Artificial Intelligence Review (2021) 54:397–425
https://doi.org/10.1007/s10462-020-09850-5

Improving coalition structure search with an imperfect algorithm: analysis and evaluation results

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

Published online: 23 June 2020

Abstract
Optimal Coalition Structure Generation (CSG) is a significant research problem in multi-agent systems that remains difficult to solve. This problem has many important applications in transportation, eCommerce, distributed sensor networks and others. The CSG problem is NP-complete and finding the optimal result for n agents needs to check $O(n^n)$ possible partitions. The ODP–IP algorithm (Michalak et al. in Artif Intell 230:14–50, 2016) achieves the current lowest worst-case time complexity of $O(3^n)$. In the light of its high computational time complexity, we devise an Imperfect Dynamic Programming (ImDP) algorithm for the CSG problem with runtime $O(n2^n)$ given n agents. *Imperfect algorithm* means that there are some contrived inputs for which the algorithm fails to give the optimal result. We benchmarked ImDP against ODP–IP and proved its efficiency. Experimental results confirmed that ImDP algorithm performance is better for several data distributions, and for some it improves dramatically ODP–IP. For example, given 27 agents, with ImDP for agent-based uniform distribution time gain is 91% (i.e. 49 min).

Keyword Coalition structure generation · Dynamic programming · Coalition formation · Imperfect algorithm

1 Introduction

Agents form coalitions when they find that they cannot achieve certain goals, that can be accomplished when they "team up" with other agents with complementary capabilities. The resulting teams are called *coalitions*. Several applications use coalition formation methods. 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

A preliminary version of this paper have been accepted in the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19) Student Abstract and Poster Program (SA-19).

✉ Narayan Changder
narayan.changder@gmail.com

[1] Department of Computer Science and Engineering, National Institute of Technology, Durgapur, India

[2] LIRIS Laboratory, Claude Bernard University Lyon 1, Lyon, France

track their target of interest (Dang et al. 2006). Delivery companies may agree together and build coalitions to make their profit by reducing transportation costs (Sandhlom and Lesser 1997). 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. Cooperative game theory studies CSG problem in a characteristic function form, where each coalition is associated with a positive value. In the CSG problem, 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 positive value, we have to find a maximal valued disjoint set of coalitions with the same union as the whole set

In other words, the coalition structure generation 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.

There is a vast amount of literature on the CSG problem [see e.g. Rahwan et al. (2015) for a survey on this problem]. In this context, Yun Yeh (1986) developed a dynamic programming algorithm to solve the complete set partitioning problem. A few years later, Rothkopf et al. (1998) described another similar algorithm for solving the winner determination problem in combinatorial auctions. Solution approaches in Yun Yeh (1986), Rothkopf et al. (1998) are based on Dynamic Programming (DP) with time complexity $O(3^n)$ and are directly applicable to the optimal CSG problem. The Improved Dynamic Programming (IDP) algorithm (Rahwan and Jennings 2008a) with time complexity $O(3^n)$ is an improved version of the dynamic programming algorithm. The main idea in IDP is to avoid some evaluations in the dynamic programming network, without losing the guarantees of finding the optimal coalition structure. An anytime algorithm, called IP (a tree-search algorithm), was developed by Rahwan et al. (2009). Later a modified version of IDP reported on a new method which runs IDP and IP in parallel (Rahwan et al. 2012).

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. In Michalak et al. (2016), authors also proposed a hybrid version of ODP and IP - called ODP–IP and showed empirically that it is faster than other algorithms. Finally, the Inclusion-Exclusion algorithm proposed by Björklund et al. (2009) was tested in practice by Michalak et al. (2016) and authors found "the growth rate resembles $O(6^n)$, not $O(2^n)$". 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.

Numerous anytime CSG algorithms operate on the space of all coalition structures, between $\Omega(n^n)$ to $O(n^n)$. In Sandholm et al. (1999), authors proposed the first anytime algorithm for CSG with worst-case guarantees on the solution quality. The algorithm proposed by Dang and Jennings (2004) empirically generates tighter quality guarantees than Sandholm et al. (1999). Other proposals involve anytime algorithms that search the space of coalition structure graphs in different ways (Rahwan and Jennings 2008b; Rahwan et al. 2009). In Rahwan et al. (2009), authors used an entirely new representation of coalition structures search space based on integer partitions and shows empirically that this approach (called the anytime IP algorithm) generates higher-quality solutions than any other previous anytime algorithm. The algorithm in Rahwan and Jennings (2008b), called IDP–IP, avoids the weaknesses of IDP and IP by combining the positive aspects of IDP and IP. In Service and Adams (2011), Service and Adams (2010), authors furthered the idea of dynamic programming in different ways to make CSG algorithm anytime.

Many metaheuristic algorithms have been proposed to tackle the CSG problem. In Shehory and Kraus (1995a), Shehory and Kraus (1995b), Shehory and Kraus (1998), 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), authors addressed a solution approach based on GRASP. The problem with metaheuristic algorithms is that they do not guarantee if an optimal solution is ever found nor do they provide any guaranty on the quality of the solutions achieved.

The CSG problem over graph restricted games has been introduced by Myerson (1977). In this setting, given a graph $G = (\mathcal{V}, \mathcal{E})$, the agents are represented by the vertex set $\mathcal{V}$ and the edges can be interpreted as communication channels, or trust relationships, which facilitate the cooperation. A coalition $\mathcal{C}$ in the graph G is feasible iff it induces a connected subgraph of G. A modified version of IDP algorithm for graph restricted games is proposed in Voice et al. (2012), the authors called this IDP based algorithm DyCE. DyCE only considers a split of a coalition $\mathcal{C}$ if $\mathcal{C}'$ is feasible in $\{\mathcal{C}', \mathcal{C} \setminus \mathcal{C}'\}$. The DyPE (Vinyals et al. 2013) algorithm imposes a hierarchical structure over the set of agents. DyPE solves the CSG problem faster than both IDP and DyCE in a range of graph structures. An alternative algorithm was proposed in Bistaffa et al. (2014), Bistaffa et al. (2017), based on edge contraction. The process follows two steps: i) removing an edge e from the graph G, and ii) merging the two nodes that were previously joined by the edge e. Since every node represents an agent (i.e., a singleton coalition), "merging the two nodes" corresponds to merging the two coalitions that were represented by those nodes.

Practically ODP–IP (Michalak et al. 2016) algorithm runs faster for wide variety of problem instances, but there are problems for which ODP–IP is unable to produce exact result faster. For this type of problems if an efficient algorithm can be found to solve most of the CSG problem instances except a few instances, then it might be a practical method. To deal with this challenge, we define a new imperfect algorithm (Karp 1983) called ImDP. Karp stated that (Karp 1983) the criteria of correctness and worst-case efficiency are particularly inapplicable to the class of NP-hard combinatorial problems. There is strong circumstantial evidence, although no conclusive proof, that these problems are intractable, in the sense that no correct algorithm for such a problem can run within a polynomial time bound. If this folk belief about the intractability of NP-hard problems proves correct, then every algorithm for such a problem must inevitably be imperfect: there will be some inputs for which the algorithm either runs too long or fails to give a correct result. Nevertheless, such imperfect algorithms can be useful if they do not fail too often and especially if the failure is detectable. One way to validate or compare imperfect algorithms for NP-hard combinatorial problems is simply to run them on typical instances and see how often they fail. Against the research aims outlined above, this paper makes the following contributions to the coalition structure generation problem.

- We propose a novel imperfect dynamic programming algorithm, called ImDP for the CSG problem.
- We analyze how the symmetry and geometry introduced properties play an important role for CSG by using two merge functions.
- We prove theoretically that ImDP algorithm's time complexity is $O(n2^n)$.

Throughout this paper, $\mathcal{A}$ will denote the set of agents, n the number of agents, $\mathcal{C}$ a coalition, v the input table (i.e. $v(\mathcal{C})$ is the value of coalition $\mathcal{C}$), and P_t the partition table (i.e. $P_t(\mathcal{C})$ stores one optimal partition of coalition $\mathcal{C}$. There can be more than one optimal partition of

a coalition $\mathcal{C}$, $P_t(\mathcal{C})$ stores any one of them), and V_t the optimal value table (i.e. $V_t(\mathcal{C})$ stores the optimal value of coalition $\mathcal{C}$).

The rest of the paper is organized as follows: Section two describes the optimal CSG problem. Section three delineates the proposed imperfect algorithm (ImDP) and the two merge functions used in proposed imperfect algorithm. Sections four describes the time complexity of ImDP algorithm. Section five details the comparison between ImDP and ODP–IP and provides the results of the experimental evaluation. Finally section six proposes a conclusion.

2 The optimal CSG problem formulation

Let $\mathcal{A}$ be the set of agents $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, n the number of agents in $\mathcal{A}$. We denote any coalition $\mathcal{C} = \{a_1, a_2, \ldots, a_l\}$ as a coalition of agents $a_1, a_2, \ldots,$ 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, $v\colon 2^{\mathcal{A}} \rightarrow \mathbb{R}$.

A coalition structure ($\mathcal{CS}$) over $\mathcal{A}$ is a partitioning of $\mathcal{A}$ into a set of disjoint coalitions $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$. In other words, $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ satisfies the following constraints:

I. $\mathcal{C}_i, \mathcal{C}_j \neq \emptyset,\ i, j \in \{1, 2, \ldots, k\}$
II. $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$, for all $i \neq j$
III. $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{A}$.

Definition 1 Given a characteristic function v which maps each coalition $\mathcal{C}$ to a utility value, the value of any coalition structure $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ 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^*.

2.1 Dynamic programming

Let, P_t be a partition table, $P_t(\mathcal{C})$ stores one optimal partition of each coalition $\mathcal{C}$. There can be more than one optimal partition of a coalition $\mathcal{C}$, and $P_t(\mathcal{C})$ stores any one of them. Let V_t be an optimal value table, $V_t(\mathcal{C})$ stores the optimal value of the coalition $\mathcal{C}$. Dynamic programming offers an exact algorithm for computing the optimal coalition structure by constructing the tables P_t and V_t (cf. Fig. 1) using the below recursion.

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

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

For each coalition $\mathcal{C}$, $P_t(\mathcal{C})$ stores the corresponding partition of $V_t(C)$. After DP evaluates all possible coalitions, the optimal coalition structure $\mathcal{CS}^*$ is computed recursively from the partition table P_t (cf. Fig. 1). In our example, this is done by first setting

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}	47	$v[\{1,2\}] = 47, V_t\{1\} + V_t\{2\} = 59$	{1}{2}	59
	{1,3}	43	$v[\{1,3\}] = 43, V_t\{1\} + V_t\{3\} = 44$	{1}{3}	44
	{1,4}	79	$v[\{1,4\}] = 79, V_t\{1\} + V_t\{4\} = 65$	{1,4}	79
	{2,3}	52	$v[\{2,3\}] = 52, V_t\{2\} + V_t\{3\} = 55$	{2}{3}	55
	{2,4}	65	$v[\{2,4\}] = 65, V_t\{2\} + V_t\{4\} = 76$	{2}{4}	76
	{3,4}	75	$v[\{3,4\}] = 75, V_t\{3\} + V_t\{4\} = 61$	{3,4}	75
3	{1,2,3}	85	$v[\{1,2,3\}] = 85, V_t\{1\} + V_t\{2,3\} = 79$ $V_t\{2\} + V_t\{1,3\} = 79, V_t\{3\} + V_t\{1,2\} = 79$	{1,2,3}	85
	{1,2,4}	110	$v[\{1,2,4\}] = 110, V_t\{1\} + V_t\{2,4\} = 100$ $V_t\{2\} + V_t\{1,4\} = 114, V_t\{4\} + V_t\{1,2\} = 100$	{2}{1,4}	114
	{1,3,4}	92	$v[\{1,3,4\}] = 92, V_t\{1\} + V_t\{3,4\} = 99$ $V_t\{3\} + V_t\{1,4\} = 99, V_t\{4\} + V_t\{1,3\} = 85$	{1}{3,4}	99
	{2,3,4}	108	$v[\{2,3,4\}] = 108, V_t\{2\} + V_t\{3,4\} = 110$ $V_t\{3\} + V_t\{2,4\} = 96, V_t\{4\} + V_t\{2,3\} = 96$	{2}{3,4}	110
4	{1,2,3,4}	131	$v[\{1,2,3,4\}] = 131, V_t\{1\} + V_t\{2,3,4\} = 134$ $V_t\{2\} + V_t\{1,3,4\} = 134, V_t\{3\} + V_t\{1,2,4\} = 134$ $V_t\{4\} + V_t\{1,2,3\} = 126, V_t\{1,2\} + V_t\{3,4\} = 134$ $V_t\{1,3\} + V_t\{2,4\} = 120, V_t\{1,4\} + V_t\{2,3\} = 134$	{2}{1,3,4}	134

Fig. 1 Working principle of DP algorithm computing the tables P_t and V_t, given four agents $\mathcal{A} = \{1, 2, 3, 4\}$, and a characteristic function v. With blue color, we highlight the path leading to the optimal result. (Color figure online)

$\mathcal{CS}^* = P_t(\mathcal{A}) = P_t(\{1, 2, 3, 4\})$ and found that it is beneficial to split the coalition $\{1, 2, 3, 4\}$ into two coalitions $\{2\}$ and $\{1, 3, 4\}$. In the same way, by looking at $P_t(\{1, 3, 4\})$, it is found that it is beneficial to split the coalition $\{1, 3, 4\}$ into $\{1\}$ and $\{3, 4\}$. Finally, by looking at $P_t(\{3, 4\})$, it decides that it is beneficial to keep the coalition $\{3, 4\}$ as it is. Now, the optimal $\mathcal{CS}$ is $\{\{1\}\{2\}\{3, 4\}\}$ with a value of 134.

