ODSS: Efficient Hybridization for Optimal Coalition Structure Generation

Narayan Changder,[1] **Samir Aknine,**[2] **Sarvapali Ramchurn,**[3] **Animesh Dutta**[1]
[1]National Institute of Technology Durgapur, India. [2]LIRIS, Lyon 1 University, France. [3]University of Southampton,UK.
narayan.changder@gmail.com, samir.aknine@univ-lyon1.fr, sdr1@soton.ac.uk, animesh@cse.nitdgp.ac.in

Abstract

Coalition Structure Generation (CSG) is an NP-complete problem that remains difficult to solve on account of its complexity. In this paper, we propose an efficient hybrid algorithm for optimal coalition structure generation called ODSS. ODSS is a hybrid version of two previously established algorithms IDP (Rahwan and Jennings 2008) and IP (Rahwan et al. 2009). ODSS minimizes the overlapping between IDP and IP by dividing the whole search space of CSG into two disjoint sets of subspaces and proposes a novel subspace shrinking technique to reduce the size of the subspace searched by IP with the help of IDP. When compared to the state-of-the-art against a wide variety of value distributions, ODSS is shown to perform better by up to 54.15% on benchmark inputs.

1 Introduction

Coalition formation involves the coming together of collectives of agents to achieve both their individual and common goals. It is a key concept in multi-agent systems. Coalition formation can be applied to many real-world problems such as task allocation (Shehory and Kraus 1998), and to logistics applications (Sandhlom and Lesser 1997).

Various algorithms have been proposed to solve the CSG problem. (Michalak et al. 2016) proposed a hybrid version of IDP (Rahwan and Jennings 2008) 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, Husfeldt, and Koivisto 2009) was tested in practice by (Michalak et al. 2016) and the authors found that the growth rate resembles $O(6^n)$.

The ODP-IP algorithm is the fastest exact algorithm for the CSG problem to date. However, as we show in this paper, ODP-IP struggles to cope with specific types of inputs which undermine its hybridization approach. Specifically:

1. we show that many operations in ODP-IP involve redundant searches by IDP and IP. Hence, both IDP and IP perform many duplicated operations.
2. we define a new technique to reduce the size of the subspace searched by IP with the help of IDP.
3. we show that ODSS [1] delays the overlapping of IDP and IP operations by dividing the whole subspace of CSG into two disjoint sets: IDPSET and IPSET. In the first stage, each set is assigned to IDP and IP. ODSS performs better by up to 54.15% on benchmark inputs.
4. hence, this paper advances the state-of-the-art by providing a novel hybridization method.

The rest of the paper is organized as follows: Sections 2 and 3 describe the optimal CSG problem and the ODP-IP algorithm. Section 4 shows the limitations of ODP-IP. Section 5 delineates the new techniques used in the ODSS algorithm, while sections 6 and 7 describe the empirical evaluation and draw some conclusions.

2 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, where $l \leq n$. Let v be a characteristic function, where v assigns a real value $v(\mathcal{C})$ to each coalition $\mathcal{C}$. 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: 1) $\mathcal{C}_i, \mathcal{C}_j \neq \emptyset$, $i, j \in \{1, 2, \ldots, k\}$. 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 $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ is defined by $v(\mathcal{CS}) = \sum_{\mathcal{C}_i \in \mathcal{CS}}(v(\mathcal{C}_i))$.*

The optimal solution of CSG is a 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 $\mathcal{CS}^*$. Note that $\{a_1, a_2, \ldots, a_n\}$ and $\{1, 2, \ldots, n\}$ are used interchangeably throughout this paper.

[1]ODSS stands for Overlapping, Dividing the subspace, and Subspace Shrinking.

3 The ODP-IP Algorithm

OOP-IP is a hybrid algorithm, where two algorithms IDP (Rahwan and Jennings 2008) and IP (Rahwan et al. 2009) run in parallel. As soon as any one of IDP or IP returns the final result, ODP-IP terminates. The main difference between IDP and IP is in their working principles. IDP and IP are based on different design paradigms, where each has its own strengths and weaknesses compared to the other.

3.1 The IDP Algorithm

The IDP algorithm is an improved version of the DP algorithm (Yeh 1986). The DP is based on *dynamic programming*: given n agents, to find an optimal partition of the set of agents $\mathcal{A}$, DP starts by computing an optimal partition of every subset $\mathcal{C} \subseteq \mathcal{A}$ with $|\mathcal{C}| = 2$, then DP uses these to compute an optimal partition of every $\mathcal{C} \subseteq \mathcal{A}$ with $|\mathcal{C}| = 3$, and so on, until $|\mathcal{C}| = n$.

