# BOSS: A Bi-directional Search Technique for Optimal Coalition Structure Generation with Minimal Overlapping

**Paper id 1701**


## Abstract

A major research challenge in multi-agent systems is to form effective coalitions of agents, where coalition formation plays a crucial role. In this paper, we focus on the Coalition Structure Generation (CSG) problem, which involves finding exhaustive and disjoint partitions of agents such that the efficiency of the entire system is optimized. This problem is very challenging due to the exponential size of the search space. Our main contribution is the development of an efficient hybrid algorithm for optimal coalition structure generation called BOSS[1]. When compared to the state-of-the-art against a wide variety of value distributions, BOSS is shown to perform better by up to 33.63% on benchmark inputs. The maximum time gain by BOSS is 3392 seconds for 27 agents against the state-of-the-art.


## 1 Introduction

Coalition formation is widely studied in game theory and has attracted much attention from multi-agent systems. The objective is to form partnerships or teams of agents that cooperate to achieve their common objectives. The value of the cooperation is known a priori for each coalition based on the potential partners. As such, it has been advocated for many applications such as task allocation scenarios (Shehory and Kraus 1998), logistics applications (Sandhlom and Lesser 1997), data fusion in sensor networks (Dang et al. 2006), etc.

The ODP-IP algorithm (Michalak et al. 2016) and the ODSS algorithm (Changder et al. 2020) are the fastest exact algorithms for the CSG problem. Both ODP-IP and ODSS are hybrid versions of the IDP algorithm (Rahwan and Jennings 2008) and the IP algorithm (Rahwan et al. 2009). However, as we show in this paper, ODP-IP and ODSS still need to be improved as they involve redundant operations in the search process by IDP and IP. Moreover, both ODP-IP and ODSS struggles to cope with specific types of inputs which undermine their hybridization approach. Thus, against this background, the contributions of this paper are:

- we propose a novel hybridization method based on IDP and IP algorithms, as well as developing a new bi-directional search algorithm (BOSS). BOSS delays the overlapping of IDP and IP operations by dividing the whole subspace of CSG into $\lfloor \frac{2n}{3} \rfloor$ disjoint sets.
- we prove that BOSS minimizes the duplicate operations performed by IDP and IP.
- the experimental results confirm that BOSS is better by up to 33.63% for the most challenging input distributions[2].

The rest of the paper is organized as follows: section 2 describes the optimal CSG problem. Section 3 describes the related works and their limitations. Section 4 delineates the new techniques used in the BOSS algorithm, while sections 5 and 6 describe the empirical evaluation and draw some conclusions.

## 2 CSG Preliminaries

Let $\mathcal{A}$ be a set of agents $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, $n$ the number of agents in $\mathcal{A}$. We denote $\mathcal{C} = \{a_1, a_2, \ldots, a_l\}$ 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 $k = |\mathcal{CS}|$. $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$ satisfies the following constraints: 1) $\mathcal{C}_i \neq \emptyset$, $i \in \{1, 2, \ldots, k\}$. 2) $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$, for all $i \neq j$. 3) $\bigcup_{i=1}^{k} \mathcal{C}_i = \mathcal{A}$. 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 the CSG problem is a coalition structure $\mathcal{CS}^* \in \Pi^{\mathcal{A}}$, where $\Pi^{\mathcal{A}}$ denotes the set of all the coalition structures over $\mathcal{A}$. Thus, $\mathcal{CS}^* = \arg\max_{\mathcal{CS} \in \Pi^{\mathcal{A}}} v(\mathcal{CS})$.

## 3 Related work

Before going any further, let us introduce briefly the fundamental principles of ODP-IP and ODSS algorithms. ODP-IP runs in parallel the IDP and IP algorithms. They both use a specific design paradigm and have their own strengths and weaknesses. ODP-IP terminates when one of them returns the final result. We remind you that the IDP algorithm improved the DP algorithm (Yeh 1986), which is based on dy-



[1]BOSS stands for **B**i-directional, **O**verlapping minimization and **S**ubspace **S**hrinking

[2]The code of the algorithms and datasets are provided in the supplementary material.

namic 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 $\mathcal{C} \subseteq \mathcal{A}$ where $|\mathcal{C}| = 2$. Then DP uses these to compute an optimal partition of every $\mathcal{C} \subseteq \mathcal{A}$ where $|\mathcal{C}| = 3$, 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 $\mathcal{A}$.

In contrast, IP uses an integer representation of the search space. This representation is based on partitioning the number of agents $n$ in $\mathcal{A}$ (Rahwan et al. 2009). Each subspace is represented by an integer partition of $n$. Let the function $p(n)$ denote the number of partitions of the integer $n$. A partition of $n$ is an increasing sequence of positive integers $p_1, p_2, \ldots, p_k$ the sum of which is $n$. Each $p_i$ is called a part of the partition. For example, the partitions of the integer $n = 4$ are $[4]$, $[1, 3]$, $[2, 2]$, $[1, 1, 2]$, and $[1, 1, 1, 1]$, thus $p(4) = 5$. To prune non-promising subspaces and identify the most promising ones, IP computes the upper and lower bounds on the best coalition structure value in each subspace. For every promising subspace, the 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. To explain how IP and IDP work, let us consider the integer partition graph for ten agents in Figure 1. In this figure, the nodes 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 $\mathcal{C}_3$ where $|\mathcal{C}_1| = 1$; $|\mathcal{C}_2| = 1$ 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 partition associated with these subspaces has 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 $\lceil \frac{n}{2} \rceil$. 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, the sum of which is a given number (Kleinberg and Tardos 2006).
3. Finally, IDP explores the subspaces where the integer partitions associated with these subspaces do not have a part greater than or equal to $\lceil \frac{n}{2} \rceil$ and the subset-sum of the integer partitions does not equal $\lceil \frac{n}{2} \rceil$.

