exact algorithm to solve the CSG problem is ODP-IP, which is a hybrid version of two previously established algorithms, namely Improved Dynamic Programming (IDP) and IP. In this paper, we show that the ODP-IP algorithm performs many redundant operations. To improve ODP-IP, we propose a faster abortion mechanism to speed up IP's search. Our abortion mechanism decides at runtime which of the IP's operations are redundant to skip them. Then, we propose a modified version of IDP (named MIDP) and an improved version of IP (named IIP). Based on these two improved algorithms, we develop a hybrid version (MIDP-IIP) to solve the CSG problem. After a detailed description of the new algorithm MIDP-IIP, an experimental comparison is conducted against ODP-IP. Our analysis shows that MIDP-IIP performs fewer operations than ODP-IP. In addition, MIDP-IIP reduced significantly many problem instances running times (11–37%).

Keywords Coalition structure generation · Dynamic programming · Coalition formation

✉ Changder Narayan
narayan.changder@gmail.com

Dutta Animesh
samir.aknine@univ-lyon1.fr

[1] National Institute of Technology Durgapur, Durgapur, West Bengal, India

[2] LIRIS Lab., Lyon 1 University, Lyon, France

1 Introduction

Agents form coalitions when they cannot achieve certain goals individually but are achievable when they “team up" with other agents. The resulting teams are called *coalitions*. In line with the long-standing literature in cooperative game theory, we assume the existence of a characteristic function v that maps each coalition to a real-valued utility measure. The Coalition Structure Generation (CSG) problem consists of identifying the optimal partitioning of a set of agents, such that the sum of the utilities obtained by applying the characteristic function to each partition is maximized. The CSG problem is intractable because of its combinatorial explosion of the search space[1]. Coalition formation can be applied to many real-world problems. Agents can form a coalition to satisfy particular market niches (Norman et al. 2004). In distributed sensor networks, different sensors can form a coalition and work together to track their target of interest (Dang et al. 2006). Delivery companies may agree together and can build coalitions to make their profit by reducing transportation costs (Sandhlom and Lesser 1997). In electricity grids, agents create an intelligent and robust power supply network which manages the use of energy resources (Davidson et al. 2009; Dimeas and Hatziargyriou 2007; Kok et al. 2010; Vytelingum et al. 2010a, b). In e-commerce, customers build coalitions to take advantage of price discount through bulk purchasing (Li et al. 2010). In small cell networks, small cells can mitigate the co-tier interference within a coalition and thus increase the system capacity (Yang et al. 2016). Coalition formation can also be used for information gathering, where several information servers form coalitions to answer queries (Klusch and Shehory 1996).

This paper considers the CSG problem in Characteristic Function Games (CFGs). In CFG, each coalition ($\mathcal{C}$) is a non-empty subset of agents. Given n agents there are $2^n - 1$ coalitions. Hence, in the CSG problem, given a set of $2^n - 1$ coalitions, each associated with a value (positive or negative), we have to find a maximal valued disjoint set of coalitions with the same union as the whole set. Since the inputs of the CSG problem are the $2^n - 1$ coalitions, there is no hope to devise an exact algorithm for the CSG problem in polynomial time. Indeed, any exact algorithm must inspect all the $2^n - 1$ coalitions in order to find the optimal solution. The fastest exact algorithm for the optimal CSG problem is ODP-IP (Michalak et al. 2016) algorithm with worst-case time complexity $O(3^n)$.

In this work, we design a new algorithm called MIDP-IIP that builds upon the earlier algorithm ODP-IP (Michalak et al. 2016). To date, the fastest algorithm to solve the CSG problem is ODP-IP. ODP-IP considers strength of IDP and IP algorithms, and makes a positive synergy between IDP and IP. IDP and IP run in parallel, and as soon as any one of them returns the final result, ODP-IP stops. Practically, an improvement of either one (IDP or IP) in turn may improve ODP-IP algorithm. However, it is challenging to make IDP or IP algorithm more faster in practice because the IDP and IP algorithms belong to NPC problems (Michalak et al. 2016).

[1] For n agents, we have total $\Omega((\frac{n}{\ln n})^n)$ search spaces, that is, the n^{th} Bell number (Berend and Tassa 2010)

This paper proposes a new method to solve the CSG problem and makes the following contributions:

- We propose a new abortion mechanism to speed-up the IP's search.
- We propose an improved version of ODP-IP, named MIDP-IIP. We prove that MIDP-IIP performs less operations than ODP-IP.

The rest of the paper is organized as follows: Related works and the optimal CSG problem formulation are given in Sects. 2 and 3 respectively. Section 4 delineates the ODP-IP algorithm. Section 5 details the new MIDP-IIP algorithm, while Sects. 6 and 7 describe the experimental evaluation and propose some conclusions.

2 Related Works

To date, approaches to solve the CSG problem range from mixed-integer programming to branch and bound techniques (Rahwan et al. 2009) through dynamic programming (DP) (Yun Yeh and A, 1986). The first complete systematic DP algorithm for optimal CSG was proposed by Yun Yeh (1986) for complete set partitioning problem. The algorithm proposed by Yun Yeh (1986), solves the optimization problem by breaking it into smaller subproblems. Each small subproblem is solved recursively and finally the results of the subproblems are merged and thus generate the final solution. Given n agents, DP algorithm returns the optimal result with a reasonable time complexity $O(3^n)$ but it requires to evaluate all the possible coalitions.

The Improved Dynamic Programming (IDP) algorithm (Rahwan and Jennings 2008) with time complexity $O(3^n)$ is an improved version of the DP algorithm. The IDP algorithm efficiently manages to avoid a large number of redundant calculations made by DP approach in Yun Yeh (1986), Rothkopf et al. (1998). Recently, Cruz et al. (2017) described a novel technique to identify the most frequent operations in DP and IDP search tree and proposed an optimized version by distributing the processing into multiple threads using some multi-threading techniques. The authors in Cruz et al. (2017) reported that a speed-up over ten times was obtained.

The DP algorithms is the lowest worst time complexity algorithm. The main drawback of DP is that they could not produce a solution until the algorithms complete the entire execution. On the other hand an anytime algorithm is one that returns a solution even if the algorithm terminates prematurely. This anytime property makes it more robust against failures than DP algorithms. The result quality in anytime algorithms increases monotonically as the computation time increases. The anytime algorithm based on integer partition called IP algorithm (Rahwan et al. 2012) has been shown superior to DP algorithms for many popular coalition value distributions.

The Optimal Dynamic Programming (ODP) algorithm (Michalak et al. 2016) achieves a further improvement over IDP by using a bound on the size of the coalitions to be explored. Michalak et al. (2016) also proposed a hybrid version of ODP and IP (Rahwan et al. 2009) called ODP-IP and showed empirically that it is faster than other algorithms. The Inclusion-Exclusion algorithm proposed by Björklund et al. (2009) was tested in practice by Michalak et al. (2016) and the authors found that the growth rate resembles $O(6^n)$, not $O(2^n)$.

All the existing dynamic programming techniques solve the optimal CSG problem with a worst-case time complexity $O(3^n)$ for n number of agents. Hence, such methods are not effective when the time required to produce the optimal solution is larger than the time available to the agents. In multi-agent settings without hard time limits, ODP-IP is efficient to solve many real-life problem instances. However, in other circumstances for large number of agents with a strict deadline and a short execution time, the problem would naturally become too hard to be tackled by optimal algorithms. In these cases, we need other alternative approaches. Here heuristics approximate algorithms come to the rescue.

Many meta-heuristic algorithms have been proposed to tackle the CSG problem. In Shehory and Kraus (1995a, b, 1998), the authors proposed algorithms for coalition formation for task allocation. Another heuristic algorithm based on genetic algorithm was proposed in Sen and Dutta (2000). A few years later, Keinänen proposed an algorithm (Keinänen 2009) for the CSG problem based on simulated annealing. In Di Mauro et al. (2010), the authors addressed the CSG problem by proposing a solution approach based on GRASP. In Hussin and Fatima (2016), the authors showed that Tabu search generates better quality solutions than simulated annealing for coalition games in characteristic function form and in partition function form. The problem with meta-heuristic algorithms is that they do not guarantee if an optimal solution is ever found nor do they provide any guarantee on the quality of the achieved solutions.