The IDP algorithm runs serially just like DP. However, IDP finds an optimal partition of the set of agents $\mathcal{A}$ by computing an optimal partition of all the coalitions of size $|\mathcal{C}| \in \{2, 3, \ldots, \lfloor \frac{2n}{3} \rfloor\}$. IDP uses all these optimal partitions and computes the optimal partition of $\mathcal{A}$.

3.2 The IP Algorithm

The IP is based on the representation of the search spaces of the CSG problem using integer partitions of the number of agents (Rahwan et al. 2009). A subspace is represented by an integer partition of an integer n, where n is the given number of agents. A partition of n is an increasing sequence of positive integers $p_1, p_2, \ldots, p_k$ whose sum is n. Each p_i is called a part of the partition. Let the function $p(n)$ denote the number of partitions of the integer n. As an example, $p(4) = 5$. All the partitions of the integer $n = 4$ are $[4], [1, 3], [2, 2], [1, 1, 2]$, and $[1, 1, 1, 1]$.

In this representation, it is possible to compute upper and lower bounds on the best coalition structure value in each subspace. By comparing the bounds for the different subspaces, it is possible to prune many non-promising subspaces and identify the most promising one. For every promising subspace, IP algorithm constructs multiple search branches of a search tree, where a node in the branch of the tree represents a coalition, and every path (from the root node to a leaf node) represents a partition. IP explores the tree in a depth-first manner. To speed up the search process, IP uses the branch-and-bound technique to avoid the branches of the search tree which have no potential of containing an optimal solution.

3.3 Key observations

Each subspace in the integer partition graph represents an integer partition. In ODSS, we use the parts and the properties of the integer partitions and develop a new way of exploring the CSG subspaces. We observed that the integer partitions of an integer n have some interesting properties defined below:

P$_1$: the highest part of many integer partitions is greater than or equal to $\lceil \frac{n}{2} \rceil$.

P$_2$: the subset-sum of many integer partitions is equal to $\lceil \frac{n}{2} \rceil$. Given an integer partition, the subset-sum problem is to find a subset of parts that are selected from the parts of the given integer partition whose sum adds up to a given number (Kleinberg and Tardos 2006).

P$_3$: the subset-sum of some integer partitions is not equal to $\lceil \frac{n}{2} \rceil$ nor is the highest part of these integer partitions is greater than or equal to $\lceil \frac{n}{2} \rceil$

The above properties can be categorized into two disjoint sets: one with the properties 1 and 2, and the other with the property 3. Clearly, the subspaces following the properties 1 and 2 are always disjoint with the subspaces following the property 3. We know that IDP is a serial algorithm and that it starts exploring the subspaces with the property 1, then the property 2 and so on. IDP cannot explore the subspaces with the property 2 before exploring the subspaces with the property 1. On the other hand, IP can search any of the promising subspaces out of all the subspaces. That means that IP can switch to anywhere in the integer partition graph but that IDP cannot switch.

4 Limitations of ODP-IP

To elaborate the limitations of ODP-IP, we use the integer partition graph for ten agents (cf. Figure 1). Given n agents, the nodes in the integer partition graph are categorized into n levels, where each level $L_i \in \{1, 2, \ldots, n\}$ contains the nodes representing partitions of the integer n containing i parts. For example, any node in level 2 contains the partitions of the integer n which have two parts. In the integer partition graph, each node P represents a set of coalition structures corresponding to this node P. For example, the node $[1, 1, 8]$ in the integer partition graph represents all the coalition structures containing three disjoint coalitions $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ with $|\mathcal{C}_1| = 1$; $|\mathcal{C}_2| = 1$ and $|\mathcal{C}_3| = 8$.

In ODP-IP, the way IDP explores the subspaces in the integer partition graph (cf. Figure 1) can be viewed through the properties of the integer partitions as follows:

1. First, IDP explores the subspaces where the integer partitions associated with these subspaces have a highest part greater than $\lceil \frac{n}{2} \rceil$.
2. Next, IDP explores the subspaces where the subset-sum of the integer partitions associated with these subspaces equals $\lceil \frac{n}{2} \rceil$.
3. Finally, IDP explores the subspaces where the integer partitions associated with these subspaces do not have a part greater than or equal to $\lceil \frac{n}{2} \rceil$ nor does the subset-sum of the integer partitions equals $\lceil \frac{n}{2} \rceil$.

Assume that IDP completes the evaluation of all the coalition sizes 2, 3 and 4. Figure 1 shows that all the shaded subspaces in the integer partition graph are explored by IDP. Now, IP will check only for promising subspaces out of the remaining subspaces. Any subspace out of all these subspaces may be a promising one. We observe that in many cases IP and IDP search the same subspace. This is due to the fact that IDP and IP are based on fundamentally different design paradigms. Let us consider the example given in Figure 1. Assume that the promising

subspaces are searched by IP in the following order: $[1, 1, 8]$, $[1, 2, 7]$, $[1, 1, 1, 7]$, $[1, 3, 6]$, $[1, 1, 2, 6]$, $[1, 1, 1, 1, 6]$, $\ldots$ $[3, 3, 4]$. This sequence of subspaces is the same as the sequence of subspaces explored by IDP. In this scenario, IP and IDP run in parallel, but they both duplicate the same parallel processing.