3 Imperfect algorithm

The fastest algorithm for the CSG problem is ODP–IP (Michalak et al. 2016) with a time complexity $O(3^n)$. However, if a faster algorithm can be found to solve most of the CSG

problem instances excepting a few instances, then it might be a practical method. To deal with this challenge, we define a new imperfect algorithm called ImDP. The ImDP algorithm is based on the symmetric relationship of combinatorics and two merge functions that we will describe in this section.

In the previous section, we showed that DP algorithm solves the CSG problem incrementally, that means DP solves all the coalitions of size x, then all the coalitions of size $x+1$ and so on. Our main aim is to reduce the workload to solve a coalition of size x using the partial enumeration. To evaluate a coalition $\mathcal{C}$ of size x using partial enumeration, ImDP algorithm uses the merge functions: $Merge_1$ and $Merge_2$.

To show how merge functions work, we focus on the evaluation of a single coalition. The following notation will be used to represent a coalition and its partitions. When evaluating the coalition $\{1, 2, 3\}$, it can be stored in the partition table P_t as $\{\{1\}\{2, 3\}\}$ or $\{\{2\}\{1, 3\}\}$ or $\{\{3\}\{1, 2\}\}$ or as the coalition itself, i.e. $\{1, 2, 3\}$ and the value corresponding to the partition will be stored in the V_t table. For example, in Fig. 1, the coalition $\{1, 2, 3\}$ is stored in the P_t table as $P_t(\{1, 2, 3\}) = \{1, 2, 3\}$, the coalition $\{2, 3, 4\}$ is stored as $P_t(\{2, 3, 4\}) = \{\{2\}\{3, 4\}\}$ and so on. Recall that the coalition $\{1, 2, 3\}$ cannot be stored in the partition table P_t as $\{1\}\{2\}\{3\}$ because DP algorithm splits every coalition in all possible partitions into two parts. More formally, any coalition can be stored in the partition table with any of its different possible partitions (into two parts or as the coalition itself). We will call each half a component. For example in $\{\{1\}\{2, 3\}\}$, we denote $\{1\}$ and $\{2, 3\}$ as two different components of the coalition $\{1, 2, 3\}$. In the following, we will detail each of ImDP merge functions.

3.1 *Merge*$_1$ function

We use $Merge_1$ function to evaluate[1] each coalition of size $4, 5, \ldots, \lceil \frac{n}{2} \rceil$. For any coalition $\mathcal{C}$, $Merge_1$ picks a single agent a_s and creates all partitions of the coalition $\mathcal{C}$ as $\{\{a_s\}\{C \setminus a_s\}\}$. For each $a_s \in \mathcal{C}$, $Merge_1$ is applied between $\{a_s\}$ and $\{C \setminus a_s\}$. As the coalitions are evaluated in size order, then the coalition $\{\mathcal{C} \setminus a_s\}$ has already been evaluated by $Merge_1$ before solving the coalition $\{a_s \cup \mathcal{C} \setminus a_s\}$. Hence, $Merge_1$ checks how the coalition $\{C \setminus a_s\}$ is stored in the P_t table. If the coalition $\{C \setminus a_s\}$ is stored in the P_t table as it is, then $Merge_1$ is not used. But, let us suppose the coalition $\{\mathcal{C} \setminus a_s\} = \mathcal{U}$ is stored in the P_t table with two different components as $\{\{u_1\}\{u_2\}\}$, where u_1 and u_2 are two different coalitions of any size[2]. Our assumption is that $\mathcal{U}$ is stored in the P_t table with two components. If this is not the case, it means the algorithm will still work but $Merge_1$ will not be applied. The Fig. 2 and Algorithm 1 show the detailed operation of $Merge_1$ function.

Formally, for a coalition $\mathcal{C} = \{a_1, a_2, \ldots, a_x\}$ of size x containing x agents, $Merge_1$ function partitions the coalition $\mathcal{C}$ into x ways as follows:

$$\begin{aligned} &\{a_1\}\{a_2, a_3, \ldots, a_x\} \\ &\{a_2\}\{a_1, a_3, \ldots, a_x\} \\ &\qquad\vdots \\ &\{a_x\}\{a_1, a_2, \ldots, a_{x-1}\} \end{aligned}$$

The main steps of this function can be summarized as follows:

[1] Use of $Merge_1$ function for coalitions of size 1, 2 and 3 is redundant. We prove this later in property 1.

[2] The set difference $\{\mathcal{C} \setminus a_s\}$ is defined as $\{\mathcal{C} \setminus a_s\} = \{x : x \in \mathcal{C} \text{ and } x \notin a_s\}$. We use $\mathcal{U}$ to denote coalition $\{\mathcal{C} \setminus a_s\}$

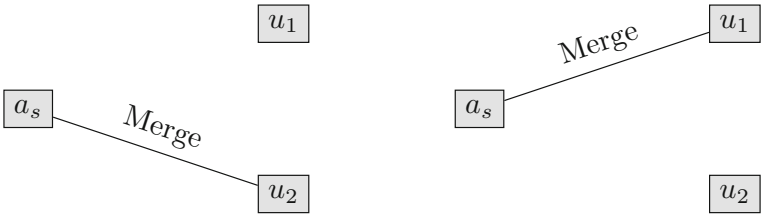


Fig. 2 $Merge_1$ function operates on a coalition $\{a_s \cup \mathcal{C} \setminus a_s\}$. In this figure, $Merge_1$ is applied between the coalitions a_s and $\{\mathcal{C} \setminus a_s\}$. It is assumed that the coalition $\{\mathcal{C} \setminus a_s\}$ is stored in the partition table P_t as $\{\{u_1\}\{u_2\}\}$. In the left part, $Merge_1$ is applied between the coalitions $\{a_s\}$ and $\{u_2\}$, it results with a new partition $\{\{a_s \cup u_2\}\{u_1\}\}$ of the coalition $\{a_s \cup \mathcal{C} \setminus a_s\}$. In the right part, $Merge_1$ is applied between the coalitions $\{a_s\}$ and $\{u_1\}$, it results with another new partition $\{\{a_s \cup u_1\}\{u_2\}\}$ of the coalition $\{a_s \cup \mathcal{C} \setminus a_s\}$

Algorithm 1 Merge function ($Merge_1$)

Input: Two coalitions $\{a_s\}$ and $\{\mathcal{C} \setminus a_s\}$, where $\{\mathcal{C} \setminus a_s\}$ is stored in partition table P_t with its two components $\mathcal{U} = \{\{u_1\}\{u_2\}\}$.
Output: Feasible maximum valued coalition structure CS_p over $\{a_s \cup \mathcal{U}\}$ and its value.

```
1: Value ← V_t({a_s}) + V_t(U)
2: CS_p ← {{a_s}{U}}
3: if |U| ≠ 1 and 3 < |U| then                    ▷ |U| = 1 means
                                                  ▷ coalition U is stored as it is.
4:     for Each component i in U = {{u_1}{u_2}} do
5:         if |i ∪ a_s| < |U| then
6:             if V_t(i ∪ a_s) + V_t(U \ i) > Value then
7:                 Value ← V_t(i ∪ a_s) + V_t(U \ i)
8:                 CS_p ← {{i ∪ a_s}{U \ i}}
9:             end if
10:        end if
11:    end for
12: end if
13: return CS_p, Value
```

Partition each coalition $\mathcal{C}$ of size x in all possible ways into two disjoint coalitions such that one half contains a singleton coalition $\{a_s\}$ and the other half is the coalition $\{\mathcal{C} \setminus a_s\}$ of size $x - 1$.

1. For all such partitions, $Merge_1$ checks how the coalition $\{\mathcal{C} \setminus a_s\}$ of size $x - 1$ is stored.
2. For the coalition $\{\mathcal{C} \setminus a_s\}$ of size $x - 1$, if it is stored into two components in the partition table P_t, then according to Algorithm 1, each component merges with the singleton coalition $\{a_s\}$, one at a time, creating a new partition of the coalition $\mathcal{C}$.

Theorem 1 *Partition of any coalition of size x takes $O(2^x)$ steps using DP algorithm. ImDP takes $O(2x)$ steps.*

Proof To evaluate a coalition $\mathcal{C}$ of size x, DP algorithm needs to consider all possible ways of partitioning the coalition $\mathcal{C}$ into two parts. Hence, it needs to check total $2^{x-1} - 1$ partitions. On the other hand, ImDP checks x number of partitions for a coalition of size x, each of them may create two extra partitions by using the $Merge_1$ function. So, total $2x$ number of partitions are needed by ImDP. □

Corollary 1 *To evaluate a coalition $\mathcal{C}$ of size n, DP algorithm needs to partition the coalition $\mathcal{C} = \{\{c_i\}\{c_j\}\}$ into all possible ways into two disjoint coalitions, where total number of coalitions in c_i is*

$$\binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \cdots + \binom{n}{\frac{n}{2}}$$

Neverthless, ImDP only evaluates $\binom{n}{1}$ number of splitting. □

Property 1 *The $Merge_1$ function is not effective for coalitions of size 1, 2 and 3.*

The $Merge_1$ function is used when there are two coalitions $\{a_s\}$ and $\{\mathcal{C} \setminus a_s\}$, where $\{a_s\}$ is a singleton coalition and $\{\mathcal{C} \setminus a_s\}$ is stored in the partition table P_t with two components. For any coalition $\mathcal{C}$ of size 1 or 2, it is not possible to apply the $Merge_1$ function because the size of the coalition $\{\mathcal{C} \setminus a_s\}$ becomes 0 or 1.

For any coalition $\mathcal{C}$ of size 3, the size of the coalition $\{\mathcal{C} \setminus a_s\}$ becomes 2 and if $Merge_1$ is used, it produces an already examined partition. □

Example 1 In Fig. 1, for coalition {1, 2, 3, 4}, $Merge_1$ will perform the following steps. For 4 agents ImDP would only evaluate coalitions of size upto 2. Here, we are evaluating a coalition of size 4 to explain how $Merge_1$ works.

$$\{1\}\{2,3,4\} \xrightarrow{Merge_1} \{1\}\{\{2\}\{3,4\}\} \rightarrow \{1,2\}\{3,4\}$$
$$\{2\}\{1,3,4\} \xrightarrow{Merge_1} \{2\}\{\{1\}\{3,4\}\} \rightarrow \{1,2\}\{3,4\}$$
$$\{3\}\{1,2,4\} \xrightarrow{Merge_1} \{3\}\{\{2\}\{1,4\}\} \rightarrow \{2,3\}\{1,4\}$$
$$\{4\}\{1,2,3\} \xrightarrow{Merge_1} \{4\}\{1,2,3\}$$

The last merge operation in the above example does not produce any new partition because the coalition {1, 2, 3} is stored as it is (see row 3 of Fig. 1 where $V_t(\{1,2,3\}) = 85$, i.e. the maximum). When performing merge operation for {1}{2, 3, 4}, it merges the coalition {1} with the component {2} of the coalition {2, 3, 4} and it does not merge the coalition {1} with the component {3, 4} of the coalition {2, 3, 4} because in that case it would create an extra partition {2}{1, 3, 4} of the coalition {1, 2, 3, 4} (this situation is already considered in the Algorithm 3, lines 15–23). As mentioned before, $Merge_1$ function works on the coalition {1, 2, 3, 4} by partitioning it as {1}{2, 3, 4}, {2}{1, 3, 4}, {3}{1, 2, 4} and {4}{1, 2, 3}. Hence, the partition {2}{1, 3, 4} is an extra partition.