In all the algorithms discussed so far, the main focus was on maximizing the social welfare, where agents consider every possible subset of agents as a potential coalition. However, this general assumption is not feasible in some realistic scenarios. To tackle this issue, a variation of CSG problems arises. To name a few, (Skibski et al. 2016) presented different algorithms to solve non-utilitarian coalition structure generation: A balanced CSG and an egalitarian CSG. In a balanced CSG, the goal is to minimize the difference between the values of smallest and largest agent utilities, whereas, in an egalitarian CSG, the goal is to find the coalition structure with the maximal value of smallest agent utility. In another scenario, certain agents may be prohibited from being in the same coalition or the coalition structure may be required to be composed of coalitions of the same size. To overcome these situations, (Rahwan et al. 2011) proposed a Constrained Coalition Formation (CCF) framework for multi-agent systems. Liu et al. (2016) developed stochastic search algorithms for coalitional skill games (CSGs), where it is difficult to calculate the value of a coalition.

3 CSG Problem Formulation

Let $\mathcal{A}$ be the set of agents $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$, n the number of agents in $\mathcal{A}$. We denote any coalition $\mathcal{C} = \{a_1, a_2, \dots, a_l\}$ as a coalition of agents a_1, $a_2, \dots$, and a_l, where $l \leq n$. Let v be a characteristic function, v assigns a real value $v(\mathcal{C})$ to each coalition $\mathcal{C}$ (i.e. $v(\mathcal{C})$). Formally, $v : 2^{\mathcal{A}} \rightarrow \mathbb{R}$.

A coalition structure ($\mathcal{CS}$) over $\mathcal{A}$ is a partitioning of $\mathcal{A}$ into a set of disjoint coalitions $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$. In other words, $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$ satisfies the following constraints: 1) $\mathcal{C}_i$, $\mathcal{C}_j \neq \emptyset$, $i, j \in \{1, 2, \dots, k\}$. 2) $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$, for all $i \neq j$. 3) $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{A}$.

Definition 1 Given a characteristic function v which maps each coalition $\mathcal{C}$ to a utility value, the value of any coalition structure $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$ is defined by $v(CS) = \sum_{\mathcal{C}_i \in CS}(v(\mathcal{C}_i))$.

The optimal solution of CSG is an optimal coalition structure $\mathcal{CS}^* \in \Pi^{\mathcal{A}}$, where $\Pi^{\mathcal{A}}$ denotes the set of all coalition structures over $\mathcal{A}$. Thus, $\mathcal{CS}^* = \arg\max_{\mathcal{CS} \in \Pi^{\mathcal{A}}} v(\mathcal{CS})$. The CSG problem is then the problem of finding such CS^*. Note that $\{a_1, a_2, \dots, a_n\}$ and $\{1, 2, \dots, n\}$ are used interchangeably throughout this paper.

4 ODP-IP Algorithm

The ODP-IP algorithm is a hybrid version of ODP and IP algorithms. ODP part of ODP-IP is based on the IDP algorithm. IDP and DP algorithms follow the same working principle, but IDP is an improved version of the DP algorithm. DP is an exact algorithm for computing the optimal coalition structure. Let, P_t be the partition table, $P_t(\mathcal{C})$ stores one optimal partition of each coalition $\mathcal{C}$ (cf. Fig. 1). There can be more than one optimal partition of a coalition $\mathcal{C}$, $P_t(\mathcal{C})$ stores any one of them (cf. Fig. 1). Let V_t be the optimal value table, $V_t(\mathcal{C})$ stores the optimal value of the coalition $\mathcal{C}$. DP produces two tables P_t and V_t using the below recursion (cf. Eq. 1).

Let $\mathcal{C}\simeq\simeq = \left\{\mathcal{C}' | \mathcal{C}\simeq \subset \mathcal{C} \text{ and } 0 \leq |\mathcal{C}\simeq| \leq \frac{|\mathcal{C}|}{2}\right\}$, table V_t for each coalition $\mathcal{C}$ is constructed as follows:

$$V_t(\mathcal{C}) = \begin{cases} v(\mathcal{C}) & \text{if } |\mathcal{C}| = 1 \\ \arg\max_{\mathcal{C}' \in \mathcal{C}''} \{V_t(\mathcal{C}') + V_t(\mathcal{C} \setminus \mathcal{C}')\} & \text{otherwise} \end{cases} \tag{1}$$

To evaluate the coalitions, DP starts by evaluating all possible splits of every possible coalition of size 2, and then gradually increases in size by 1 unit till size becomes n and completes tables P_t and V_t for each evaluated coalition $\mathcal{C}$.

Having described how DP operates, now we detail IDP algorithm (Rahwan and Jennings 2008). The main idea in IDP algorithm is to avoid the evaluation of some splitting operations in the DP network, without losing the guarantees of finding the optimal coalition structure. In Rahwan and Jennings (2008), the authors proved that, given n agents, IDP algorithm does not evaluate any of the possible ways of splitting a coalition of size s, where $s \in \{\lfloor \frac{2n}{3} \rfloor + 1, \dots, n-1\}$, without losing the guarantees of finding the optimal coalition structure. In particular, they showed that it is necessary to evaluate the splits of a coalition $\mathcal{C}$ of size c into two coalitions of sizes c' and c'', where (c', c'') is in: $dep(c) = \{(c', c'') \in \mathbb{N}^2 : (c' \geq c'') \wedge (c' + c'' = c) \wedge [(c' \leq n - c' - c'') \vee (c = n)]\}$.

Size	Coalition (C)	$v(C)$	Splitting	Optimal partition P_t	Optimal value V_t
1	({1})	24	$V_t(\{1\}) = 24$	$(\{1\})$	24
	({2})	35	$V_t(\{2\}) = 35$	$(\{2\})$	35
	({3})	20	$V_t(\{3\}) = 20$	$(\{3\})$	20
	({4})	41	$V_t(\{4\}) = 41$	$(\{4\})$	41
2	({1,2})	59	$v(\{1,2\}) = 59,\ \boxed{V_t(\{1\}) + V_t(\{2\}) = 59}$	$(\{1\})(\{2\})$	59
	({1,3})	43	$v(\{1,3\}) = 43,\ \boxed{V_t(\{1\}) + V_t(\{3\}) = 44}$	$(\{1\})(\{3\})$	44
	({1,4})	79	$\boxed{v(\{1,4\}) = 79,}\ V_t(\{1\}) + V_t(\{4\}) = 65$	$(\{1,4\})$	79
	({2,3})	52	$v(\{2,3\}) = 52,\ \boxed{V_t(\{2\}) + V_t(\{3\}) = 55}$	$(\{2\})(\{3\})$	55
	({2,4})	65	$v(\{2,4\}) = 65,\ \boxed{V_t(\{2\}) + V_t(\{4\}) = 76}$	$(\{2\})(\{4\})$	76
	({3,4})	75	$\boxed{v(\{3,4\}) = 75,}\ V_t(\{3\}) + V_t(\{4\}) = 61$	$(\{3,4\})$	75
3	({1,2,3})	85	$\boxed{v(\{1,2,3\}) = 85,}\ V_t(\{1\}) + V_t(\{2,3\}) = 79$ $V_t(\{2\}) + V_t(\{1,3\}) = 79, V_t(\{3\}) + V_t(\{1,2\}) = 79$	$(\{1,2,3\})$	85
	({1,2,4})	110	$\boxed{v(\{1,2,4\}) = 110,}\ V_t(\{1\}) + V_t(\{2,4\}) = 100$ $V_t(\{2\}) + V_t(\{1,4\}) = 114, V_t(\{4\}) + V_t(\{1,2\}) = 100$	$(\{2\})(\{1,4\})$	114
	({1,3,4})	92	$\boxed{v(\{1,3,4\}) = 92,}\ V_t(\{1\}) + V_t(\{3,4\}) = 99$ $V_t(\{3\}) + V_t(\{1,4\}) = 99,\ V_t(\{4\}) + V_t(\{1,3\}) = 85$	$(\{1\})(\{3,4\})$	99
	({2,3,4})	108	$\boxed{v(\{2,3,4\}) = 108,}\ V_t(\{2\}) + V_t(\{3,4\}) = 110$ $V_t(\{3\}) + V_t(\{2,4\}) = 96, V_t(\{4\}) + V_t(\{2,3\}) = 96$	$(\{2\})(\{3,4\})$	110
4	({1,2,3,4})	131	$v(\{1,2,3,4\}) = 131, V_t(\{1\}) + V_t(\{2,3,4\}) = 134$ $V_t(\{2\}) + V_t(\{1,3,4\}) = 134, V_t(\{3\}) + V_t(\{1,2,4\}) = 134$ $V_t(\{4\}) + V_t(\{1,2,3\}) = 126, \boxed{V_t(\{1,2\}) + V_t(\{3,4\}) = 134}$ $V_t(\{1,3\}) + V_t(\{2,4\}) = 120, V_t(\{1,4\}) + V_t(\{2,3\}) = 134$	$(\{1,2\})(\{3,4\})$	134

Fig. 1 Working principle of DP and IDP algorithms computing the tables P_t and V_t, given four agents $A = \{1, 2, 3, 4\}$, and a characteristic function v. With arrowed line in column P_t, we highlight the path leading to the optimal result. Locally optimal results are shaded in small rectangualr box. The dark shade indicates splits that are considered by DP, but not IDP

As in Pawłowski et al. (2014), $dep(c)$ indicates the dependencies between different coalition sizes. Based on this formula, IDP only evaluates the partitionning of the coalition $\mathcal{C}$ of size c into two coalitions of sizes c' and c'', where $(c', c'') \in dep(c)$. In Rahwan and Jennings (2008), the authors proved that $dep(c) = \emptyset$ for all $c \in \{\lfloor \frac{2n}{3} \rfloor + 1, \ldots, n-1\}$.