5 The ODSS Algorithm

ODSS is a hybrid algorithm like ODP-IP, where IDP and IP run in parallel. However, ODSS minimizes the overlapping between IDP and IP by dividing the whole search space of CSG into two disjoint sets of subspaces and proposes a novel subspace shrinking technique to reduce the size of the subspace searched by IP with the help of IDP.

ODSS divides the subspaces into two sets of disjoint subspaces: IDPSET and IPSET and allocates IDPSET to IDP and IPSET to IP. Before going into more details about the subspace division technique, first we formally detail the subset-sum problem.

Subset-sum: We define the subset-sum problem as follows: Given a set (or multiset) of numbers X and a target integer T, does there exist a set (or multiset) $X' \subseteq X$ such that the elements of X' sum to T? Formally, is SUBSETSUM(X')=T? where SUBSETSUM checks if there is a subset X' of the given set X with the sum equal to T.

Example 1 *Assume that the multiset* $X = \{1, 1, 2, 2, 4\}$ *and* $T = 5$*. In this example,* $X' = \{1, 2, 2\}$ *or* $\{1, 4\}$*. The sum of the integers in* $\{1, 2, 2\}$ *and* $\{1, 4\}$ *equals 5.*

We now describe how ODSS divides the whole search space of CSG into IDPSET and IPSET.

5.1 Creating IDPSET

ODSS creates the IDPSET using the properties 1 and 2 of the integer partitions. ODSS assigns the subspaces to IDPSET, where the integer partitions associated with these subspaces follow the properties 1 and 2. Lines 1-7 in algorithm 1 detail the process of computing IDPSET. In line 3, MAXINTEGER($\mathcal{SP}$) returns the highest part in the integer partitions associated with the subspace $\mathcal{SP}$.

For example, given ten agents, consider the subspace $[1, 2, 3, 4]$. Clearly, one of the subsets $\{1, 4\}$ or $\{2, 3\}$ of the set $\{1, 2, 3, 4\}$ sums to $\lceil \frac{10}{2} \rceil = 5$. i.e. SUBSETSUM($X'$) $=$ 5, where X' is $\{1, 4\}$ or $\{2, 3\}$. Hence, ODSS assigns the subspace $[1, 2, 3, 4]$ to IDPSET. On the other hand, the subspace $[1, 2, 7]$ has seven as the highest part which is greater than $\lceil \frac{10}{2} \rceil = 5$. So, ODSS assigns $[1, 2, 7]$ to IDPSET.

5.2 Creating IPSET

ODSS creates the IPSET using the property 3 of the integer partitions. The integer partitions associated with all the subspaces in IPSET follow the property 3. Lines 2-8 in algorithm 1 detail the computing of the IPSET.

In the integer partition graph (cf. Figure 1), ODSS assigns the subspaces $[2, 4, 4]$, $[3, 3, 4]$, $[2, 2, 2, 4]$, $[1, 3, 3, 3]$ and $[2, 2, 2, 2, 2]$ to IPSET since the subset-sum of the integer partitions associated with all these subspaces is not equal to $\lceil \frac{n}{2} \rceil = 5$ and nor is the highest part of the integer partitions associated with any of these subspaces greater than 5.

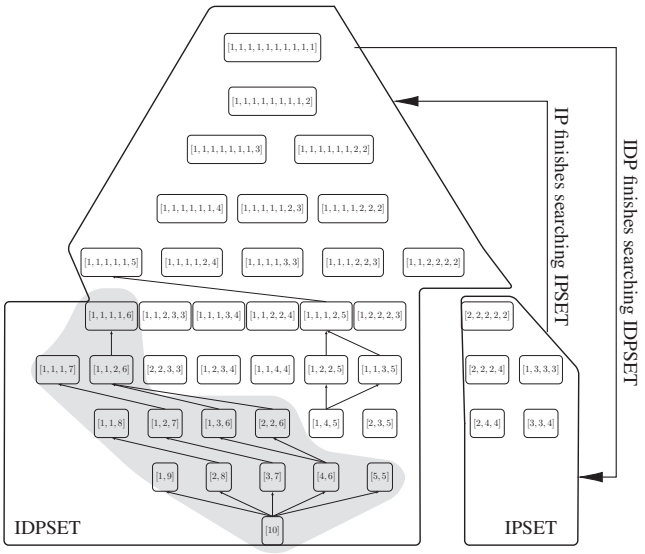

Figure 1: All the subspaces of CSG, given ten agents. The shaded area indicates the subspaces that are explored when IDP finishes evaluating all the coalitions of size four. The white colored subspaces have not yet been searched. Initially, ODSS assigns IDPSET to IDP and IPSET to IP in order to search for the optimal coalition structure.