3.2 *Merge*$_2$ function

Given two disjoint coalitions $\mathcal{X}$ and $\mathcal{Y}$ respectively of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$, where the coalitions $\mathcal{X}$ and $\mathcal{Y}$ are stored in the partition table P_t as $\{\{x_1\}\{x_2\}\}$ and $\{\{y_1\}\{y_2\}\}$. The principle of $Merge_2$ function is shown in Algorithm 2. $Merge_1$ function is used to evaluate all the coalitions of size 4, 5, ..., $\lceil \frac{n}{2} \rceil$, whereas $Merge_2$ function is used on $\mathcal{X}$ and $\mathcal{Y}$. Each component of $\mathcal{X}$ is merged with the components of $\mathcal{Y}$ once at a time. Figure 3a shows that first component $\{x_1\}$ of the coalition $\mathcal{X}$ is merged with the component $\{y_1\}$ of the coalition $\mathcal{Y}$ leaving the component $\{x_2\}$ of the coalition $\mathcal{X}$ and the component $\{y_2\}$ of the coalition $\mathcal{Y}$ unchanged, thus creating a coalition structure $\{\{x_1 \cup y_1\}\{x_2\}\{y_2\}\}$. In Fig. 3b, the component $\{x_1\}$ of the coalition $\mathcal{X}$ is merged with the component $\{y_2\}$ of $\mathcal{Y}$, leaving the components $\{x_2\}$ and $\{y_1\}$ unchanged, thus creating another coalition structure $\{\{x_1 \cup y_2\}\{x_2\}\{y_1\}\}$ and so on.

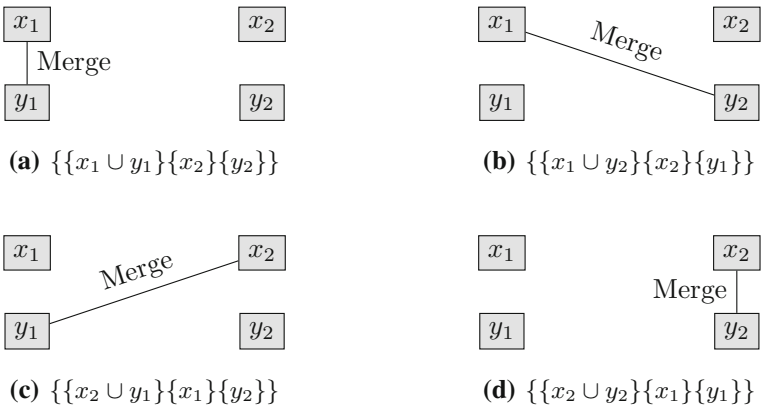


Fig. 3 $Merge_2$ function is applied on two coalitions $\mathcal{X}$ and $\mathcal{Y}$ of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$. Each figure shows how a new coalition structure is formed by using the $Merge_2$ function. In **a** the component $\{x_1\}$ of the coalition $\mathcal{X}$ is merged with the component $\{y_1\}$ of the coalition $\mathcal{Y}$, resulting a coalition structure $\{\{x_1 \cup y_1\}\{x_2\}\{y_2\}\}$, and so on

Algorithm 2 Merge function ($Merge_2$)

Input: Given two disjoint coalitions $\mathcal{X}$ and $\mathcal{Y}$ respectively of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$, where coalitions $\mathcal{X}$ and $\mathcal{Y}$ are stored in the partition table P_t as $\{\{x_1\}\{x_2\}\}$ and $\{\{y_1\}\{y_2\}\}$ or as they are.
Output: maximum valued $\mathcal{CS}$ from the coalitions $\mathcal{X}$ and $\mathcal{Y}$.
Used variables: val and CS_o are used to keep track of the values and the coalition structures obtained so far from $\mathcal{X}$ and $\mathcal{Y}$.

1: $val \leftarrow V_t(\mathcal{X}) + V_t(\mathcal{Y})$
2: $CS_o \leftarrow \{\{\mathcal{X}\}\{\mathcal{Y}\}\}$
3: **for** Each component i in $\mathcal{X} = \{\{x_1\}\{x_2\}\}$ **do**
4: **for** Each component j in $\mathcal{Y} = \{\{y_1\}\{y_2\}\}$ **do**
//When $|i \cup j| \geq \lceil \frac{n}{2} \rceil$, no need to merge because this situation is already considered in the Algorithm 3, lines 15-23.
5: **if** $|i \cup j| < \lceil \frac{n}{2} \rceil$ **then**
6: **if** $V_t(i \cup j) + V_t(\mathcal{X} \setminus i) + V_t(\mathcal{Y} \setminus j) > val$ **then**
7: $val \leftarrow V_t(i \cup j) + V_t(\mathcal{X} \setminus i) + V_t(\mathcal{Y} \setminus j)$ ▷ see footnote a.
8: $CS_o \leftarrow \{\{i \cup j\}\{\mathcal{X} \setminus i\}\{\mathcal{Y} \setminus j\}\}$
9: **end if**
10: **end if**
11: **end for**
12: **end for**
13: return CS_o, val

[a] This line merges each component of $\mathcal{X}$ with another component of $\mathcal{Y}$ one at a time and leaves the other parts unchanged.

3.3 ImDP algorithm

ImDP runs for coalitions of size $1, 2, \ldots, \lceil \frac{n}{2} \rceil$ as shown in lines 1–13 of Algorithm 3. Line 6 shows how $Merge_1$ function is called for each coalition.

Next, ImDP picks all coalitions of size $n, \ldots, \lceil \frac{n}{2} \rceil$ and each time takes the rest of the unassigned agents i.e. complement of the chosen coalition. Lines 15–23 describe how ImDP

Algorithm 3 Imperfect dynamic programming algorithm for CSG problem.

Input: Set of all possible non-empty subsets of n agents ($2^n - 1$). The value of any coalition $\mathcal{C}$ is $v(\mathcal{C})$.
Output: Coalition structure $\mathcal{CS}$ and its value $v(\mathcal{CS})$.
Used variables: Variables $\mathcal{X}$ and $\mathcal{Y}$ are used to store coalitions of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$. P_t the partition table and V_t the optimal value table. To store coalition structures we use the variables $\mathcal{CS}_o$.

// Algorithm runs for coalitions size $1, 2, \ldots \lceil \frac{n}{2} \rceil$. Lines 5-10 show working procedure of $Merge_1$.

1: **for** $i = 1$ to $\lceil \frac{n}{2} \rceil$ **do**
2: **for** each coalition $\mathcal{C} \subseteq \mathcal{A}$, where $|\mathcal{C}| = i$ **do**
3: $V_t(\mathcal{C}) \leftarrow v(\mathcal{C})$
4: $P_t(\mathcal{C}) \leftarrow \mathcal{C}$
5: **for** each agent $a_s \in \mathcal{C}$ **do**
6: $\mathcal{CS}_p, Value \leftarrow Merge_1\ (a_s, \mathcal{C} \setminus a_s)$
7: **if** $V_t(\mathcal{C}) < Value$ **then**
8: $V_t(\mathcal{C}) = Value$
9: $P_t(\mathcal{C}) \leftarrow \mathcal{CS}_p$
10: **end if**
11: **end for**
12: **end for**
13: **end for**
14: Maximum $\leftarrow 0$

// Lines 15-23 show that the algorithm goes through all coalitions of size $n, n-1, \ldots \lceil \frac{n}{2} \rceil$ and each time picks rest of the unassigned agents.

15: **for** $j = n$ downto $\lceil \frac{n}{2} \rceil$ **do**
16: $\mathcal{X} \leftarrow \mathcal{C}^j$ ▷ $\mathcal{C}^j$ denotes a coalition of size j
17: $\mathcal{Y} \leftarrow \mathcal{A} \setminus \mathcal{X}$ ▷ $\mathcal{Y}$ is the complement of $\mathcal{X}$.
18: $\text{Temp}_{value} \leftarrow v(\mathcal{X}) + V_t(\mathcal{Y})$
19: **if** $\text{Temp}_{value} > Maximum$ **then**
20: $Maximum \leftarrow \text{Temp}_{value}$
21: $\mathcal{CS} \leftarrow \{\{\mathcal{X}\}\{\mathcal{Y}\}\}$
22: **end if**
23: **end for**

// Lines 24-30 show working procedure of $Merge_2$.

24: **for** each coalition $\mathcal{Z}$ of size $\lceil \frac{n}{2} \rceil$ **do**
25: $\mathcal{CS}_o, Value \leftarrow Merge_2(\mathcal{Z}, \mathcal{A} \setminus \mathcal{Z})$
26: **if** $value > Maximum$ **then**
27: $Maximum \leftarrow Value$
28: $\mathcal{CS} \leftarrow \mathcal{CS}_0$
29: **end if**
30: **end for**

// Lines 31-36 show how final $\mathcal{CS}$ is generated.

31: **for** each coalition $\mathcal{C} \in \mathcal{CS}$ **do**
32: **if** $P_t(\mathcal{C}) \neq \mathcal{C}$ **then**
33: $\mathcal{CS} \leftarrow (\mathcal{CS} \setminus \mathcal{C} \cup P_t(\mathcal{C}))$
34: **end if**
35: **end for**
36: Return $\mathcal{CS}$ and $v(\mathcal{CS})$.

evaluates and computes the maximum valued coalition structure found so far. Lines 24–30 elaborate on how to check all the feasible coalition structures from two disjoint coalitions $\mathcal{X}$ and $\mathcal{Y}$ of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$ using $Merge_2$ function. In the merging process, $Merge_2$ checks the size of the merged coalitions. If this size is greater than or equal to $\lceil \frac{n}{2} \rceil$, then no need to merge because it has already been computed by ImDP, as shown in lines 15–23 of Algorithm 3. Finally, lines 31–35 find the optimal coalition structures in the bottom-up fashion. Now, let's propose a numerical example for better understanding of the whole procedure.

Example 2 In our example shown in Fig. 1, ImDP evaluates all coalitions of size $\lceil \frac{4}{2} \rceil = 2$ incrementally starting from a coalition of size 1. Now it picks all coalitions of size 4 and 3 and selects their complement coalitions as follows: $v(\{1, 2, 3, 4\}) = 131$, $v(\{1, 2, 3\}) + v_t(\{4\}) = 85 + 41 = 126$, $v(\{1, 2, 4\}) + v_t(\{3\}) = 110 + 20 = 130$, $v(\{1, 3, 4\}) + v_t(\{2\}) = 92 + 35 = 127$ and $v(\{2, 3, 4\}) + v_t(\{1\}) = 108 + 24 = 132$.