Before further discussion on ODP-IP, we explain how the IP algorithm (Rahwan et al. 2007) works. The IP algorithm uses a novel representation of the search space. It divides the whole search space of the CSG problem into different subspaces based on the size of the coalitions each subspace contains. The IP algorithm treats each node in the integer partition graph (cf. Fig. 3) as a subspace. For example, in the case of 4 agents, the possible integer partitions are [4], [1, 3], [2, 2], [1, 1, 2] and [1, 1, 1, 1], and each of these represents a subspace containing all the coalition structures within which the coalition sizes match the parts of the integer partition. For example, the subspace [1, 1, 2] represents all the coalition structures within which two disjoint coalitions are of size 1, and one disjoint coalition is of size 2. It is possible to compute an upper bound and a lower bound on the values of all the coalition structures in each subspace. These bounds are then used to prune the subspaces which do not have any potential of containing an optimal coalition structure. Out of all the remaining potential subspaces, IP sorts all these subspaces according to the upper bound values and starts searching them one by one[2] until all the subspaces are searched.

We now show how IDP is combined with the IP algorithm and make a hybrid ODP-IP algorithm (Michalak et al. 2016). In the ODP-IP algorithm, IDP and IP run in parallel and terminate as soon as any one of the IDP or IP returns the final result. To detail the operation of ODP-IP, we introduce the following integer partition graph for ten agents (cf. Fig. 2).

Suppose IDP completes the evaluation of all the coalitions of size 2, 3, 4. Figure 2 shows that all the red colored subspaces in the integer partition graph have already been explored by IDP. Hence, IP will consider only white colored subspaces.[3]

The preceding discussion implies that ODP-IP divides the search between IDP and IP and makes ODP-IP algorithm faster in practice. In our work, we propose a Modified Improved Dynamic Programming algorithm (MIDP) and an Improved IP (IIP) algorithm in place of ODP and IP and thus making new hybrid algorithm MIDP-IIP. The basic idea of MIDP-IIP is to maintain a partition table P_t for all the coalitions. To solve any subspace, the IIP algorithm uses this partition table P_t to speed-up the search process so that the overall runtime of MIDP-IIP is minimized. We provide the details of the MIDP-IIP algorithm in the next section.

5 MIDP-IIP Algorithm

In MIDP-IIP algorithm, MIDP and IIP run in parallel, and IIP uses the information provided by MIDP algorithm to speed up the search process. In the next section, we first provide a detailed description of the MIDP algorithm, then we explain the IIP algorithm.

[2] A detailed presentation of IP is proposed in Rahwan et al. (2007).

[3] For the techniques used in IP algorithm in the hybrid ODP-IP algorithm, we recommend readers to refer to the novel techniques to improve IP's search process using the branch and bound techniques proposed by Michalak et al. (2016).

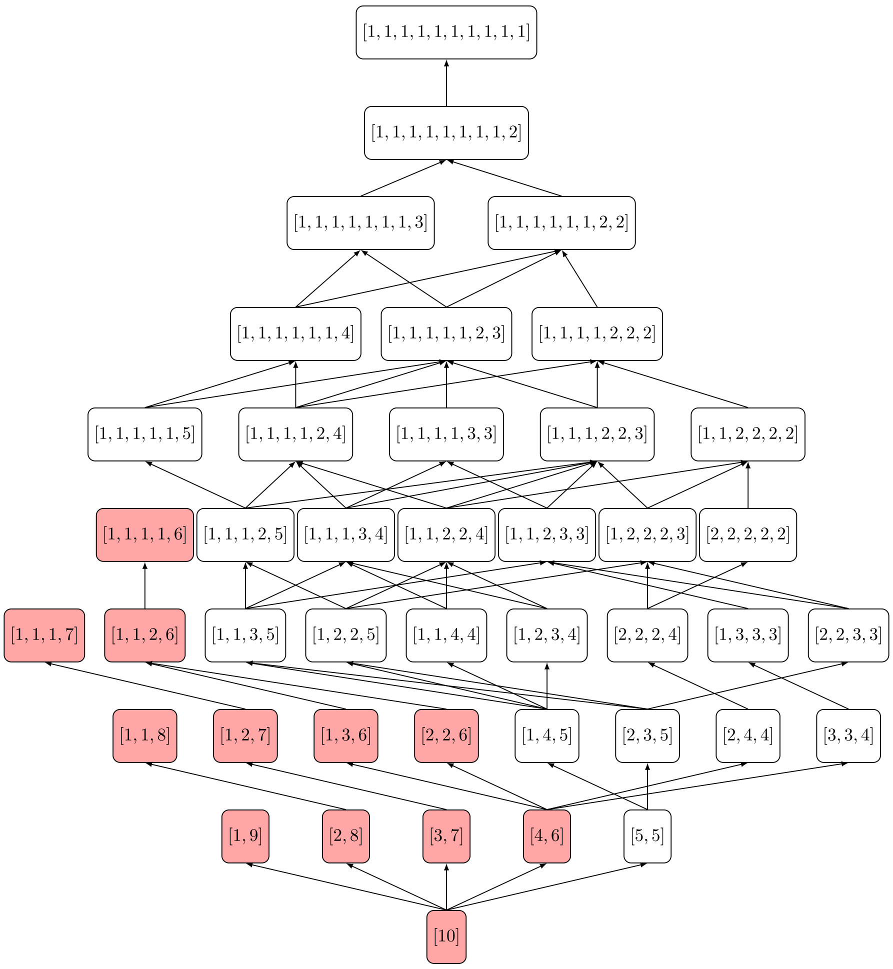
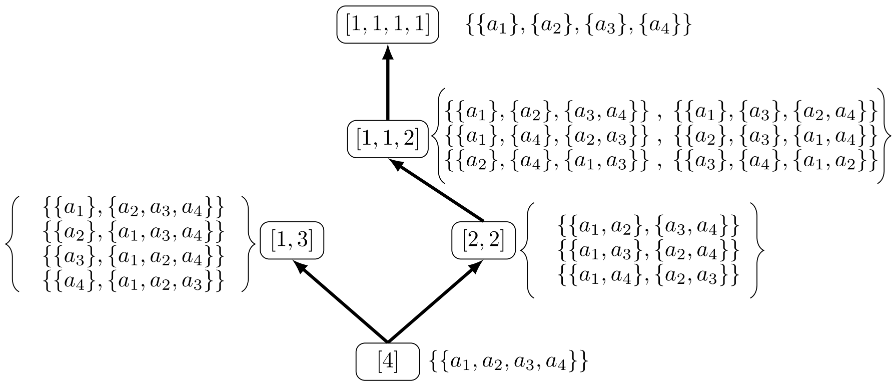


Fig. 2 Searched subspaces after evaluation of all the coalitions of size 4, given 10 agents. The red colored subspaces are fully searched by IDP, whereas the white colored subspaces are not yet searched

5.1 MIDP Algorithm

The working principle of MIDP algorithm is shown in algorithm 1. MIDP algorithm stores the optimal partition of each evaluated coalition in the partition table P_t. To explain the difference between the IDP and MIDP, let's take the example in Fig. 1. IDP evaluates all the coalitions of size $s \in \{2, 3, \dots, \lfloor \frac{2n}{3} \rfloor\}$ and then evaluates the grand coalition. In Fig. 1, IDP evaluates all coalitions of size 2 and then evaluates



[1,1,1,1] {{a1},{a2},{a3},{a4}}
[1,1,2] {{a1},{a2},{a3,a4}} , {{a1},{a3},{a2,a4}} {{a1},{a4},{a2,a3}} , {{a2},{a3},{a1,a4}} {{a2},{a4},{a1,a3}} , {{a3},{a4},{a1,a2}}
[1,3] {{a1},{a2,a3,a4}} {{a2},{a1,a3,a4}} {{a3},{a1,a2,a4}} {{a4},{a1,a2,a3}}
[2,2] {{a1,a2},{a3,a4}} {{a1,a3},{a2,a4}} {{a1,a4},{a2,a3}}
[4] {{a1,a2,a3,a4}}

Fig. 3 The integer partition graph for 4 agents. IDP does not need to evaluate any coalition of size three. Each node represents a unique integer partition of 4

the grand coalition $\mathcal{A} = \{1, 2, 3, 4\}$. Recall that, IDP does not store the optimal partition of the coalitions (Rahwan and Jennings 2008). Hence, there is no instance or immediate access to any coalition's optimal partition. IDP can get the optimal solution without using any partition table in different ways. For example in Fig. 1, IDP checks that the grand coalition $\mathcal{A} = \{1, 2, 3, 4\}$ is optimally partitioned into two coalitions $\{1, 2\}$ and $\{3,4\}$. Next IDP re-evaluates the coalitions $\{1, 2\}$, $\{3, 4\}$ and finds that the coalition $\{1, 2\}$ is optimally partitioned into the coalitions $\{1\}$ and $\{2\}$, whereas it is not beneficial to split the coalition $\{3, 4\}$. By doing so, IDP finds that the optimal coalition structure is $\{\{1\}\{2\}\{3, 4\}\}$ with the value 134.