In Figure 1, we can see that IPSET=$\{[2, 4, 4]$, $[3, 3, 4]$, $[2, 2, 2, 4]$, $[1, 3, 3, 3]$, $[2, 2, 2, 2, 2]\}$ and IDPSET contains the rest of the subspaces.

Algorithm 1 Subspace division technique

Input: Set of all possible subspaces of size 3,4,... $n-1$, given n agents.
Output: Two disjoint sets of subspaces: IDPSET and IPSET.

```
1:  for i = 3 to n − 1 do
2:      for each SP, |SP| == i do
3:          if MAXINTEGER(SP) >= ⌈n/2⌉ or
               SUBSETSUM(SP) == ⌈n/2⌉ then
4:              IDPSET← SP
5:          else
6:              IPSET← SP
7:          end if
8:      end for
9:  end for
10: Return IDPSET and IPSET
```

We now formalize the number of subspaces assigned to IDPSET and IPSET for n agents. The IDPSET contains the subspaces when the highest part (k) in the integer partitions associated with any of the subspaces is greater than or equal to $\lceil \frac{n}{2} \rceil$. It means that one part is fixed, namely k and that the other parts are all the possible integer partitions of

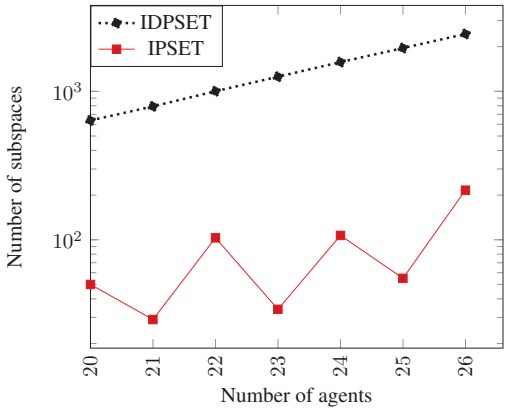

Figure 2: The number of subspaces in IDPSET and IPSET for different numbers of agents.

$n-k$. For example, given ten agents, consider that the highest part in the integer partitions is 7. All the possible subspaces where the integer partitions have a highest part seven are $[3,7]$, $[1,2,7]$, and $[1,1,1,7]$. In all these subspaces, if we delete 7 from these integer partitions, then it is exactly the integer partitions of 3. If the highest part in the integer partitions of the subspace is greater than or equal to $\lceil \frac{n}{2} \rceil$, then the number of subspaces assigned to the IDPSET is defined by the Equation 1. Here, $p(n-i)$ represents the number of distinct partitions of integer $n-i$, where i is the highest part in the integer partitions of subspaces.

$$\sum_{i=\lceil \frac{n}{2} \rceil + 1}^{n-1} p(n-i) \tag{1}$$

In the other case, where the integer partitions have the highest part less than or equal to $\lceil \frac{n}{2} \rceil$, these subspaces in IDPSET are reachable from the subspace $[\lceil \frac{n}{2} \rceil - 1, \lceil \frac{n}{2} \rceil]$ in the integer partition graph. In this case, the integer $\lceil \frac{n}{2} \rceil - 1$ is partitioned in all the possible ways, and in the CSG search space each partition is repeated at most $\lceil \frac{n}{2} \rceil$ times. So, the total number of subspaces in this case is:

$$\sum_{1}^{i=\lceil \frac{n}{2} \rceil} p(n - \lceil \frac{n}{2} \rceil + 1) \tag{2}$$

By combining the Equations 1 and 2, we observe that the IDPSET contains at most $\sum_{i=\lceil \frac{n}{2} \rceil + 1}^{n-1} p(n-i) - 1 + \sum_{1}^{i=\lceil \frac{n}{2} \rceil} p(n - \lceil \frac{n}{2} \rceil + 1)$ subspaces. Figure 2 shows the actual number of subspaces allocated to IDPSET and IPSET for different numbers of agents.

5.3 ODSS Search process

Initially, ODSS assigns IDPSET to IDP and IPSET to IP. Now, three cases may arise.

Case:1 IP finishes searching all the subspaces in IPSET, and IDP finishes exploring all the subspaces in IDPSET. Hence, all the subspaces of the CSG have been explored. ODSS stops and returns the optimal solution. There is no duplicated work done by IP and IDP because IP and IDP finished searching two disjoint sets of subspaces: IPSET and IDPSET. Lines 2-21 in Algorithm 2 show that IDP and IP algorithms run in parallel. IP finishes searching all the promising subspaces in IPSET, and when IP checks for any other promising subspaces in the IDPSET (Line 19 in Algorithm 2), IP finds that all the subspaces in IDPSET have been explored by IDP. Hence, IP stops and returns the final result.