So far, maximum valued coalition structure is $\{\{2, 3, 4\}\{1\}\}$ with value 132. Next, all the compatible pairs of the coalitions of size 2 are checked using the $Merge_2$ function as follows: First, ImDP picks $\{1, 2\}$ and $\{3, 4\}$ and checks that it gives value $59 + 75 = 134$. Now, at this stage, $Merge_2$ function finds that one of the coalition, i.e., $\{3, 4\}$, is stored as it is. So, no need to merge, because whenever $Merge_2$ function tries to merge $\{3, 4\}$ with $\{1\}$ or $\{2\}$ it will generate already computed coalition structures $\{1, 3, 4\}\{2\}$ or $\{2, 3, 4\}\{1\}$ (this situation is already considered in the Algorithm 3, lines 15–23). Next, $Merge_2$ function picks two disjoint coalitions $\{1, 3\}$ and $\{2, 4\}$ and finds that they are stored as $\{1\}\{3\}$ and $\{2\}\{4\}$. First, $Merge_2$ function merges $\{1\}$ with $\{2\}$ and $\{4\}$ once at a time, thus creates coalition structures $\{\{1, 2\}\{3\}\{4\}\}$ and $\{\{1, 4\}\{2\}\{3\}\}$. Next, $Merge_2$ merges $\{3\}$ with $\{2\}$ and $\{4\}$ once at a time and creates the coalition structures $\{\{2, 3\}\{1\}\{4\}\}$ and $\{\{3, 4\}\{1\}\{2\}\}$. Figure 3 details these operations. Then, $Merge_2$ picks the compatible pairs $\{1, 4\}$ and $\{2, 3\}$, evaluates their value as $79 + 55 = 134$ and finds that there is no need to perform merge operation because one of the coalitions is stored as it is in the table P_t. $Merge_2$ function calculates values for all these coalition structures as follows:

$$
\begin{aligned}
&V_t(\{1, 2\}) + V_t(\{3, 4\}) = 59 + 75 = 134\\
&V_t(\{1, 3\}) + V_t(\{2, 4\}) = 44 + 76 = 120\\
&V_t(\{1, 2\}) + V_t(\{3\}) + V_t(\{4\}) = 59 + 20 + 41 = 120\\
&V_t(\{1, 4\}) + V_t(\{2\}) + V_t(\{3\}) = 79 + 35 + 20 = 134\\
&V_t(\{2, 3\}) + V_t(\{1\}) + V_t(\{4\}) = 55 + 24 + 41 = 120\\
&V_t(\{3, 4\}) + V_t(\{1\}) + V_t(\{2\}) = 75 + 24 + 35 = 134
\end{aligned}
$$

ImDP finds that the maximum valued $\mathcal{CS}$ is $\{\{3, 4\}\{1\}\{2\}\}$ with value 134.

To better understand the principle of ImDP algorithm, let us consider the example given in Table 1 for five agents. In this example, ImDP evaluates all the coalitions of size 2, then size 3 till the size becomes $\lceil \frac{n}{2} \rceil$. Next, ImDP picks all the coalitions of size 5, 4, 3 and each time takes the complement of the choosen coalition. For example, when ImDP picks the coalition $\{1, 2, 3, 4\}$, the complement coalition would be the coalition $\{5\}$ and the value of the coalition structure is $v\{1, 2, 3, 4\} + v\{5\} = 1 + 3 = 4$. After this step, $Merge_2$ function picks all the coalitions of size $\lceil \frac{n}{2} \rceil = 3$ and each time selects the complement coalition of size 2. For example, when it picks the coalition $\{3, 4, 5\}$, the complement coalition selected by $Merge_2$ is $\{1, 2\}$. Now, at this stage, $Merge_2$ function finds that the coalition, $\{3, 4, 5\}$, is stored as $\{3, 4\}\{5\}$ and the coalition $\{1, 2\}$ is stored as $\{1\}\{2\}$. First, $Merge_2$ function merges $\{3, 4\}$ with $\{1\}$ and $\{2\}$ once at a time, thus creates the coalition structures $\{\{1, 3, 4\}\{2\}\{5\}\}$ and $\{\{2, 3, 4\}\{1\}\{5\}\}$. Next, $Merge_2$ merges $\{5\}$ with $\{1\}$ and $\{2\}$ once at a time and creates the coalition structures $\{\{1, 5\}\{3, 4\}\{2\}\}$ and $\{\{2, 5\}\{3, 4\}\{1\}\}$. The value of the coalition structure $\{\{1, 5\}\{3, 4\}\{2\}\}$ is calculated as $V_t\{1, 5\} + V_t\{3, 4\} + V_t\{2\} = 5 + 4 + 3 = 12$. Figure 3 details these operations. In this way, $Merge_2$ function calculates the value of the optimal coalition structure.

ImDP algorithm finds the optimal coalition structure when one of the optimal partitions is actually considered by $Merge_1$ and $Merge_2$ functions. If none of these partitions is considered by ImDP, then it fails to give the optimal result. However, ImDP will still give a

Table 1 Working principle of ImDP algorithm computing the tables P_t and V_t, given five agents $\mathcal{A} = \{1, 2, 3, 4, 5\}$, and a characteristic function v

Size	Coalition ($\mathcal{C}$)	$v(\mathcal{C})$	Optimal partition P_t	Optimal value V_t
	{1}	2	{1}	2
	{2}	3	{2}	3
1	{3}	2	{3}	2
	{4}	2	{4}	2
	{5}	3	{5}	3
	{1, 2}	1	{1}{2}	5
	{1, 3}	1	{1}{3}	4
	{1, 4}	3	{1}{4}	4
	{1, 5}	1	{1}{5}	5
	{2, 3}	2	{2}{3}	5
2	{2, 4}	1	{2}{4}	5
	{2, 5}	3	{2}{5}	6
	{3, 4}	2	{3}{4}	4
	{3, 5}	1	{3}{5}	5
	{4, 5}	1	{4}{5}	5
	{1, 2, 3}	3	{1, 2}{3}	7
	{1, 2, 4}	1	{1, 2}{4}	7
	{1, 2, 5}	1	{1, 2}{5}	8
	{1, 3, 4}	1	{1, 3}{4}	6
3	{1, 3, 5}	2	{1, 3}{5}	7
	{1, 4, 5}	3	{1, 4}{5}	7
	{2, 3, 4}	1	{2, 3}{4}	7
	{2, 3, 5}	2	{2, 3}{5}	8
	{2, 4, 5}	1	{2, 4}{5}	8
	{3, 4, 5}	1	{3, 4}{5}	7
	{1, 2, 3, 4}	1	{1, 2, 3, 4}	1
	{1, 2, 3, 5}	2	{1, 2, 3, 5}	2
4	{1, 2, 4, 5}	1	{1, 2, 4, 5}	1
	{1, 3, 4, 5}	2	{1, 3, 4, 5}	2
	{2, 3, 4, 5}	1	{2, 3, 4, 5}	1
5	{1, 2, 3, 4, 5}	1	{1, 2, 3, 4, 5}	12

sub-optimal solution. To better understand the fail cause of ImDP algorithm, we need to introduce the coalition structure graph (cf. Fig. 4) Sandholm et al. (1999), which consists of a number of nodes and a number of edges connecting these nodes. Each node represents a coalition structure, and each edge represents how the DP algorithm moves in the coalition structure graph. Given n agents, the nodes in this graph are categorized into n levels, where each level $L_i : i \in \{1, 2, \ldots, n\}$ contains nodes representing coalition structures containing i coalitions. Each edge in this graph represents merging of two coalitions into one coalition (when followed downwards) and the splitting of one coalition into two coalitions (when followed upwards).

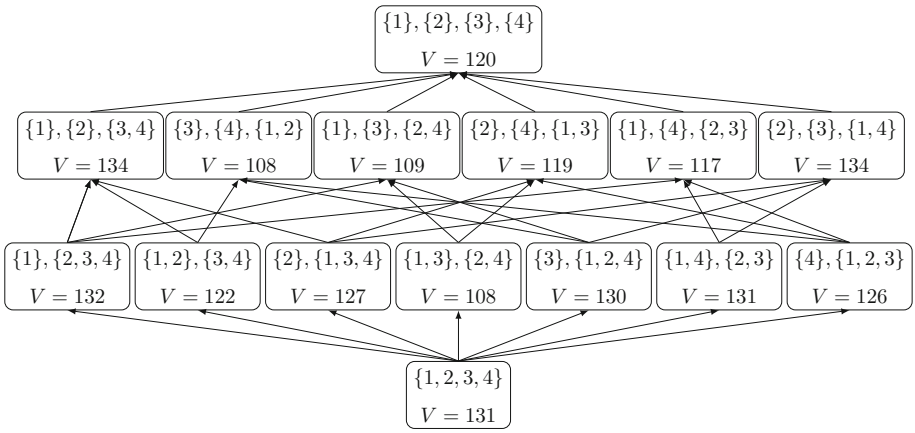


Fig. 4 The coalition structure graph of 4 agents

Let's fix a node P in the coalition structure graph, which contains the optimal coalition structure. ImDP algorithm finds the optimal coalition structure in the coalition structure graph if the movement of ImDP in this graph reaches the node P, starting from the bottom node where all agents are in one set. Otherwise ImDP fails to produce the optimal result.

4 Computational efficiency of ImDP

First, we have to check ImDP's worst case time complexity for n number of agents. ImDP evaluates all coalitions of size $1, \ldots \lceil \frac{n}{2} \rceil$, i.e. it performs the following steps.

$$\binom{n}{1} * 1 + \binom{n}{2} * 2, + \cdots, + \binom{n}{\lceil n/2 \rceil} * (\lceil n/2 \rceil) = \sum_{k=1}^{(\lceil \frac{n}{2} \rceil)} \binom{n}{k} * k$$

Using the identity $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ (assume n is even), we get

$$\sum_{k=1}^{\frac{n}{2}} \binom{n}{k} * k = \sum_{k=1}^{\frac{n}{2}} k \frac{n}{k} \binom{n-1}{k-1} = n \sum_{j=0}^{\frac{n}{2}-1} \binom{n-1}{j}.$$

Now we are going to compute the bound for $j = 0, 1, \ldots, n/2$. Let $f(m, j) = \sum_{j=0}^{\frac{n}{2}-1} \binom{n-1}{j}$, we know

$$\frac{\binom{m}{j} + \binom{m}{j-1} + \binom{m}{j-2} \cdots}{\binom{m}{j}} = 1 + \frac{j}{m-j+1} + \frac{j(j-1)}{(m-j+1)(m-j+2)} + \cdots$$

Now we can bound the right side from above by the geometric series $1 + \frac{j}{m-j+1} + \left(\frac{j}{m-j+1}\right)^2 + \cdots$, which is the sum of a geometric series $\frac{1-r^{j+1}}{1-r}$, where $r = \frac{j}{m-j+1}$. Here $r < 1$, then

we obtain a simpler expression

$$f(m, j) \leq \binom{n}{j} \frac{m-(j-1)}{m-(2j-1)}$$

Therefore we get

$$f(m, n/2) \leq \binom{n}{n/2} \equiv O(2^n)$$

$$\sum_{k=1}^{(\lceil \frac{n}{2} \rceil)} \binom{n}{k} * k \equiv O(n2^n)$$

There remain $\frac{2^n}{2}$ coalitions that ImDP needs to check which lead to a total of $O(2^n)$ steps necessary for this stage. Finally, in the last step, ImDP checks all the coalitions in the middle level using $Merge_2$ function. The central term in the binomial series is the largest one, and it is proved that $\binom{n}{n/2}$ is $O(2^n)$ because it follows from Wallis product. $Merge_2$ function requires 4 merge operations in the worst case when applied to two disjoint coalitions of size $\lceil \frac{n}{2} \rceil$ and $n - \lceil \frac{n}{2} \rceil$. Hence, total 4×2^n operations are performed by $Merge_2$, which is an order of 2^n. The input to our algorithm is $K = 2^n$, where, n is the number of agents in $\mathcal{A}$.

So, the total time complexity is $O(n2^n) + O(2^n) + O(2^n) = O(n2^n)$. Putting the value of $n = \log K$, we get the time complexity of ImDP is $O(K \log K)$.

5 Experimental evaluation

Having described all the parts of our algorithm, we now seek to evaluate its effectiveness by comparing it against the ODP–IP. 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 under Linux operating system (64 bit) with 160 GB of RAM. For ODP–IP, we used the code provided by the authors of ODP–IP (Michalak et al. 2016).

5.1 Dataset generation

One way to validate or compare imperfect algorithm for a NP- hard combinatorial optimization problem is to run them on typical problem instances and see how often they fail. Any imperfect algorithm is convenient if the algorithm does not fail too often. With this in mind, we compare the proposed algorithm with ODP–IP using different value distributions. Specifically, we considered the following distributions, and made tests for sets of 5–27 agents as shown in Fig. 5. For these sets, we tested 50 instances except for sets of 27 agents for which we tested 25 instances because for 27 agents the input file size becomes very large. That means, for a single distribution we tested over 1125 problem instances.