On the other hand, MIDP stores the optimal partition of all the evaluated coalitions in the partition table P_t. By using the P_t table, MIDP finds the optimal partition of the grand coalition $\mathcal{A} = \{1, 2, 3, 4\}$ in the same way as DP does (shown in Fig. 1). Next, we show how IP's search procedure can go faster using the information stored in the P_t table.

Algorithm 1 Modified IDP algorithm.

Input: Set of all possible non-empty subsets of n agents $(2^n - 1)$. The value of a coalition $\mathcal{C}$ is $v(\mathcal{C})$. If no $v(\mathcal{C})$ is specified then $v(\mathcal{C}) = 0$. $\mathcal{A}$ denotes a set of n agents.

Output: The optimal coalition structure $\mathcal{CS}^*$ and its value.

```
1:  for $i = 1$ to $n$ do
2:      for each $\mathcal{C}$, $\mathcal{C} \subseteq \mathcal{A}$, where $|\mathcal{C}| = i$ do
3:          $V_t(\mathcal{C}) \leftarrow v(\mathcal{C})$
4:          $P_t(\mathcal{C}) \leftarrow \mathcal{C}$
5:      end for
6:  end for
7:  for $i = 2$ to $\lfloor \frac{2n}{3} \rfloor$ do
8:      for each $\mathcal{C}$, $\mathcal{C} \subset \mathcal{A}$, where $|\mathcal{C}| = i$ do
9:          for each $\mathcal{C}'$, $\mathcal{C}' \subset \mathcal{C}$, where $1 \leq |\mathcal{C}'| \leq \frac{|\mathcal{C}|}{2}$ do
10:             if $V_t(\mathcal{C}') + V_t(\mathcal{C} \setminus \mathcal{C}') > V_t(\mathcal{C})$ then
11:                 $V_t(\mathcal{C}) \leftarrow V_t(\mathcal{C}') + V_t(\mathcal{C} \setminus \mathcal{C}')$
12:                 $P_t(\mathcal{C}) \leftarrow \{\mathcal{C}', \mathcal{C} \setminus \mathcal{C}'\}$
13:             end if
14:         end for
15:     end for
16: end for
17: for each $\mathcal{C}'$, $\mathcal{C}' \subset \mathcal{A}$, where $1 \leq |\mathcal{C}'| \leq \frac{|\mathcal{A}|}{2}$ do
18:     if $V_t(\mathcal{C}') + V_t(\mathcal{A} \setminus \mathcal{C}') > v(\mathcal{A})$ then
19:         $V_t(\mathcal{C}) \leftarrow V_t(\mathcal{C}') + V_t(\mathcal{A} \setminus \mathcal{C}')$
20:         $P_t(\mathcal{C}) \leftarrow \{\mathcal{C}', \mathcal{A} \setminus \mathcal{C}'\}$
21:     end if
22: end for
23: $\mathcal{CS}^* \leftarrow \{\mathcal{A}\}$
24: for each $\mathcal{C}$, $\mathcal{C} \in \mathcal{CS}^*$ do
25:     if $P_t(\mathcal{C}) \neq \{\mathcal{C}\}$ then
26:         $\mathcal{CS}^* \leftarrow (\mathcal{CS}^* \setminus \mathcal{C} \cup P_t(\mathcal{C}))$
27:     end if
28: end for
29: Return $\mathcal{CS}^*$, $V_t(\mathcal{CS}^*)$
```

Theorem 1 *Given n agents, MIDP runs in $O(3^n)$ time.*

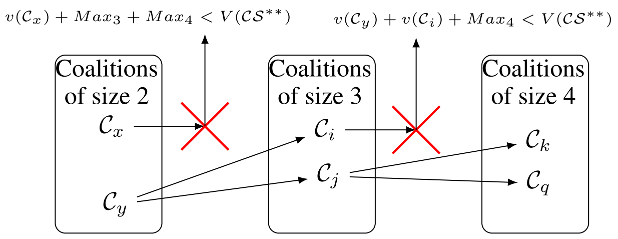


Fig. 4 Given 9 agents, illustration of IP's branch and bound technique when searching the subspace [2, 3, 4] using the inequality 2. Here, the algorithm proceeds according to the tree depth starting from depth $d = 1$. First in depth $d = 1$, the IP algorithm recognises that the coalition structure containing the coalition $\mathcal{C}_x$ cannot be optimal. Next, in depth $d = 2$, the IP algorithm finds that the coalition structures containing the coalitions $\mathcal{C}_y$, and $\mathcal{C}_i$ cannot be optimal. So, in both cases, IP does not dive deep into any of these branches of the search tree

Proof MIDP algorithm performs atmost the same amount of operations as IDP does. Recall that, IDP does not use the partition table P_t. Given n agents, IDP recomputes at most $2n - 1$ entries in the partition table for $2n - 1$ coalitions on-the-fly. Given the optimal values of a coalition $\mathcal{C}$, IDP computes $P_t(\mathcal{C})$ in time $2^{|\mathcal{C}|}$. Thus, IDP requires $(2n - 1) \times 2^n$ additional operations. The time complexity of IDP algorithm is $O(3^n)$. The extra steps MIDP performs is to store the optimal partitions of $2^n - 1$ coalitions in the partition table P_t but MIDP does not perform $(2n - 1) \times 2^n$ operations to recompute the entries in the partition table P_t. Hence, the total operations performed by MIDP is the operations performed by IDP plus the operations necessary to store $2^n - 1$ entries in the partition table P_t. So, total time complexity of MIDP is $maximum(O(3^n), O(2^n)) = O(3^n)$. □

5.2 Improved IP (IIP) Algorithm

In this section, first we recall the IP algorithm (Rahwan et al. 2007), and then we show how the partition table P_t produced by MIDP speeds up IP's search process. The IP algorithm is built on the integer partition-based representation (Rahwan et al. 2007) of the space of all possible coalition structures. Each integer partition represents one subspace. We can compute the maximum and the minimum possible valued coalition structures inside any subspace. Let, Max_i and Avg_i be the maximum and average values over all the coalitions of size $i \in \{1, 2, \ldots, n\}$. In Rahwan et al. (2009), the authors proved that by computing Avg_i for all $i \in \{1, 2, \ldots, n\}$, it is possible to compute the average value of the coalition structures in each subspace as follows:

Theorem 2 (Rahwan et al. Rahwan et al. (2009)). *Given n agents, for every integer partition I of n, let I(i) be the multiplicity of i in I. Then*

$$\frac{\sum_{\mathcal{CS}\in\Pi^{\mathcal{A}}} V(\mathcal{CS})}{|\Pi_I^{\mathcal{A}}|} = \sum_{i\in I} I(i) \times Avg_i$$