Case:2 IP finishes searching all the subspaces in IPSET, and IDPSET is not fully explored by IDP. In this case, IP starts searching only the promising subspaces in IDPSET (IP ignores the subspaces already explored by IDP). If IP finishes searching all the promising subspaces in IDPSET before IDP explores all the subspaces in IDPSET, ODSS will return the optimal solution. Otherwise, IDP will explore all the subspaces in IDPSET.

Case:3 IDPSET has been explored by IDP, but IPSET is not fully searched by IP. In this case, IDP starts exploring the subspaces in IPSET in a sequential manner. Note that IDP starts searching the subspaces in IPSET irrespective of whether IP has already searched those subspaces or not.

5.4 Searching Multiple Subspaces

We now detail how, with the help of IDP, IP can search more subspaces. Assume that IP is now searching the subspace $I = [i_1, i_2, \ldots, i_k]$ and that, at the same time, IDP already finished evaluating all the coalitions of size $s \in \{2, 3, \ldots s^*\}$ (s^* is the maximum coalition size evaluated by IDP). IP now performs the following steps:

a) **Finding reachable subspaces:**Assume that the current subspace to be searched by IP is I. IP finds the set of all reachable subspaces from the subspace I using the paths already evaluated by IDP in the integer partition graph.

b) **Identifying the integer to split:** To identify a single integer in the subspace I, IP picks an integer $x \in I$ so that splitting x makes it possible to reach 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 $[2,4,4]$ are $[1,2,3,4]$, $[2,2,2,4]$, $[1,1,2,2,4]$, and $[1,1,1,1,2,4]$ by splitting integer four only. IP simulteneously searches these extra subspaces.

As both IDP and IP algorithms primarily work on two disjoint sets of subspaces, their overlapped computations are delayed. We show in the experimental results that this delayed overlap between IDP and IP ensures more positive synergies between them.

5.5 Subspace size reduction

The efficiency of the IP algorithm depends on the input data distributions as well as on the size of the subspaces. In this section, we detail how IP can search more subspaces by making the subspace size smaller using a new technique

Algorithm 2 ODSS Algorithm

Input: Set of all possible non-empty subsets $(2^n - 1)$ of n agents. The value of a coalition $\mathcal{C}$ is $v(\mathcal{C})$. Two disjoint sets of subspaces: IDPSET and IPSET are also given, where IDPSET ∪ IPSET=whole subspaces of CSG.

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

```
IP algorithm sorts IDPSET and IPSET according to the upper bound values of subspaces.
//Begin parallel (IDP algorithm runs in parallel)
for i = 2 to ⌊2n/3⌋ do
    for each C, C ⊂ A, where |C| = i do
        for each C', C' ⊂ C, where 1 ≤ |C'| ≤ |C|/2 do
            if v(C') + v(C \ C') > v(C) then
                v(C) ← v(C') + v(C \ C')
            end if
        end for
    end for
end for
for each C', C' ⊂ A, where 1 ≤ |C'| ≤ |A|/2 do
    if v(C') + v(A \ C') > v(A) then
        v(A) ← v(C') + v(A \ C')
    end if
end for
//End parallel
//Begin parallel (IP algorithm runs in parallel)
for each promising subspace SP ∈ IPSET do
    IP searches the subspace SP
end for
for each promising subspace SP ∈ IDPSET do
    IP searches the subspace SP
end for
Return CS*, v(CS*)
//End parallel
```

called subspace shrinking. Assume that the current subspace IP should search, is $\mathcal{Y}$. The subspace shrinking technique searches for a smaller sized subspace $\mathcal{T}$ and guarantees that, if $\mathcal{T}$ is searched then the subspace $\mathcal{Y}$ will also be searched simultaneously.

The subspace shrinking method visits each node $\mathcal{T}$ in the integer partition graph below the node $\mathcal{Y}$ and checks whether the node $\mathcal{Y}$ is reachable from the node $\mathcal{T}$ by splitting exactly one integer. If IP finds such a node $\mathcal{T}$, then IP searches the node $\mathcal{T}$ and still guarantees to explore the node $\mathcal{Y}$ because IP is searching the nodes $\mathcal{T}$, $\mathcal{Y}$ and other reachable nodes from the node $\mathcal{T}$ simultaneously. On the other hand, if IP does not find the node $\mathcal{T}$, then IP searches the subspace $\mathcal{Y}$ and other subspaces reachable from the node $\mathcal{Y}$ by splitting exactly one integer.