I. *Agent-based Uniform (ABU)* Each agent a_i is assigned a random power $p_i \sim U(0, 10)$, reflecting its average performance over all coalitions (Rahwan et al. 2012). Then for all coalitions, $\mathcal{C}$ in which agent a_i appears, the actual power of a_i in $\mathcal{C}$ is determined as $p_i^{\mathcal{C}} \sim U(0, 2 \times p_i)$ and the coalition value is calculated as the sum of all the members' power in that coalition. That is, $\forall \mathcal{C},\ v(\mathcal{C}) = \sum_{a_i \in \mathcal{C}} p_i^{\mathcal{C}}$.

II. *Agent-based Normal (ABN)* Each agent a_i is assigned a random power $p_i \sim N(10, 0.01)$ (Michalak et al. 2016). Then for all coalitions, $\mathcal{C}$ in which agent a_i

appears, the actual power of a_i in $\mathcal{C}$ is determined as $p_i^C \sim N(p_i, 0.01)$ and the coalition value is calculated as the sum of all the members' power in that coalition. That is, $\forall \mathcal{C},\ v(\mathcal{C}) = \sum_{a_i \in \mathcal{C}} p_i^C$.

III. *Chi-square* (χ^2) 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.

IV. *Beta* (β) The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times$ Beta (α, β), where $\alpha = \beta = 0.5$.

V. *Exponential (EXPO)* The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times \text{Exp}\ (\lambda)$, where $\lambda = 1$ and n is the number of agents for all the coalitions $C \in 2^A - 1$.

VI. *Geometric (GEO)* For each coalition $\mathcal{C}$, a value is generated as $rv = U(|\mathcal{C}|/n, 1)$, where n is the number of agents. The value of a coalition $v(\mathcal{C}) = Geometric(rv) * rv * |\mathcal{C}|$, next a random number r is generated $r \sim U(0, 50)$ and is added to the coalition value $v(\mathcal{C})$ with probability 0.2.

VII. *Weibull Distribution (WB)* The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |C| \times$ Weibull $(|\mathcal{C}|)$.

VIII. *Gamma* The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times$ Gamma (x, θ), where $x = \theta = 2$.

IX. *Modified Normal (MN-D)* The value of each coalition $\mathcal{C}$ is first drawn as $v(\mathcal{C}) \sim N(a, b)$, where $a = 10 \times |\mathcal{C}|$ and $b = 0.01$ (Rahwan et al. 2012), next a random number r is generated $r \sim U(0, 50)$ and is added to the coalition value $v(\mathcal{C})$ with probability 0.2.

X. *Modified Uniform (MU-D)* The value of each coalition $\mathcal{C}$ is drawn uniformly as $v(\mathcal{C}) \sim U(a, b)$, where $a = 0$ and $b = 10 \times |\mathcal{C}|$ (Adams et al. 2010), next a random number r is generated $r \sim U(0, 50)$ and is added to the coalition value $v(\mathcal{C})$ with probability 0.2.

XI. *Normally Distributed Coalition Structures (NDCS)* the value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim N(\mu, \sigma^2)$ (Rahwan et al. 2009), where $\mu = |\mathcal{C}|$ and $\sigma = \sqrt{|\mathcal{C}|}$.

XII. *Normal (N-D)* Every coalition value is drawn from $v(\mathcal{C}) \sim N(\mu, \sigma^2)$ (Rahwan et al. 2007), where $\mu = 10 \times |\mathcal{C}|$ and $\sigma = 0.1$.

XIII. *Rayleigh (RAL)* The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim$ Rayleigh $(Modevalue)$, where $Modevalue$ is defined as $10 * \sqrt{\frac{2}{\pi}} * |C|$.

XIV. *Uniform (U-D)* For all coalitions $\mathcal{C} \in 2^A - 1$, $v(\mathcal{C}) \sim U(a, b)$, where $a = 0$ and $b = |\mathcal{C}|$ (Larson and Sandholm 2000).

XV. *F Distribution (F-D)* Each coalition value is calculated using F-distribution with degrees of freedom in the numerator $Dfnum = 1$ and degrees of freedom in the denominator $Dfden_1 = |\mathcal{C}| + 1$.

XVII. *Laplace or double exponential (LAP)* The value of each coalition $\mathcal{C}$ is drawn as $v(C) \sim$ Laplace (μ, λ) where $\mu = 10 \times |\mathcal{C}|$ and $\lambda = 0.1$, the exponential decay.

5.2 Performance evaluation

We empirically evaluated the ImDP algorithm and benchmarked it against ODP–IP. We compared the performances of both algorithms given different numbers of agents (5–27). As can be seen, ImDP is faster for some distributions shown in Figs. 5 and 6. These tests revealed that for Agent-based uniform and Agent-based normal datasets, we can use ImDP with an average runtime gain of 90.91% and 89.50% for 27 agents. Further analysis showed the behavior of ODP–IP and ImDP as follows:

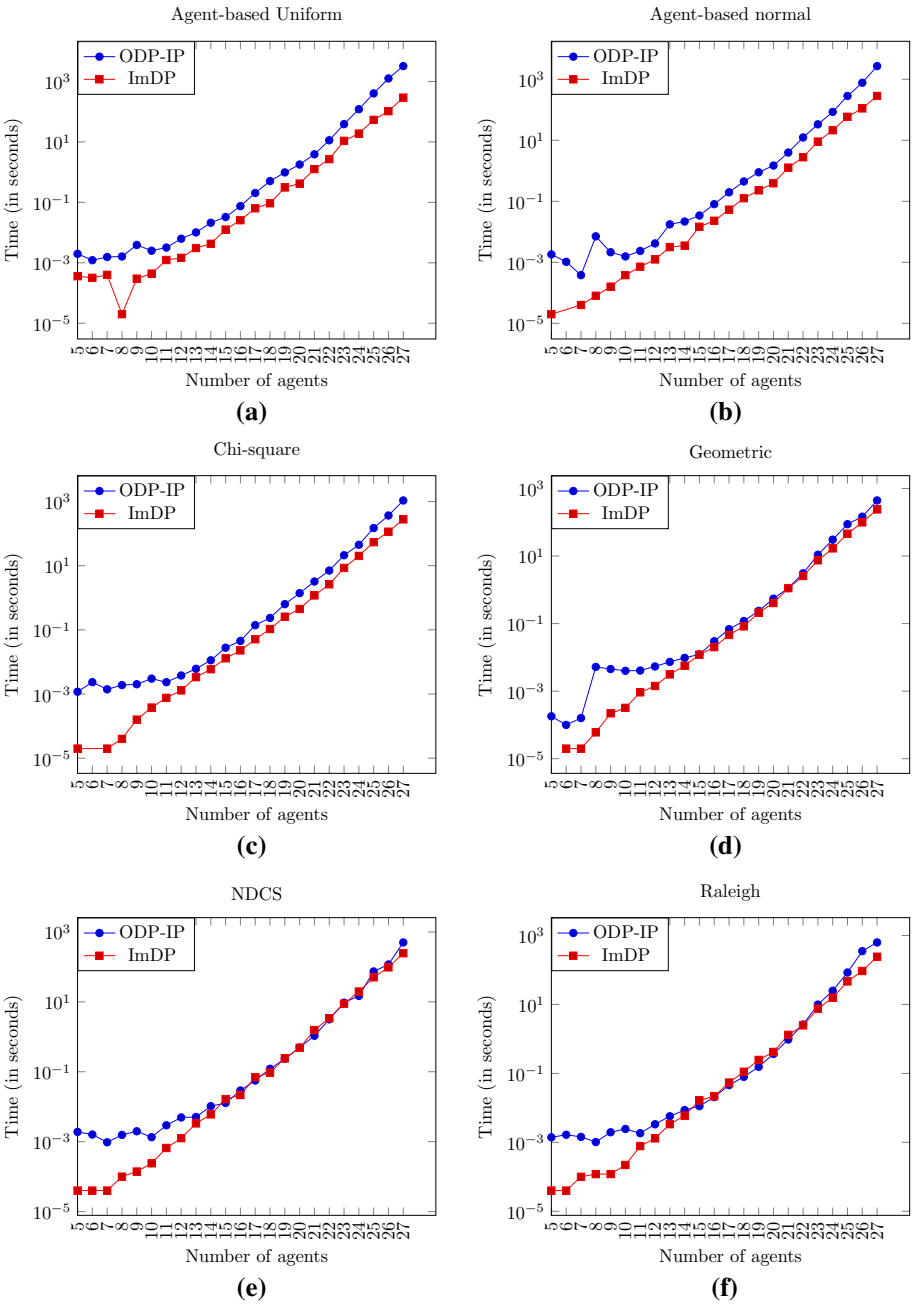


Fig. 5 Time performance of ODP–IP versus ImDP. Here, time is measured in seconds and plotted *on a log scale*