In Rahwan et al. (2009), the authors proved that it is possible to compute the upper and the lower bounds of the coalition structures in any subspace by using Max_i and Avg_i. More formally the upper and lower bounds of the subspaces corresponding to the integer partition I can be computed as follows: $UB_I = \sum_{i\in I} I(i) \times Max_i$ and $LB_I = \sum_{i\in I} I(i) \times Avg_i$. In the search process, IP algorithm prunes all the subspaces whose upper bound is less than the maximum lower bound over all the subspaces. Next, IP starts searching the subspaces based on the upper bound of the subspaces, i.e. the subspace with highest upper bound is searched first. Now we explain how IP searches any subspace in an efficient way. IP algorithm searches any subspace in a depth-first manner. Let's say IP is now searching a subspace $[i_1, i_2, \dots, i_k]$. IP algorithm first iterates over all the coalitions $\mathcal{C}^{i_1}$ of size i_1. Next for each coalition $\mathcal{C}_{x_1} \in \mathcal{C}^{i_1}$, IP iterates over all the coalitions $\mathcal{C}_{x_2} \in \mathcal{C}^{i_2}$ of size i_2 that do not overlap with $\mathcal{C}_{x_1}$. Similarly, IP iterates over all the coalitions $\mathcal{C}_{x_3} \in \mathcal{C}^{i_3}$ of size i_3 that do not overlap with the coalition $\mathcal{C}_{x_1} \cup \mathcal{C}_{x_2}$, and so on. This process is repeated until the last coalition of size i_k is picked. Using this process all the coalition structures in the subspace $[i_1, i_2, \dots, i_k]$ are searched. However, a straightforward approach generates repeated coalition structures if the multiplicity of any integer in the subspace is greater than one. Rahwan et al. (2009) detail how IP avoids such redundant operations. To speed up the search process, IP applies a branch-and-bound technique at every depth d in the search tree. Specifically, after generating d coalitions $\mathcal{C}_{x_1} \in \mathcal{C}^{i_1}, \dots, \mathcal{C}_{x_d} \in \mathcal{C}^{i_d}$, and before iterating over the next feasible coalitions of size $d+1, \dots, k$, IP checks the inequality 2.

$$\sum_{i=1}^{d} v(\mathcal{C}_i) + \sum_{i=d+1}^{k} Max_i < V(\mathcal{CS}^{**}) \tag{2}$$

Let $V(\mathcal{CS}^{**})$ denotes the best coalition structure found by the IP algorithm at any point in time. If the inequality 2 holds, then all the coalition structures composed of the coalitions $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_d$ can be skipped because the coalition structures containing the coalitions $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_d$ in the subspace $[i_1, i_2, \dots, i_k]$ will always generate a coalition structure value less than $V(\mathcal{CS}^{**})$ and cannot be part of the optimal coalition structure. Hence, IP can skip all such coalition structures during its search. To clarify this process, let us now consider the following example. Given 9 agents, Fig. 4 gives the graphical representation of how IP searches the subspaces [2, 3, 4].

Furthermore, the authors in Michalak et al. (2016) have shown with the help of IDP algorithm that some coalition structures can still be pruned even if they are promising. IDP evaluates all the coalitions sequentially and stores the optimal value of the coalitions in a table w. Then, every time IP reaches a certain depth d, it performs the following operation:

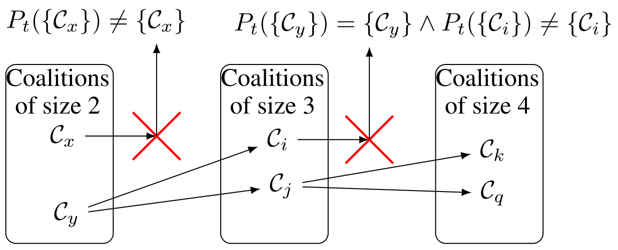


Fig. 5 Given 9 agents, illustration of IIP's branch and bound technique when searching the subspace [2, 3, 4] using the inequality 5. Here, the algorithm proceeds according to the tree depth starting from depth $d = 1$. First in depth $d = 1$, the IIP algorithm recognises that the coalition structure containing the coalition $\mathcal{C}_x$ cannot be optimal. Next, in depth $d = 2$, the IIP algorithm finds that the coalition structures containing the coalition $\mathcal{C}_i$ cannot be optimal. So, in both cases IIP does not dive deep into any of these branches of the search tree

$$\sum_{i=1}^{d} w(\mathcal{C}_i) > \sum_{j=1}^{d} v(\mathcal{C}_j) \tag{3}$$

$$w(\mathcal{C}_d) > v(\mathcal{C}_d) \tag{4}$$

If the inequality 3 holds then any coalition structure containing the coalition $\{\mathcal{C}_1, \ldots, \mathcal{C}_d\}$ cannot be the optimal coalition structure in the subspace $[i_1, i_2, \ldots, i_k]$ and all such coalition structures can be skipped. Similarly, if the inequality 4 holds then the coalition $\{\mathcal{C}_d\}$ is not part of any optimal coalition structure in the subspace $[i_1, i_2, \ldots, i_k]$. Hence, every coalition structure containing the coalition $\{\mathcal{C}_1, \ldots, \mathcal{C}_d\}$ can be skipped during IP's search. To use the strength of IDP in IP's search, the authors in Michalak et al. (2016) used the same table w[4] used by IP and IDP proposed in Michalak et al. (2016).

Michalak et al. (2016) proved that, given n agents, the IP algorithm's worst-case runtime is $O(n^n)$. In the worst scenario, IP can end up by searching all the coalition structures in each subspace. Next, we describe an alternate way to improve IP's branch and bound technique using the partition table P_t produced by MIDP algorithm.

5.3 Abortion Mechanism

In this section, we describe the abortion mechanism to speed up IP's search. Consider the example in Fig. 1. Suppose at any point in time, IP algorithm starts searching all the coalition structures in the subspace [2, 1, 1]. Let's say that the value of the best coalition structure found by the algorithm so far is

[4] More details are provided in Michalak et al. (2016).

$V(\mathcal{CS}^{**}) = v(\{1\}) + v(\{2\}) + v(\{3\}) + v(\{4\}) = 24 + 35 + 20 + 41 = 120$. When searching the subspace [2, 1, 1] by IP algorithm, IP checks that the value of the coalition $\{1, 2\}$ of size 2 is 59. Next, IP adds two maximum values of the coalitions of size 1, which are 35 and 41 with 59. Hence, the value in the left side of inequality 2 is 59+41+35=135 which is greater than the value of the best coalition structure $V(\mathcal{CS}^{**}) = 120$ found by the algorithm so far. Hence, IP will proceed to search the next feasible coalitions of size 1 and so on. In this case, the inequality 2 does not help IP to skip any coalition structure. Similarly, for $d = 1$, inequalities 3 and 4 do not help IP because in this case the value of the coalition $\{1, 2\}$ is same (i.e. 59) before and after evaluation of the coalition $\{1, 2\}$.

In IIP, we impose a new rule in the branch and bound search technique of IP and call the new IP algorithm as *Improved IP* (IIP) algorithm. Specifically, after generating d coalitions $\mathcal{C}_1 \in \mathcal{C}^{i_1}, \dots, \mathcal{C}_d \in \mathcal{C}^{i_d}$, and before iterating over the next feasible coalitions of the size $d + 1, \dots, k$, IIP checks the inequality 5.

$$P_t(\{\mathcal{C}_d\}) \neq \{\mathcal{C}_d\} \tag{5}$$

Now, if inequality 5 holds, then all the coalition structures composed of coalitions $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_d$ can be skipped during IIP's search, because the coalition $\mathcal{C}_d$ cannot be part of the optimal coalition structure as the coalition $\mathcal{C}_d$ is stored in the optimal partition table P_t in two disjoint coalitions.

In IIP's search technique, first the inequality 5 is applied. If it holds, then there is no need to check the inequalities 2, 3 and 4 .

To clarify this process, suppose IIP is searching a subspace $[i_1, i_2, \dots, i_k]$. If any coalition structure in the subspace $[i_1, i_2, \dots, i_k]$ has the potential to become the optimal coalition structure, then the resulting coalition structure "must be in" $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$, where $|\mathcal{C}_x| = i_x$. Suppose, MIDP finished evaluating all the coalitions of size i_2 and in this situation any coalition structure in the subspace $[i_1, i_2, \dots, i_k]$ will contain the coalition $\mathcal{C}_2$ of size i_2 if $P_t(\{\mathcal{C}_2\}) = \{\mathcal{C}_2\}$. If $P_t(\{\mathcal{C}_2\}) \neq \{\mathcal{C}_2\}$, then the coalition $\mathcal{C}_2$ is not part of the optimal coalition structure. To better understand the process, let us now consider the following example. Given 9 agents, Fig. 5 illustrates how IIP searches the subspace [2, 3, 4].

Inside the subspace [2, 3, 4], any coalition structure will have only three coalitions, where the first, the second, and the third coalitions are of size 2, 3, and 4 respectively. Figure 5 shows that $P_t(\{\mathcal{C}_x\}) \neq \{\mathcal{C}_x\}$. That means, MIDP evaluated the coalition $\mathcal{C}_x$ and optimally stores the coalition $\mathcal{C}_x$ in two disjoint coalitions. Hence, the size of the coalition $\mathcal{C}_x$ is no more 2. So, all the coalition structures containing the coalition $\mathcal{C}_x$ inside the subspace [2, 3, 4] can be skipped.