For example, in Figure 1 assume that IDP has evaluated all the coalitions of size $s \in \{2, 3, 4\}$ and that the current upper bound subspace is $\mathcal{Y} = [1, 1, 1, 2, 5]$. Using IP's multiple subspace search technique, IP can search only two subspaces, i.e. $[1, 1, 1, 1, 1, 5]$, and $[1, 1, 1, 2, 5]$. On the other hand using the subspace shrinking method, before searching the subspace $[1, 1, 1, 2, 5]$, IP visits all the nodes $\mathcal{T}$ below the node $\mathcal{Y} = [1, 1, 1, 2, 5]$ in the integer partition graph (cf. Figure 1) and checks if the subspace $[1, 1, 1, 2, 5]$ is reachable from the node $\mathcal{T}$. In our example, IP finds that the node $\mathcal{Y} = [1, 1, 1, 2, 5]$ is reachable from the node $\mathcal{T} = [1, 4, 5]$ by splitting integer 4 only. Now, if IP searches the subspace $\mathcal{T} = [1, 4, 5]$, it will simultaneously search the subspaces $[1, 4, 5]$, $[1, 1, 3, 5]$, $[1, 2, 2, 5]$, $[1, 1, 1, 2, 5]$, and $[1, 1, 1, 1, 1, 5]$. The improvement in the subspace shrinking method is two-fold: First, the size is reduced, thus speeding up IP's depth-first search; second, IP is now searches more subspaces.

Time complexity of the search space division: If the highest part of the integer partition associated with a subspace is greater than $\lceil \frac{n}{2} \rceil$, then this subspace is stored in the IDPSET. In Figure 1 every node is sorted, so the algorithm needs to check the last part of the integer partitions. For example, in the subspace $\mathcal{SP} = [1, 1, 2, 6]$), the last part of the integer partitions is 6. Hence, the algorithm takes constant time to return the highest part of the integer partitions in the subspace $\mathcal{SP}$.

SUBSETSUM($\mathcal{SP}$) returns true if the sum of any subset of parts that are selected from the parts of the integer partitions associated with the subspace $\mathcal{SP}$ equals $\lceil \frac{n}{2} \rceil$. One naive solution strategy is to check all the subsets of the parts from the integer partitions of $\mathcal{SP}$. Therefore, time complexity of the naive solution is exponential.

To compute SUBSETSUM($\mathcal{SP}$), we used a pseudo-poly-time algorithm (Papakonstantinou 2006).

We now prove that our algorithm always finds a solution if it exists and we analyze the computational complexity of ODSS.

Theorem 1 *Given n agents, ODSS 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 ODSS is a hybrid version of IDP and IP. We prove the correctness of ODSS by using the previously established algorithms IDP and IP. The correctness of algorithms IDP (Rahwan and Jennings 2008) and IP (Rahwan et al. 2009) is well established.

In ODSS, IDP and IP start working on IDPSET and IPSET. The optimal coalition structure $\mathcal{CS}^*$ is found if IDP reaches the node P from the bottom node in the integer partition graph, or if IP finishes searching all the feasible coalition structures associated with the node P. The node P represents a subspace which is either in IDPSET or IPSET but not in both since IDPSET and IPSET are disjoint. ODSS stops if all the subspaces in IDPSET and IPSET are searched by IDP or IP. It follows that the node P containing the optimal coalition structure is always found by IDP or IP or by both of them. □

Theorem 2 *Given n agents, ODSS runs in $O(3^n)$ time.*

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

6 Empirical Evaluation

Both ODP-IP and ODSS were implemented in Java, and the experiments were run on an Intel(R) Xeon(R) CPU E7-4830 v3 with 160 GB of RAM. For ODP-IP, we used the code provided by the authors (Michalak et al. 2016). The number of operations performed by IDP is not influenced by the characteristic function at hand, i.e. it depends solely on the number of agents. On the other hand, the number of operations performed by IP (and consequently by ODP-IP and ODSS) depends on the effectiveness of IP's branch-and-bound technique, which in turn depends on the characteristic function at hand. For each distribution, we took an average of 50 tests for each point on Figure 3. We considered the following distributions:

1. **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 the coalitions. Then for each coalition $\mathcal{C}$ in which the 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 (Rahwan, Michalak, and Jennings 2012). That is, $\forall \mathcal{C},\ v(\mathcal{C}) = \sum_{a_i \in \mathcal{C}} p_i^C$.
2. **Agent-based Normal (ABN):** Each agent a_i is assigned a random power $p_i \sim N(10, 0.01)$. 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 (Michalak et al. 2016). That is, $\forall \mathcal{C},\ v(\mathcal{C}) = \sum_{a_i \in \mathcal{C}} p_i^C$.
3. **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.
4. **Beta** (β)**:** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times$ Beta (α, β), where $\alpha = \beta = 0.5$ (Michalak et al. 2016).
5. **Exponential (EXP):** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times$ Exp (λ), where $\lambda = 1$ and n is the number of agents for all the coalitions $C \in 2^A - 1$ (Michalak et al. 2016).
6. **Gamma** (γ)**:** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |\mathcal{C}| \times$ Gamma (x, θ), where $x = \theta = 2$ (Michalak et al. 2016).
7. **Modified Normal (MN):** 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 $b = 0.01$, 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 (Rahwan, Michalak, and Jennings 2012).
8. **Modified Uniform (MU):** The value of each coalition $\mathcal{C}$ is drawn uniformly as $v(\mathcal{C}) \sim U(a, b)$, where $a = 0$ and $b = 10 \times |\mathcal{C}|$, 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 (Service and Adams 2010).
9. **Normally Distributed Coalition Structures (NDCS):** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim N(\mu, \sigma^2)$, where $\mu = |\mathcal{C}|$ and $\sigma = \sqrt{|\mathcal{C}|}$ (Rahwan et al. 2009).
10. **Normal distribution (ND):** Each coalition value is drawn as $v(\mathcal{C}) \sim N(\mu, \sigma^2)$, where $\mu = 10 \times |\mathcal{C}|$ and $\sigma = 0.1$ (Rahwan et al. 2007).
11. **Uniform distribution (UD):** For all the 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).

For each of the above distributions, we plotted the termination times of ODP-IP and ODSS given different numbers of agents (cf. Figure 3). Here, time is measured in seconds. For each distribution and each number of agents, we took an average over 50 runs. As can be observed , for all the aforementioned distributions, ODP-IP is faster in a few cases, while ODSS is faster for other cases. The results show that the search space division technique in the ODSS algorithm provides more positive synergy.

- for example, given 27 agents, with agent-based normal, agent-based uniform, chi-square, modified-uniform, normal, and uniform distributions, the time gain with ODSS is 1808, 1474, 342, 311, 30, and 8 seconds, respectively, compared to ODP-IP. With these distributions, ODSS performs well (cf. Table 1).
- with modified-normal, NDCS, gamma, and exponential distributions, the search space division technique does not work well. When comparing ODSS with ODP-IP, we found that the time loss for these distributions in the case of 27 agents is, respectively 19, 48, 65, 79 seconds.
- We observe that, in most cases the optimal result is given by the IDPSET. Figure 4 shows the percentage of the optimal coalition structure location.

The results in Table 1 show that there are problem instances where the subspace division technique provides

	Time in seconds		P value
Distribution	ODP-IP time (t_1)	ODSS time (t_2)	$\alpha = 0.01$ $P < \alpha$?
Agent-based normal	3339	1531	Yes
Agent-based uniform	3617	2143	Yes
Chi-square	1000	658	Yes
Modified uniform	424	113	No
Normal	152	122	No
Uniform	169	161	No
Beta	1	1	No
Modified normal	18	37	Yes
NDCS	534	582	No
Gamma	358	423	No
Exponential	363	442	No

Table 1: Evaluating the effectiveness of ODP-IP and ODSS. The table shows the runtime (in seconds) for 27 agents.

more positive synergies. A paired-samples t-test was conducted to compare the runtimes of ODSS and ODP-IP for 27 agents. We used a two-tailed, type 1 test. We considered the null hypothesis (H_0): the means of the runtime of ODSS and ODP-IP are the same. We used the value of $\alpha = 0.01$. $P < 0.01$ means that there is very strong evidence against H_0. In Table 1, we can see that in Agent-based normal, Agent-based uniform, Chi-square, and Modified normal distributions, we have strong evidence to reject the H_0. We failed to reject the null hypothesis for other distributions. It means that for other distributions, we don't have evidence to suggest that the runtime means are different.

It is clear that ODSS performance is better for many distributions. The underlying reason is that, at first, the subspace division technique creates two disjoint sets of subspace: IPSET and IDPSET. We observe that in ODP-IP there is no control over the IP algorithm except for the condition that IP searches the subspaces according to the upper bound of the subspaces. On the other hand, in both ODP-IP and ODSS, IDP searches the subspaces sequentially, i.e. IDP evaluates the coalitions sequentially according to the size. In ODP-IP, many subspaces are searched by both IDP and IP.

On the contrary, using the subspace division technique, IDP and IP work on two disjoint subspaces. Hence, in ODSS duplicated operations performed by IDP and IP are minimized. Second, the subspace shrinking method used in the IP algorithm makes it possible to search more subspaces simulteneously.