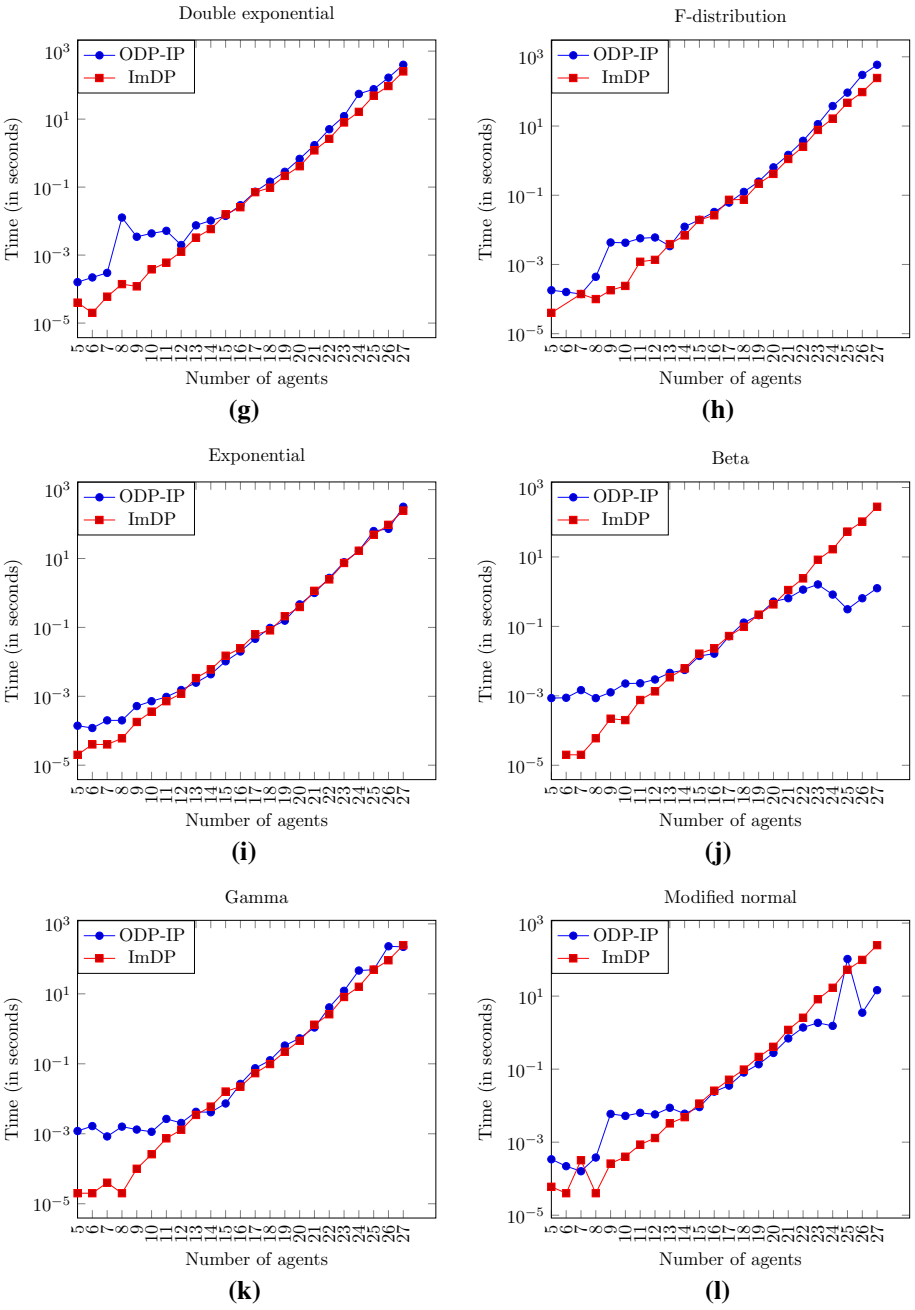


Fig. 5 continued

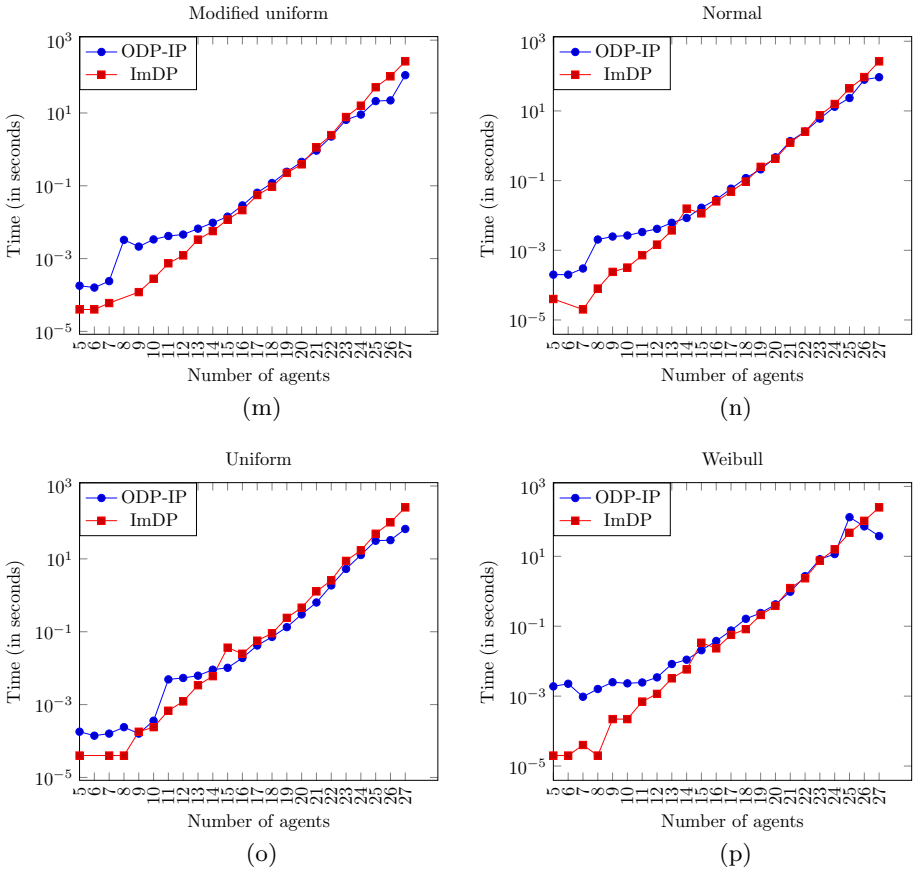


Fig. 5 continued

For instance, for Chi-square dataset, ImDP's average runtime gain over ODP–IP is 74.31% for 27 agents. In geometric distribution, average runtime gain of ImDP is 46.03% over ODP–IP for 27 agents. A similar pattern is found for NDCS, raleigh, double exponential exponential, and F- distribution. In these cases, ImDP is better than ODP–IP for 27 agents with an average runtime gain of 51%, 61.46%, 36.36%, 22.64% and 58.50% respectively. In other cases ODP–IP outperforms ImDP for 27 agents. For beta, gamma, modified-normal, modified uniform, normal, uniform, and weibull distributions ImDP is slower than ODP–IP by 222%, 12.32%, 1670%, 138%, 182%, 290%, and 555%. The runtime gain is calculated as $\frac{IMDP_{time} - ODP-IP_{time}}{ODP-IP_{time}} \times 100$.

Moreover, experimental results show that we gain maximum 2931 s in Agent-based uniform distribution. Table 2 shows the absolute runtime values for 27 agents.

Note that the number of operations performed by ImDP is not related to the characteristic function at hand, i.e., it depends solely on the number of agents. On the contrary, the number of operations performed by ODP–IP depends on the effectiveness of IP's branch-and-bound technique, which depends on the characteristic function used at hand (Michalak et al. 2016). This is the reason why there is such a significant difference in the run-time of ImDP with respect to ODP–IP when varying the distributions that define the characteristic function as

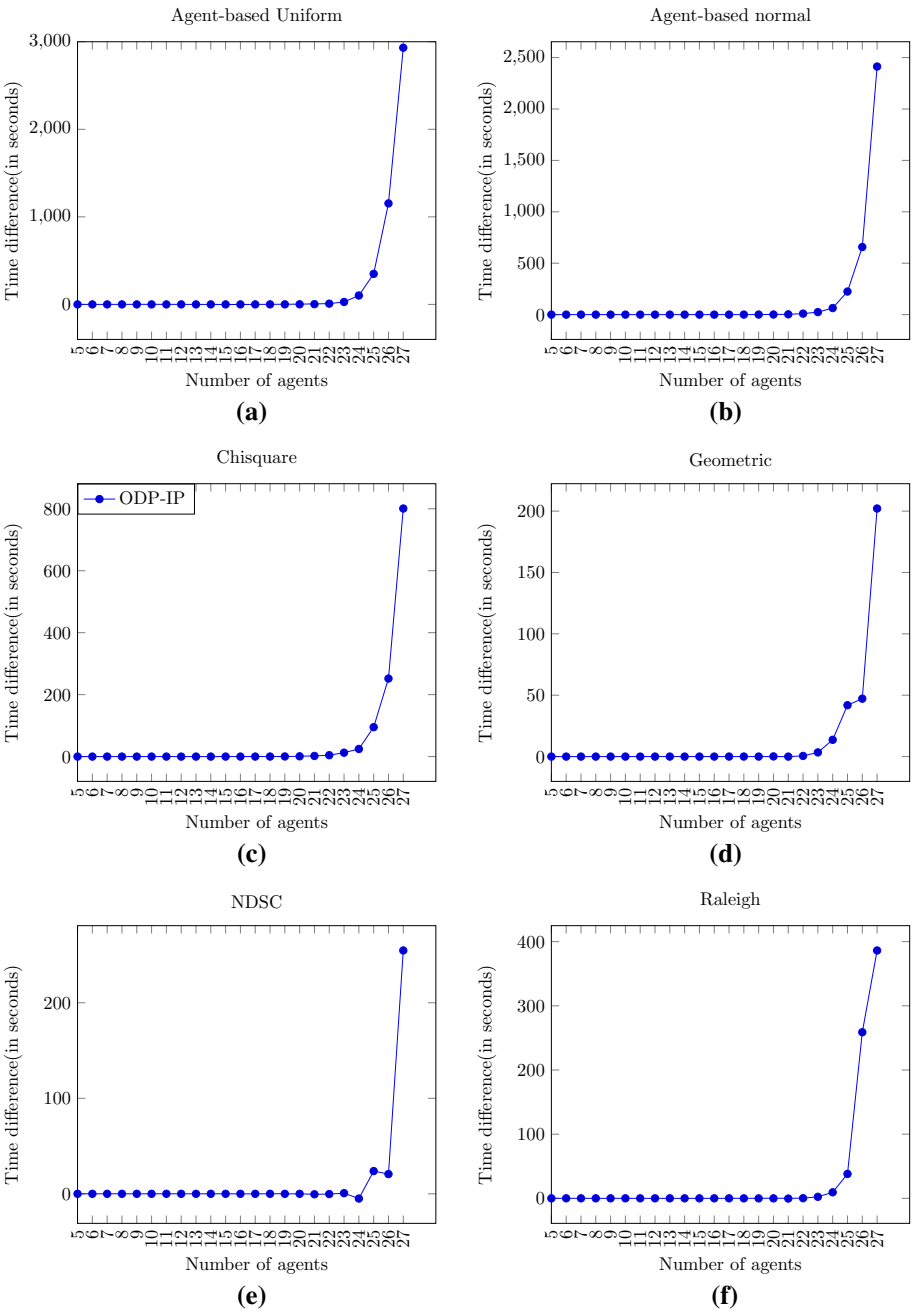


Fig. 6 Time difference (in s) between ODP–IP versus ImDP for different numbers of agents