Our approach to speed-up IIP's search process involves modifying only the back end of the IIP's branch and bound technique using the partition table P_t produced by MIDP. If the inequality 5 does not hold, then all the existing IP's branch and bound techniques are used in IIP's search. In our technique, to speed up IIP's search we proceed as follows:

- For each coalition $\{\mathcal{C}_d\}$, IIP checks the inequality 5. If the inequality 5 holds, then the coalition $\{\mathcal{C}_d\}$ is not part of the optimal coalition structure. In this

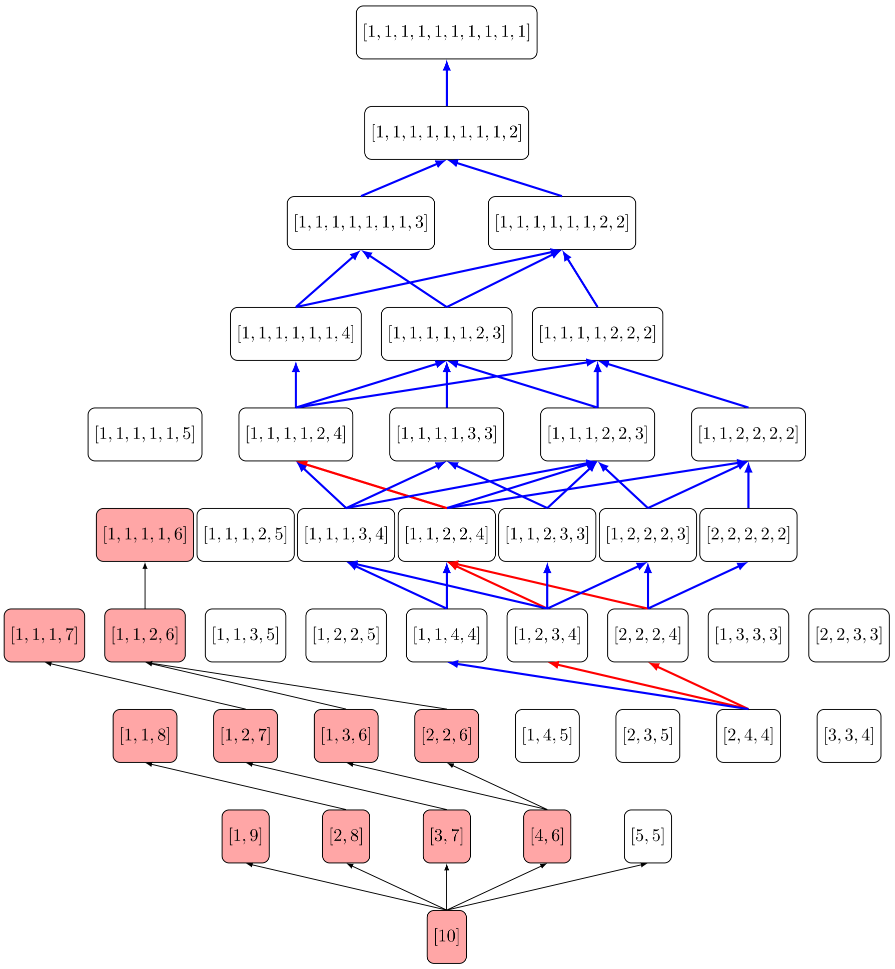


Fig. 6 IIP searching multiple subspaces simultaneously after MIDP has evaluated all the coalitions of size $s \in (2, 3, 4)$. Several subspaces are searched simultaneously by splitting exactly one coalition as shown in red edges. IIP can search more subspaces simultaneously by splitting multiple coalitions as shown in both red and blue edges

case, the abortion mechanism is used to suspend IIP's search process in this branch of the search tree.

- If IIP finds that the inequality 5 does not hold, then IIP uses all the IP's branch and bound techniques we discussed earlier.

The improvement in IIP's branch and bound technique exploits the optimal partition of the coalitions stored in the partition table P_t. To better understand the abortion mechanism used in IIP's search process, let's take a numerical example.

Example 1 Consider the example shown in Fig. 1. Suppose IIP is searching the subspace [2, 1, 1]. Let's assume that MIDP already finished evaluating all the coalitions of size 2. MIDP stores the best partition of every coalition in the partition table P_t. When IIP is searching the coalition structures in the subspace [2, 1, 1], IIP checks how the coalition is stored in the table P_t. For example, when IIP encounters the coalition $\{1, 2\}$ of size 2 and checks that this coalition is stored in two disjoint coalitions, IIP can skip all the coalition structures containing the coalition $\{1, 2\}$ because IIP knows that it is searching the subspace [2, 1, 1]. But the coalition $\{1, 2\}$ is stored into two disjoint coalitions, so the coalition structure containing the coalition $\{1, 2\}$ cannot be the optimal coalition structure in the subspace [2, 1, 1].

Theorem 3 *Given n agents, MIDP-IIP runs in $O(3^n)$ time.*

Proof In MIDP-IIP, MIDP and IIP run in parallel and return the optimal solution as soon as one of MIDP or IIP returns the optimal result. Worst case running times of MIDP and IIP algorithms are $O(3^n)$ and $O(n^n)$. Hence, the time complexity of MIDP-IIP is $minimum\,(O(3^n), O(n^n)) = O(3^n)$. □

Theorem 4 *Given n agents, MIDP-IIP always finds the optimal solution.*

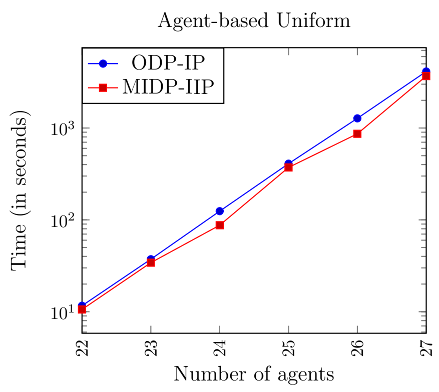
Proof Each node in the integer partition graph corresponds to a subspace consisting of all coalition structures in which the sizes of the coalitions match the parts of the integer partition.

Let us fix any particular node P in the integer partition graph, which contains the optimal coalition structure $\mathcal{CS}^*$. The MIDP-IIP is a hybrid version of MIDP and IIP. We prove the correctness of MIDP-IIP by using our established algorithms MIDP and IIP.

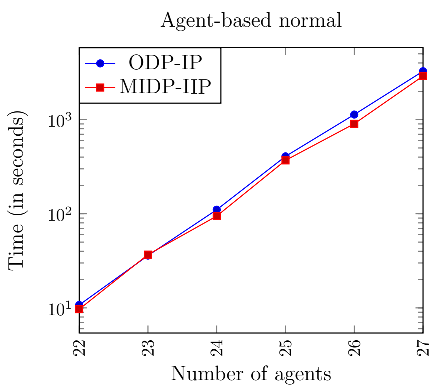
The optimal coalition structure $\mathcal{CS}^*$ is found if MIDP reaches the node P from the bottom node in the integer partition graph, or if IIP finishes searching all the feasible coalition structures associated with the node P. MIDP-IIP stops if all the subspaces are searched by MIDP or IIP. It follows that the node P containing the optimal coalition structure is always found by MIDP or IIP or by both of them. □

5.4 Searching Multiple Subspaces Simultaneously

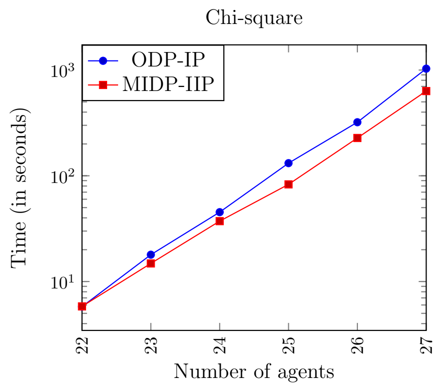
In this section, first we show how IIP searches multiple subspaces simultaneously and avoids repeating certain operations. Then, we detail how IIP combines this technique with MIDP algorithm using the partition table P_t. Recall that IIP searches each subspace in a depth-first manner. IIP picks the next subspace based on the upper bound of the subspaces. Assume that IIP is now searching the subspace $I = [i_1, i_2, \dots, i_k]$ and at the same time MIDP already finished the evaluation of all the coalitions of size $s \in \{2, 3, \dots s^*\}$ (s^* is the maximum coalition size evaluated by MIDP. s^* is decided at runtime). Now, IIP performs the following steps:



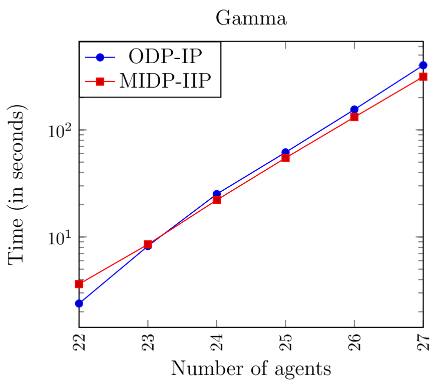
(a)



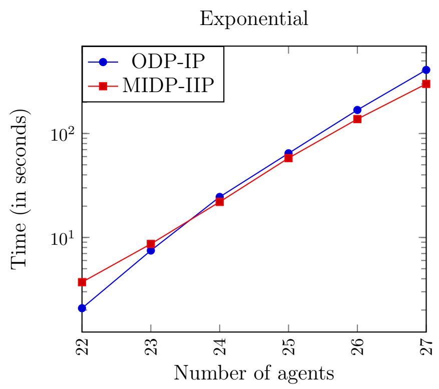
(b)



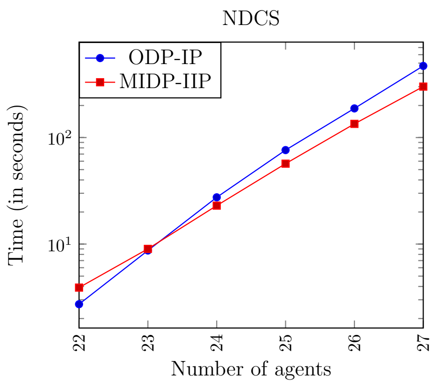
(c)



(d)



(e)



(f)

Fig. 7 Time performance of ODP-IP vs. MIDP-IIP in the interval 22–27 agents. Here, time is measured in seconds and plotted *on a log scale*. The time difference is more visible in the range of 22–27 agents

(a) *Finding reachable subspaces* $\mathcal{X}^*$: IIP finds the set of all reachable subspaces from the subspace I using the paths already evaluated by MIDP in the integer partition graph (cf. Fig. 6). For instance, given $I = [2, 4, 4]$, the set of all reachable subspaces $\mathcal{X}^*$ from I is shown in red and blue edges in Fig. 6. Here multiple subspaces can be searched simultaneously by partitioning the integers 2 and 4.

However, in Michalak et al. (2016), the authors proved that practically it is faster to partition only one integer because the difficulty with splitting multiple integers is that it may interfere with the branch-and-bound technique. Hence, to explore these multiple subspaces, IIP will always split a single integer. In this example, if we split a single integer (i.e. 4) then all the reachable subspaces from I consist of all integer partitions that are reachable only through the red edges shown in the Fig. 6.

(b) *Identifying integers to split:* In MIDP-IIP algorithm, IIP always splits a single integer to explore more subspaces. To identify the single integer in the subspace I, IIP picks an integer $x \in I$ so that splitting x allows for reaching the largest number of integer partitions in $\mathcal{X}^*$. For instance, given $I = [2, 4, 4]$, if exactly one integer is split, all the subspaces reachable from the subspace [2, 4, 4] are shown through the red edges in Fig. 6. That means, by searching the subspace [2, 4, 4], IIP simulteneously searches extra subspaces [1, 2, 3, 4], [2, 2, 2, 4], [1, 1, 2, 2, 4], and [1, 1, 1, 1, 2, 4].

(c) *Changing the order of integers in I*: IIP rearranges the integers in I by placing the integer x in the first position followed by the other integers. For example, suppose IIP is searching the subspace $[i_1, i_2, \dots, i_k]$ and the integer to split in this subspace is i_k. After rearranging the integers, I becomes $[i_k, i_1, \dots, i_{k-1}]$.

(d) *Searching the subspace* $[i_1, i_2, \dots, i_k]$: When searching the subspace $[i_1, i_2, \dots, i_k]$, IIP algorithm applies the following steps for every coalition $\mathcal{C} \in \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$:

$$P_t(\{\mathcal{C}\}) \neq \{\mathcal{C}\}, \quad \text{where } |\mathcal{C}| \neq i_k \tag{6}$$

During IIP's search at every depth d in a branch of the search tree, IIP checks the inequality $P_t(\{\mathcal{C}_d\}) \neq \{\mathcal{C}_d\}$. If this inequality holds, then all the coalition structures containing the coalition $\mathcal{C}_d$ in the subspace $[i_1, i_2, \dots, i_k]$ can be skipped safely and IIP stops searching further in that branch of the search tree. However, if this inequality is not true, then IIP checks the inequalities 2, 3, and 4 .

5.4.1 Comparison Between IP and IIP

Both IP and IIP are able to enumerate all the feasible coalition structures. But, in practice they work in different ways to search a branch of a search tree. Suppose the subspace $[i_1, i_2, \dots, i_k]$ is being searched by IP and IIP, then in the case of IIP, the coalitions of size i_1 are enumerated. If any coalition $\mathcal{C}_{i_1}$ of size i_1 is stored in the partition table in two disjoint coalitions then the coalition $\mathcal{C}_{i_1}$ cannot be part of the optimal coalition structure. In this case, IIP needs only a single operation to check the partition table. On the other hand, IP checks the inequalities 2, 3, and 4 . In Sect. 5.3, we have shown when IP fails to stop the search process in a branch of the search tree and how IIP can stop this search process.

Table 1 Effectiveness of ODP-IP and MIDP-IIP. The table shows runtime (in seconds) for 27 agents, taken for each coalition value distribution as an average over 50 runs

Distribution	Time in seconds		
	ODP-IP time (t_1)	MIDP-IIP time (t_2)	Difference $t_1 - t_2$
Agent-based uniform	4126	3677	449
Agent-based normal	3269	2904	365
Chi-square	1030	632	398
NDCS	470	300	170
Exponential	409	300	109
Gamma	401	313	88
Sum over all distributions	9705	8126	1579

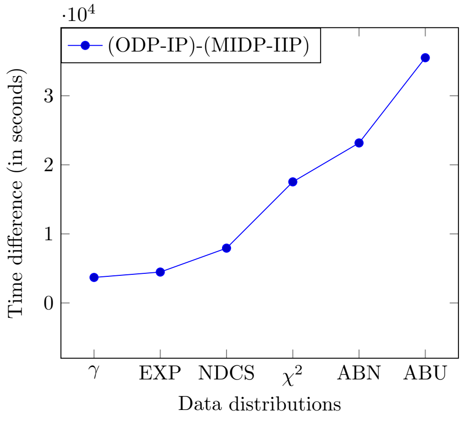


Fig. 8 Time difference in seconds of ODP-IP and MIDP-IIP. For each distribution we have considered 1125 problem instances in the range of 5 to 27 agents

6 Performance Evaluation

Having described our algorithm, we now detail its evaluation and show its effectiveness by comparing it against ODP-IP. For ODP-IP, we used the code provided by the authors of ODP-IP (Michalak et al. 2016). Both algorithms were implemented in Java, and the experiments were run on an Intel (R) Xeon (R) CPU E7-4830 v3, running at 2.10 GHz with 160 GB of RAM. We took an average of 50 tests for each point on the Fig. 7 to ensure the error becomes very small. With this in mind, we considered the following distributions: agent-based uniform (Rahwan et al. 2012), agent-based normal (Michalak et al. 2016), beta, exponential, gamma, modified normal (Rahwan et al. 2012), modified uniform (Service and Adams 2010), Normally Distributed Coalition Structures (NDCS) (Rahwan et al. 2007), and uniform (Larson and Sandholm 2000) distributions. One new distribution we considered is the Chi-square distribution. In this distribution, the value of each coalition $\mathcal{C}$ is drawn from $v(\mathcal{C}) \sim \chi^2(\nu)$, where $\nu = |\mathcal{C}|$ is the degree of freedom. The rationale behind introducing the Chi-square distribution is by changing the degree of freedom $\nu = |\mathcal{C}|$ for different coalition sizes makes it possible that a coalition structure of any size can be optimal. If the degree of freedom ν is constant and $\nu > 90$ it approximates the normal distribution.

For each of the above distributions, we plotted the execution times of ODP-IP and MIDP-IIP given different numbers of agents (see Fig. 7). Here, time is measured in seconds. For each distribution and each number of agents, we took an average over multiple runs. As can be seen, running time is reduced significantly (11–37%) when compared MIDP-IIP with ODP-IP.