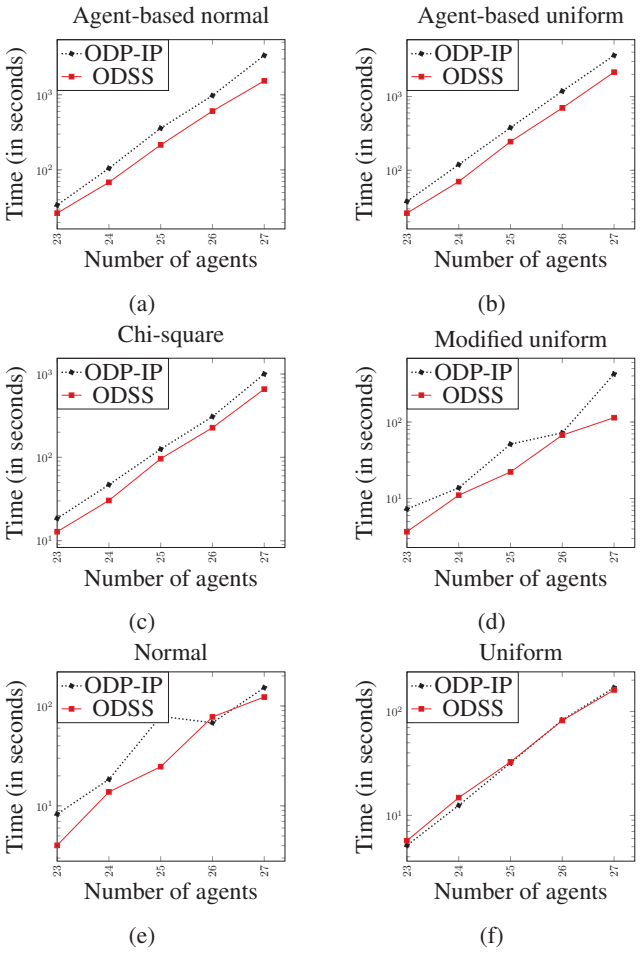

Figure 3: Time performance of ODP-IP vs. ODSS. Here, time is measured in seconds.

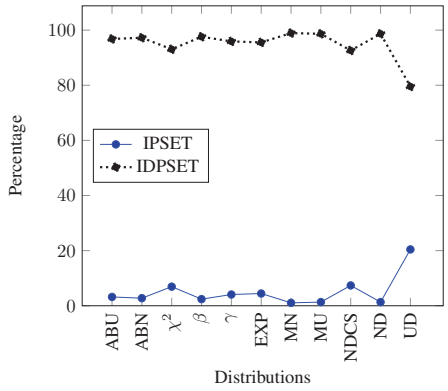

Figure 4: The optimal coalition structure location percentage in IDPSET and IPSET.

7 Conclusion

Coalition structure generation in multiagent systems is a well-known hard problem. Precisely identifying the optimal coalition structure is a hard task, and researchers have come up with different solution techniques to cope with this significant problem. Currently, the best known exact algorithm for the CSG problem is ODP-IP (Michalak et al. 2016). In our paper, we have presented a new guided search method, ODSS, using the IDP and IP algorithms. We found that IP's search techniques frequently overlap with IDP's processing. Thus, we introduced an effective search space division technique, which produces two disjoint sets of subspaces and gives one subspace to IDP and another to IP. By doing so, we reduced the duplicated processing performed by IDP and IP. We also found that, by reducing the size of the subspace with the help of IDP, IP explores more subspaces simultaneously. When experimented over 11 different value distributions, we found that, for some problem instances, ODSS speeds up the process significantly.

8 Acknowledgments

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

References

Björklund, A.; Husfeldt, T.; and Koivisto, M. 2009. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing* 39(2):546–563.

Kleinberg, J., and Tardos, E. 2006. *Algorithm design*. Pearson Education India.

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

Michalak, T.; Rahwan, T.; Elkind, E.; Wooldridge, M.; and Jennings, N. R. 2016. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence* 230:14–50.

Papakonstantinou, P. A. 2006. A pseudo-polynomial time algorithm for subset-sum*. University Lecture.

Rahwan, T., and Jennings, N. R. 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*, 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems.

Rahwan, T.; Ramchurn, S. D.; Dang, V. D.; Giovannucci, A.; and Jennings, N. R. 2007. Anytime optimal coalition structure generation. In *AAAI*, volume 7, 1184–1190.

Rahwan, T.; Ramchurn, S. D.; Jennings, N. R.; and Giovannucci, A. 2009. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research* 34:521–567.

Rahwan, T.; Michalak, T. P.; and Jennings, N. R. 2012. A hybrid algorithm for coalition structure generation. In *AAAI*, 1443–1449.

Sandhlom, T. W., and Lesser, V. R. 1997. Coalitions among computationally bounded agents. *Artificial intelligence* 94(1):99–137.

Service, T. C., and Adams, J. A. 2010. Approximate coalition structure generation. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Shehory, O., and Kraus, S. 1998. Methods for task allocation via agent coalition formation. *Artificial intelligence* 101(1-2):165–200.

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