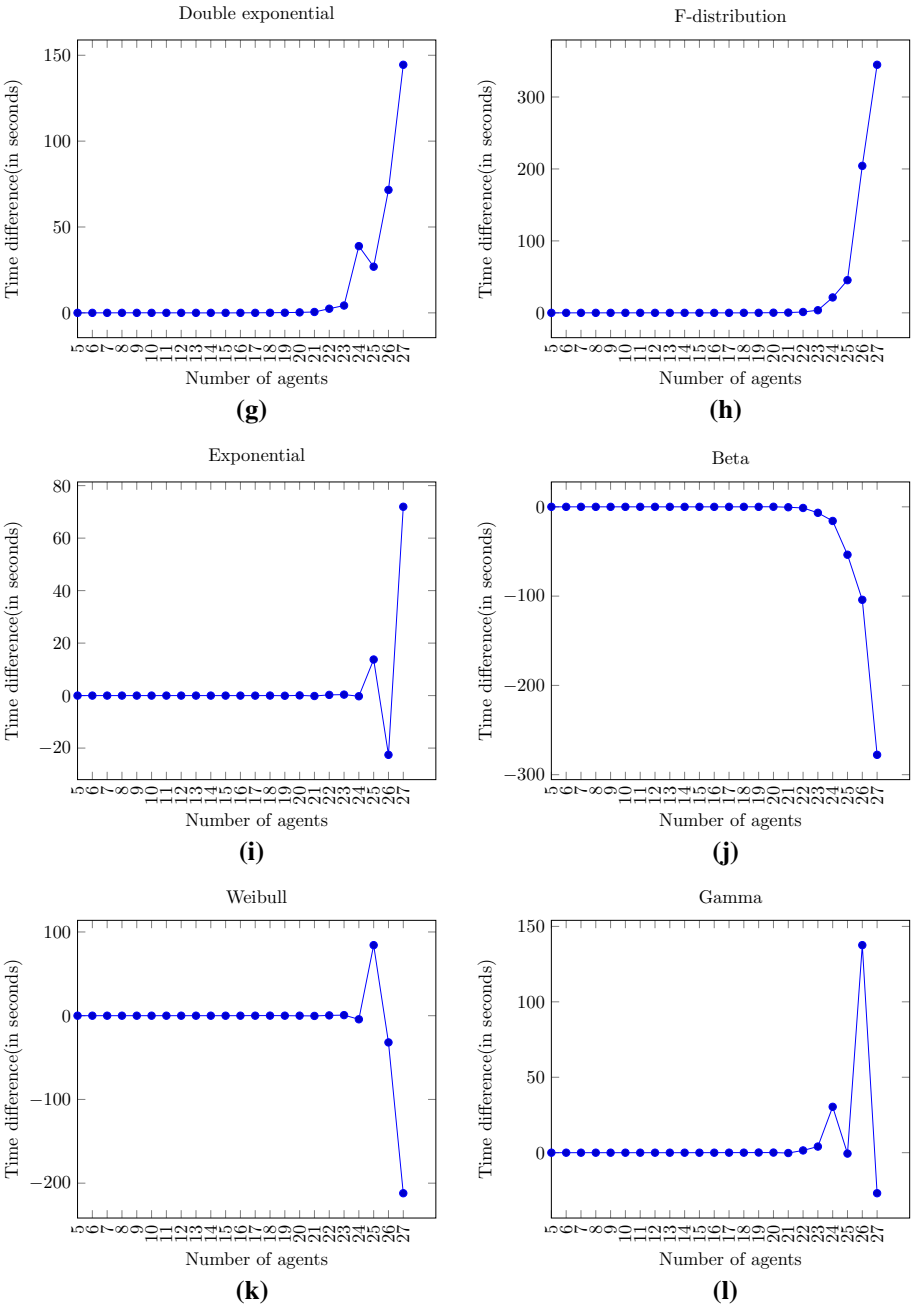


Fig. 6 continued

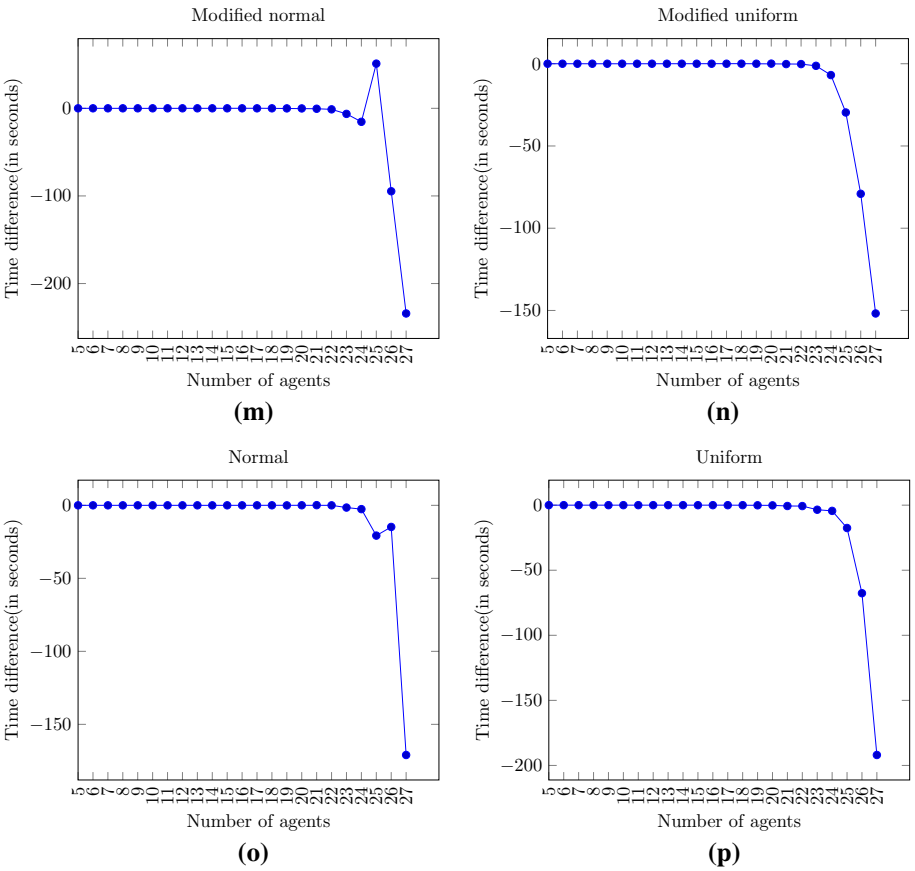


Fig. 6 continued

shown in Fig. 5. For instance, ODP–IP can generate an optimal result in a very short time in the case of β distribution, but for the agent-based uniform distribution ODP–IP is unable to generate the optimal solution quickly, because ODP–IP can prune many subspaces in the case of β distribution but it cannot prune subspaces easily in the case of the agent-based uniform distribution.

Broadly speaking, we have found some datasets for which ImDP works well for some problem instances and ODP–IP works better for others. Table 3 shows the number of problem instances for 5 to 27 agents, where ImDP time is better than or equals to ODP–IP time (cf. column 2 in Table 3).

As ImDP is an imperfect algorithm, we need to know if ImDP does not produce optimal $\mathcal{CS}$, then what is the difference between ODP–IP generated optimal $\mathcal{CS}$ and ImDP generated $\mathcal{CS}$. ImDP algorithm fails only for few datasets but the failure rate is very low. Table 4 shows the number of fail cases where ImDP is unable to produce the exact result for each distribution.

We also found that, in the case of failure, (ImDP generated $\mathcal{CS}$ value)/(Optimal $\mathcal{CS}$ value) is always greater than .90 and most ratios are .99. In Fig. 7b we have drawn the results only for 8 distributions in order to show how ImDP can surpass significantly ODP–IP on larger sets

Table 2 Time difference between ODP–IP and ImDP in seconds for 27 agents

Distribution	Time measured in seconds		
	ODP–IP Time (t_1)	ImDP Time (t_2)	Difference $t_1 - t_2$
Agent-based uniform	3224.8075	293.0295	2931.778
Agent-based normal	2696.692	283.88	2412.812
Chi-square	1078.2885	277.5145	800.774
Raleigh	628.897	242.6665	386.2305
F-distribution	588.918	244.1565	344.7615
NDCS	500.1605	245.4155	254.745
Geometric	441.038	238.922	202.116
Exponential	318.16	246.21	71.95
Beta	1.25	279.01	− 277.76
Gamma	219.47	246.34	− 26.87
Modified-normal	14.55	248.70	− 234.15
Modified-uniform	109.14	260.97	− 151.83
Normal	92.67	263.66	− 170.99
Uniform	66.19	258.15	− 191.96
Weibull	37.98	249.98	− 212

Table 3 The number of problem instances where ImDP is faster than ODP–IP. For a single distribution we tested over 1125 problem instances ranges from 5–27 agents

Distribution	*Probleminstance*	*Probleminstance*
	ImDP ≤ ODP–IP	ImDP > ODP–IP
Agent-based Uniform	1084	41
Agent-based Normal	1119	6
Chi-square	1084	41
Beta	754	371
Exponential	738	387
Geometric	934	191
Weibull	931	194
Gamma	673	452
Modified Normal	464	661
Modified Uniform	695	430
NDCS	658	467
Normal	695	430
Rayleigh	623	502
Uniform	448	677
F	699	426
Laplace	782	343

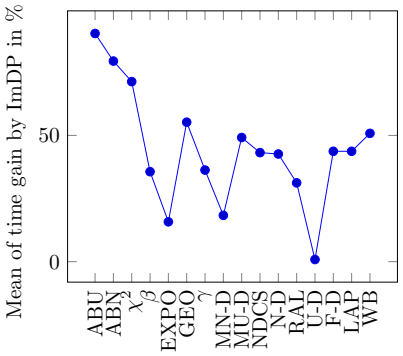
Table 4 The number of failure against 1125 problem instances for each distribution

Distribution	Number of failure	Failure rate (%)
Agent-based uniform	58	5.11
Agent-based normal	57	5.06
Chi-square	83	7.38
Beta	83	12.08
Exponential	45	4.00
Geometric	19	1.70
Weibull	13	1.15
Gamma	51	4.53
Modified normal	14	1.24
Modified uniform	16	1.42
NDCS	86	7.64
Normal	19	1.69
Rayleigh	67	5.95
Uniform	260	23.11
F	24	2.13
Laplace	37	3.30

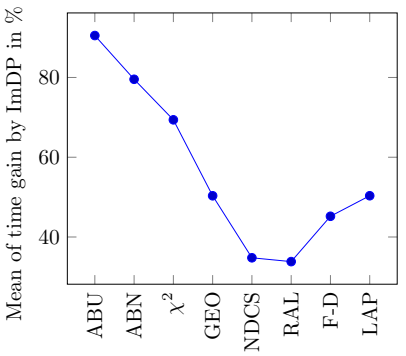
and Fig. 7a shows the behavior of ImDP under 22 agents. As ImDP is an imperfect algorithm, we have shown in Fig. 7c the percentage of the optimal $\mathcal{CS}$ value difference generated by ODP–IP with the $\mathcal{CS}$ value generated by ImDP.

ImDP is not an exact algorithm, but it can generate a near optimal $\mathcal{CS}$ in less time than ODP–IP for these distributions. When tested, we found that for Agent-based uniform distribution ImDP is 49 min faster than ODP–IP for 27 agents as it is depicted in Figs. 5 and 6. Considering this set of 27 agents, the difference in term of values has also been measured as can be seen in Fig. 8. The effectiveness of IP's branch and bound techniques play an essential role in ODP–IP, which in turn depends on the characteristic functions at hand. For some distributions, ODP–IP is much faster because of IP's effectiveness on those distributions. On the other hand, the number of operations performed by ImDP is not influenced by the characteristic functions at hand, i.e. it depends solely on the number of agents. For instance, with Beta, Exponential, Gamma, Modified normal-distribution, Modified uniform-distribution, Normal-distribution and Uniform-distribution, we found that ODP–IP is faster in those cases as compared to ImDP and very slow for other distributions. Now, the question is, why is ODP–IP faster in some cases and slower in other cases? The reason is that applying branch-and-bound in some distributions is very tough and for other distributions most of the search spaces are pruned away easily.

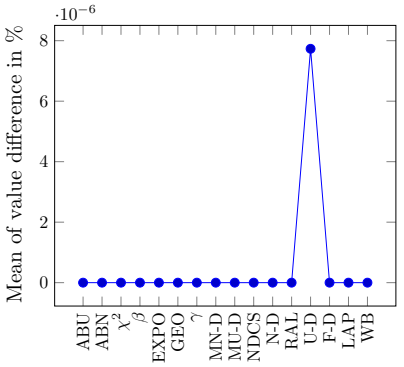
As shown in Fig. 8, the curves have a zigzag shape, but this does not really impact the results, since the differences in values of the two algorithms are very small. They are of the order of 0.2 on average which is negligible compared to the value of the optimal coalition structure.



(a) Time difference for 5-21 agents with 16 different distributions.



(b) Time difference for 5-27 agents with 8 different distributions.



(c) Mean of the differences of solutions values for 5-27 agents

Fig. 7 Time performance and value difference of ODP–IP versus ImDP. **a** Shows the mean of time gain for 5–21 numbers of agents and **b** shows the mean of time gain for 5–27 numbers of agents. In this case ImDP outperforms ODP–IP for all the 8 distributions. **c** shows the mean difference of coalition structure value generated by ODP–IP and ImDP. Here, time is measured in seconds. Percentage of time gain is calculated taking mean of time percentage gain. Similarly, value difference between ODP–IP generated optimal CS and ImDP generated CS are calculated

6 Conclusion and future works

This paper has proposed a new imperfect dynamic programming algorithm (ImDP) for solving the CSG problem. The design of our algorithm is based on two merge functions: $Merge_1$ and $Merge_2$ that we have introduced and illustrated. The experimental results demonstrate that ImDP algorithm fails in rare cases. We have shown that ImDP strongly surpasses ODP–IP in several distributions. Even if the runtimes obtained are still high, the interest of such results is to show that it is still possible to make significant improvements to existing algorithms like ODP–IP. With this first step, it will undoubtedly be possible for future work to improve the results of ImDP as we did for ODP–IP in order to make the run times of the next algorithms shorter. As such, the result already obtained for ImDP will be very beneficial for future

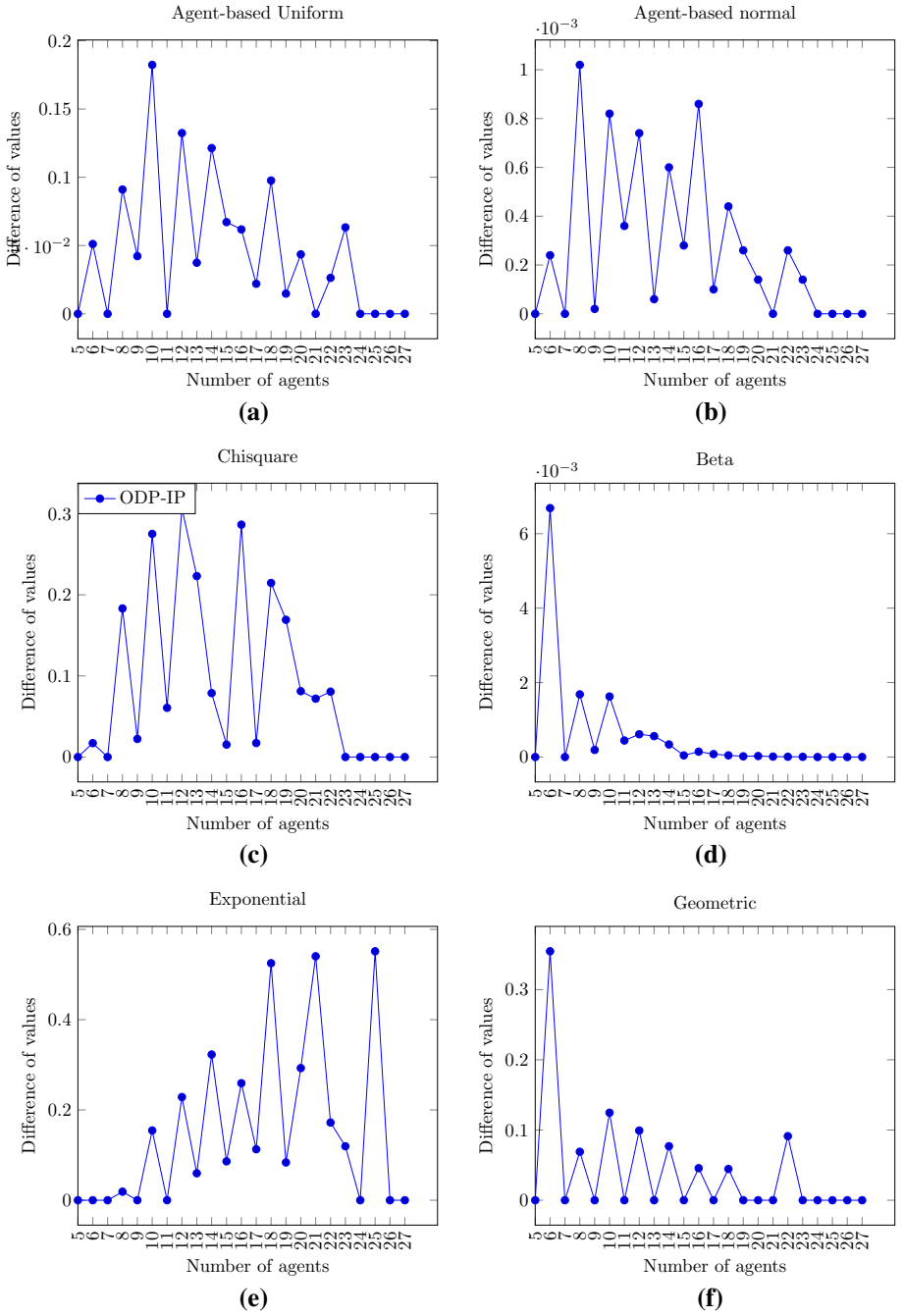


Fig. 8 The mean of the differences of solutions values (absolute value) produced by ODP–IP and ImDP