The experimental results show that MIDP-IIP algorithm performs well for many problem instances. In particular, we observe the following:

- Given 27 agents, with agent-based uniform, agent-based normal, gamma, exponential, NDCS, and Chi-square distributions, running time is reduced significantly by 10.88, 11.17, 21.95, 26.65, 36.17 and 36.64% respectively when compared with ODP-IP algorithm. In this class of problems, the inequality 5 works well and IIP algorithm does not use IP's branch and bound technique frequently.
- With beta, modified-uniform, normal, uniform, and modified-normal distributions MIDP-IIP and ODP-IP performance are almost the same. When compared MIDP-IIP with ODP-IP we found that in this class of problems, sometimes the inequality 5 works and sometimes IP's branch and bound technique works. When the inequality 5 works, IP's branch and bound technique is not used and when the inequality 5 does not work, IP's branch and bound technique is used by IIP algorithm.

The results in Table 1 show that there are data distributions in which MIDP-IIP gives more positive synergies. To better understand the positive side of new abortion mechanism, we have plotted the total time difference of MIDP-IIP and ODP-IP in Fig. 8. Here, the time is measured in seconds. For each distribution, we have taken 1125 problem instances in the range of 5 to 27 agents. Each point in this figure represents the difference of total time to solve 1125 problem instances by ODP-IP and MIDP-IIP. From the Fig. 8 it is clear that MIDP-IIP performance is better.

7 Conclusion

Coalition structure generation in multiagent systems is a well known hard problem. Precisely identifying the optimal coalition structure is a hard task. Different solution techniques to cope with this significant problem have been proposed. The current best known exact algorithm for CSG problem is ODP-IP (Michalak et al. 2016). In this paper, we have presented a new hybrid algorithm MIDP-IIP which extends IDP and IP algorithms. We tested both ODP-IP and MIDP-IIP algorithms over 11 different value distributions. The experimental results confirmed that MIDP-IIP outperforms ODP-IP for several distributions. Out of 11 distributions, MIDP-IIP outperforms ODP-IP significatly on 6 distributions. Given, 27 agents, in the case of agent-based uniform distribution, MIDP-IIP took 449 seconds less time as compared to ODP-IP. Our measurements show that improving IDP and IP algorithms improves the performance of optimal CSG algorithms upto great extent. When compared with ODP-IP, with agent-based uniform, agent-based normal, gamma, exponential, NDCS, and Chi-square distributions, running time is reduced significantly by 10.88, 11.17, 21.95, 26.65, 36.17 and 36.64% respectively.

Acknowledgements The research presented in this article is funded by "Visvesvaraya Ph.D. Scheme for Electronics & IT", Grant No: PhD-MLA/4(29)/2015-16.

References

Berend D, Tassa T (2010) Improved bounds on bell numbers and on moments of sums of random variables. Probab Math Stat 30(2):185–205

Björklund A, Husfeldt T, Koivisto M (2009) Set partitioning via inclusion-exclusion. SIAM J Comput 39(2):546–563

Cruz F, Espinosa A, Moure JC, Cerquides J, Rodriguez-Aguilar JA, Svensson K, Ramchurn SD (2017) Coalition structure generation problems: optimization and parallelization of the idp algorithm in multicore systems. Concurr Comput Pract Exp 29(5):e3969

Dang VD, Dash RK, Rogers A, Jennings NR (2006) Overlapping coalition formation for efficient data fusion in multi-sensor networks. AAAI Conf Artif Intell 6:635–640

Davidson E, Dolan M, McArthur S, Ault G (2009) The use of constraint programming for the autonomous management of power flows, in: Intelligent system applications to power systems, 2009. ISAP'09. 15th International Conference on, IEEE, pp. 1–7

Di Mauro N, Basile TM, Ferilli S, Esposito F (2010) Coalition structure generation with grasp. In: International conference on artificial intelligence: methodology, systems, and applications, Springer, pp 111–120

Dimeas AL, Hatziargyriou ND (2007) Agent based control of virtual power plants. In: Intelligent systems applications to power systems. ISAP 2007. International Conference on, IEEE, pp. 1–6

Hussin A, Fatima S (2016) Heuristic methods for optimal coalition structure generation. In: Multi-agent systems and agreement technologies, Springer, pp. 124–139

Keinänen H (2009) Simulated annealing for multi-agent coalition formation. In: KES international symposium on agent and multi-agent systems: technologies and applications, Springer, pp 30–39

Klusch M, Shehory O (1996) A polynomial kernel-oriented coalition algorithm for rational information agents, Tokoro, ed 157–164

Kok J, Scheepers M, Kamphuis I (2010) Intelligence in electricity networks for embedding renewables and distributed generation. In: Intelligent infrastructures, Springer, pp. 179–209

Larson KS, Sandholm TW (2000) Anytime coalition structure generation: an average case study. J Exp Theor Artif Intell 12(1):23–42

Li C, Sycara K, Scheller-Wolf A (2010) Combinatorial coalition formation for multi-item group-buying with heterogeneous customers. Decis Support Syst 49(1):1–13

Liu Y, Zhang G-F, Su Z-P, Yue F, Jiang J-G (2016) Using computational intelligence algorithms to solve the coalition structure generation problem in coalitional skill games. J Comput Sci Technol 31(6):1136–1150

Michalak T, Rahwan T, Elkind E, Wooldridge M, Jennings NR (2016) A hybrid exact algorithm for complete set partitioning. Artif Intell 230:14–50

Norman TJ, Preece A, Chalmers S, Jennings NR, Luck M, Dang VD, Nguyen TD, Deora V, Shao J, Gray WA et al (2004) Agent-based formation of virtual organisations. Knowl-Based Syst 17(2):103–111

Pawłowski K, Kurach K, Svensson K, Ramchurn S, Michalak TP, Rahwan T (2014) Coalition structure generation with the graphics processing unit. In: AAMAS, international foundation for autonomous agents and multiagent systems, pp. 293–300

Rahwan T, Ramchurn SD, Dang VD, Giovannucci A, Jennings NR (2007) Anytime optimal coalition structure generation. AAAI Conf Artif Intell 7:1184–1190

Rahwan T, Ramchurn SD, Dang VD, Jennings NR (2007) Near-optimal anytime coalition structure generation. Int Joint Conf Artif Intell 7:2365–2371

Rahwan T, Ramchurn SD, Jennings NR, Giovannucci A (2009) An anytime algorithm for optimal coalition structure generation. J Artif Intell Res 34:521–567

Rahwan T, Michalak TP, Elkind E, Faliszewski P, Sroka J, Wooldridge M, Jennings NR (2011) Constrained coalition formation. AAAI Conf Artif Intell 11:719–725

Rahwan T, Jennings NR (2008) An improved dynamic programming algorithm for coalition structure generation. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1417–1420

Rahwan T, Michalak TP, Jennings NR (2012) A hybrid algorithm for coalition structure generation., In: AAAI conference on artificial intelligence, pp 1443–1449

Rothkopf MH, Pekeč A, Harstad RM (1998) Computationally manageable combinational auctions. Manage Sci 44(8):1131–1147

Sandhlom TW, Lesser VR (1997) Coalitions among computationally bounded agents. Artif Intell 94(1):99–137

Sen S, Dutta PS (2000) Searching for optimal coalition structures. In: MultiAgent Systems, Proceedings. Fourth international conference on, IEEE, pp 287–292

Service TC, Adams JA (2010) Approximate coalition structure generation. In: Twenty-Fourth AAAI conference on artificial intelligence, pp 854–859

Shehory O, Kraus S (1998) Methods for task allocation via agent coalition formation. Artif Intell 101(1–2):165–200

Shehory O, Kraus S (1995) Coalition formation among autonomous agents: Strategies and complexity (preliminary report), Springer, pp. 55–72

Shehory O, Kraus S (1995) Task allocation via coalition formation among autonomous agents. In: International joint conference on artificial intelligence, pp 655–661

Skibski O, Michalewski H, Nagórko A, Michalak TP, Dowell AJ, Rahwan T, Wooldridge M (2016) Non-utilitarian coalition structure generation. In: European conference on artificial intelligence, pp 1738–1739

Vytelingum P, Ramchurn SD, Voice TD, Rogers A, Jennings NR (2010) Trading agents for the smart electricity grid. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems, international foundation for autonomous agents and multiagent systems, pp 897–904

Vytelingum P, Voice TD, Ramchurn SD, Rogers A, Jennings NR (2010) Agent-based micro-storage management for the smart grid. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems: volume 1, international foundation for autonomous agents and multiagent systems, pp 39–46

Yang G, Esmailpour A, Cao Y, Nasser N (2016) A novel coalitional structure generation algorithm for interference mitigation in small cell networks. Global Communications Conference (GLOBECOM). IEEE, IEEE, pp 1–4

Yun Yeh D (1986) A dynamic programming approach to the complete set partitioning problem. BIT Numer Math 26(4):467–474