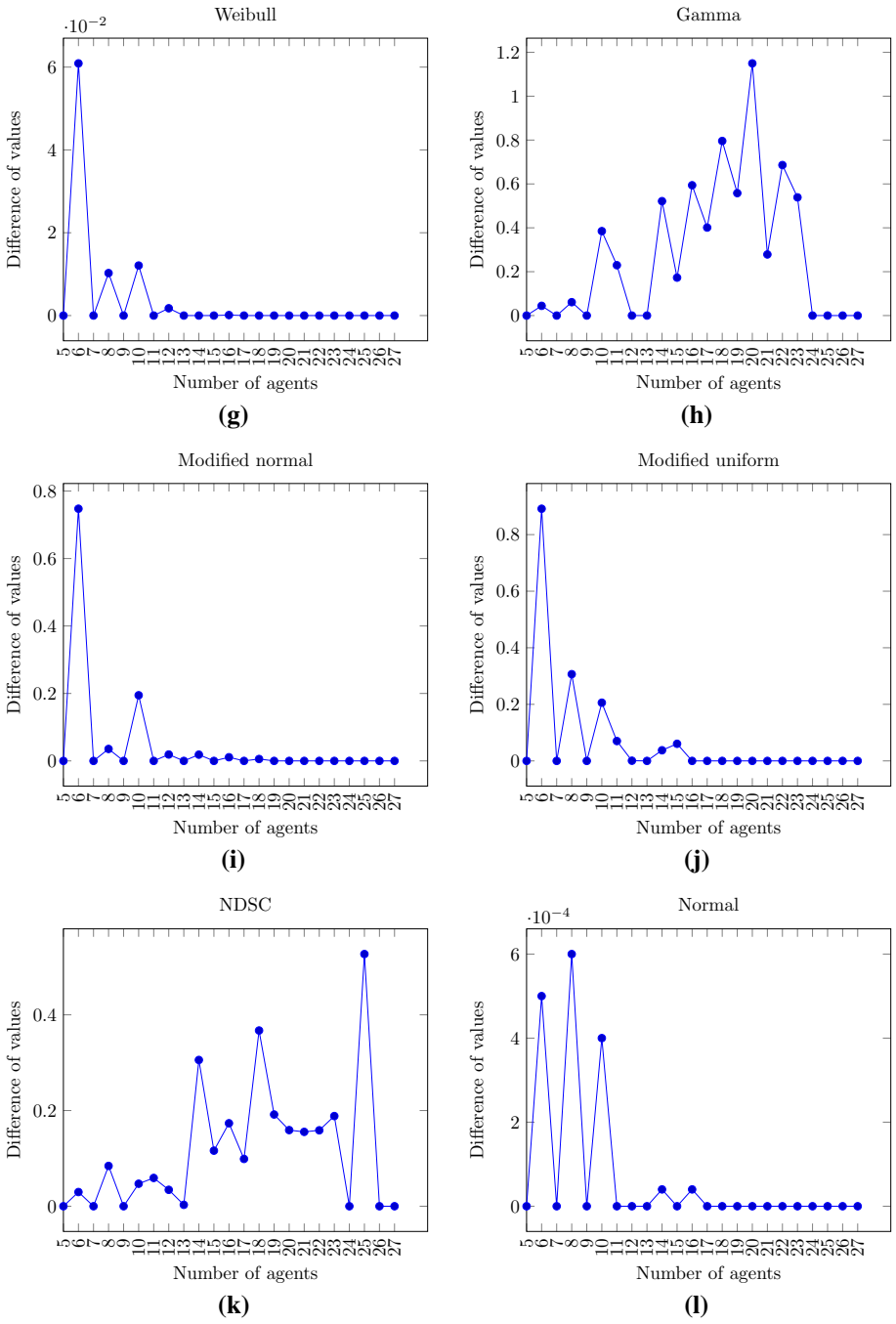


Fig. 8 continued

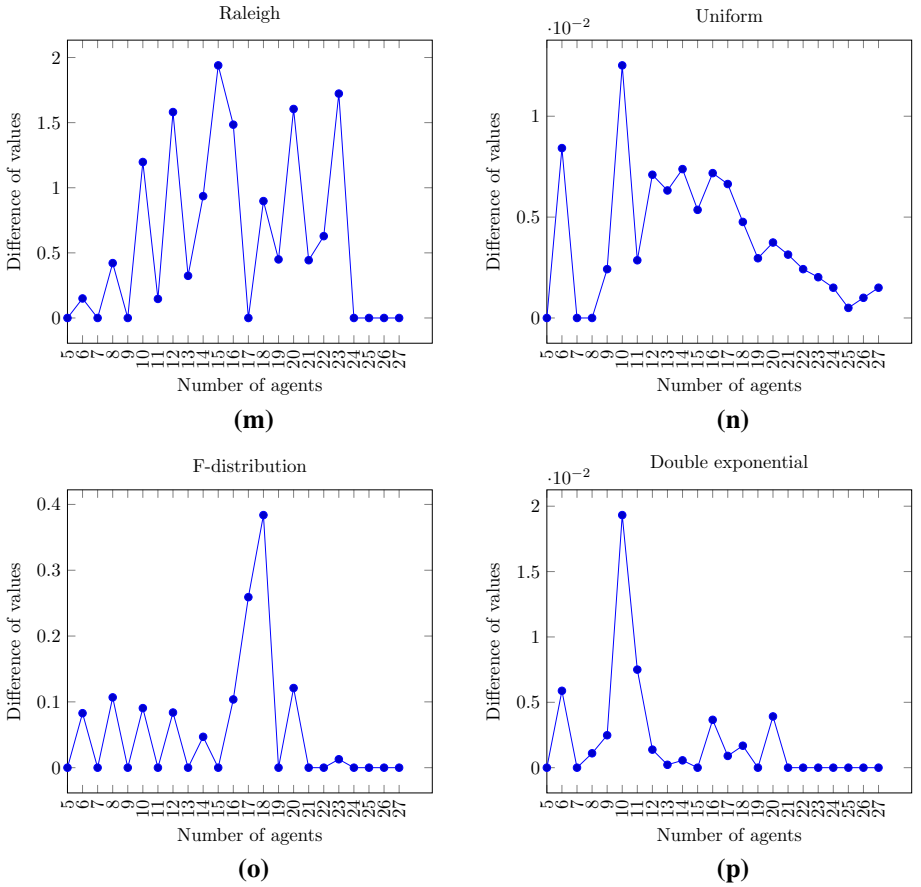


Fig. 8 continued

work since the improvement already obtained on ODP–IP is of the order of 91% for some distributions (cf. Fig. 7a, b).

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

Adams J et al (2010) Approximate coalition structure generation. In: Proceedings of the 24th AAAI conference on artificial intelligence, AAAI, pp 854–859

Bistaffa F, Farinelli A, Cerquides J, Rodríguez-Aguilar J, Ramchurn SD (2014) Anytime coalition structure generation on synergy graphs. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, pp 13–20

Bistaffa F, Farinelli A, Cerquides J, Rodríguez-Aguilar J, Ramchurn SD (2017) Algorithms for graph-constrained coalition formation in the real world. ACM Trans Intell Syst Technol (TIST) 8(4):60

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. Concur Comput Pract Exp 29(5):3969
Dang VD, Jennings NR (2004) Generating coalition structures with finite bound from the optimal guarantees. In: Proceedings of the third international joint conference on autonomous agents and multiagent systems, vol 2. IEEE Computer Society, pp 564–571
Dang VD, Dash RK, Rogers A, Jennings NR (2006) Overlapping coalition formation for efficient data fusion in multi-sensor networks. AAAI 6:635–640
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
Karp RM (1983) The probabilistic analysis of combinatorial optimization algorithms. In: Proceedings of the 10th international symposium on mathematical programming, pp 1601–1609
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
Larson KS, Sandholm TW (2000) Anytime coalition structure generation: an average case study. J Exp Theor Artif Intell 12(1):23–42
Michalak T, Rahwan T, Elkind E, Wooldridge M, Jennings NR (2016) A hybrid exact algorithm for complete set partitioning. Artif Intell 230:14–50
Myerson RB (1977) Graphs and cooperation in games. Math Oper Res 2(3):225–229
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
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, vol 3, AAMAS, pp 1417–1420
Rahwan T, Jennings NR (2008) Coalition structure generation: Dynamic programming meets anytime optimization. In: Proceedings of the 23rd AAAI conference on artificial intelligence, vol 8, AAAI, pp 156–161
Rahwan T, Ramchurn SD, Dang VD, Giovannucci A, Jennings NR (2007) Anytime optimal coalition structure generation. AAAI 7:1184–1190
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, Jennings NR (2012) A hybrid algorithm for coalition structure generation. In: Proceedings of the 26th AAAI conference on artificial intelligence, AAAI, pp 1443–1449
Rahwan T, Michalak TP, Wooldridge M, Jennings NR (2015) Coalition structure generation: a survey. Artif Intell 229:139–174
Rothkopf MH, Pekeč A, Harstad RM (1998) Computationally manageable combinational auctions. Manag Sci 44(8):1131–1147
Sandhlom TW, Lesser VR (1997) Coalitions among computationally bounded agents. Artif Intell 94(1):99–137
Sandholm T, Larson K, Andersson M, Shehory O, Tohmé F (1999) Coalition structure generation with worst case guarantees. Artif Intell 111(1):209–238
Sen S, Dutta PS (2000) Searching for optimal coalition structures. In: Proceedings of the sixth international conference on multi-agent systems, ICMAS, pp 286–292
Service TC, Adams JA (2010) Anytime dynamic programming for coalition structure generation. In: Proceedings of the 9th international conference on autonomous agents and multiagent systems, vol 1, AAMAS, pp 1411–1412
Service TC, Adams JA (2011) Constant factor approximation algorithms for coalition structure generation. Auton Agent Multi-Agent Syst 23(1):1–17
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: IJCAI (1), pp 655–661
Shehory O, Kraus S (1998) Methods for task allocation via agent coalition formation. Artif Intell 101(1–2):165–200
Vinyals M, Voice T, Ramchurn S, Jennings NR (2013) A hierarchical dynamic programming algorithm for optimal coalition structure generation. arXiv preprint arXiv:1310.6704
Voice T, Ramchurn SD, Jennings NR (2012) On coalition formation with sparse synergies. In: Proceedings of the 11th international conference on autonomous agents and multiagent systems, vol 1. International Foundation for Autonomous Agents and Multiagent Systems, pp 223–230

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