Finally, IP checks only for promising subspaces out of the remaining subspaces not explored by IDP or IP. However, we observed that, in many cases, IP and IDP search the same subspaces. To give an example, let us consider the example given in Figure 1. Assume that the promising subspaces to be searched by IP are 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 to be explored by IDP. Since IP and IDP run in parallel, they both duplicate the same processing.

ODSS (Changder et al. 2020) is a hybrid algorithm, where IDP and IP run in parallel. ODSS minimizes the overlapping between IDP and IP in the ODP-IP algorithm by dividing the whole search space of CSG into two sets of disjoint subspaces, IDPSET and IPSET, and allocates IDPSET to IDP and IPSET to IP. ODSS terminates when IDP or IP finishes searching all the subspaces in IDPSET and IPSET and returns the final result. We observe that ODSS has also conducted many duplicated works for challenging inputs.

# 4 The BOSS Algorithm

BOSS is a hybrid algorithm where IDP and IP run in parallel. BOSS minimizes the overlapping between IDP and IP by dividing the whole search space of CSG into $\lfloor \frac{2n}{3} \rfloor$ disjoint sets of subspaces and proposes a novel bi-directional search using IDP and IP. BOSS divides the whole search space of CSG using the following properties observed on the integer partitions of an integer $n$:

- $\mathbf{P}_1$: the highest part of an integer partition can be greater than $\lceil \frac{n}{2} \rceil$.
- $\mathbf{P}_2$: the highest part or the subset-sum of an integer partition can be equal to $\lceil \frac{n}{2} \rceil$.
- $\mathbf{P}_3$: the subset-sum of an integer partition is not equal to $\lceil \frac{n}{2} \rceil$ and the highest part of the integer partition is less than $\lceil \frac{n}{2} \rceil$.

The subspaces following the properties $P_1$ and $P_2$ are always disjoint with the subspaces following the property $P_3$. We know that IDP starts exploring the subspaces with the property $P_1$, then the property $P_2$ and so on. IDP cannot explore the subspaces with the property $P_2$ before exploring the subspaces with the property $P_1$ because, to explore the subspaces with the property $P_2$, IDP needs to explore the subspaces with the property $P_1$. However, IP can search any of the promising subspaces out of all the subspaces not yet explored by IDP or IP. That means that IP can switch to anywhere in the integer partition graph but that IDP cannot switch.

In the BOSS algorithm, we use the above properties to generate the $\lfloor \frac{2n}{3} \rfloor$ disjoint sets $D_i$, $\forall i \in \{1, 2, \ldots, \lfloor \frac{2n}{3} \rfloor\}$ of subspaces. All the subspaces in the sets $D_i$, $\forall i \in \{1, 2, \ldots \lceil \frac{n}{2} \rceil - 1\}$ follow the property $P_1$. The set $D_{\lceil \frac{n}{2} \rceil}$ follows the property $P_2$, while the sets $D_i$, $\forall i \in \{\lceil \frac{n}{2} \rceil + 1, \ldots, \lfloor \frac{2n}{3} \rfloor\}$ follow the property $P_3$.

First, BOSS considers each subspace $\mathcal{SP}$ in the CSG. If the largest integer in this subspace $\mathcal{SP}$ is $k > \lceil \frac{n}{2} \rceil$, then $\mathcal{SP}$ is added to the set $D_{n-k}$ (cf. Algorithm 1 lines 2-4). In our example, the subspaces $[1, 9]$, $[1, 2, 7]$ are added to the sets $D_1$, and $D_3$ respectively. On the other hand, if

$k = \lceil \frac{n}{2} \rceil$ or $k < \lceil \frac{n}{2} \rceil$ and SUBSETSUM($\mathcal{SP}$) $= \lceil \frac{n}{2} \rceil$, then $\mathcal{SP}$ is added to the set $D_{\lceil \frac{n}{2} \rceil}$, where SUBSETSUM checks whether there is a subset $X$ of the given set $\mathcal{SP}, X \subseteq \mathcal{SP}$ and the parts of $X$ sum to $\lceil \frac{n}{2} \rceil$. For example, given ten agents, the subspaces $[1, 1, 4, 4]$ and $[1, 1, 3, 5]$ are added to the set $D_5$ (cf. Algorithm 1 lines 7-8) because SUBSETSUM $([1, 1, 4, 4]) = 1+4 = 5$, and $\lceil \frac{n}{2} \rceil = 5$ is in $[1, 1, 3, 5]$.

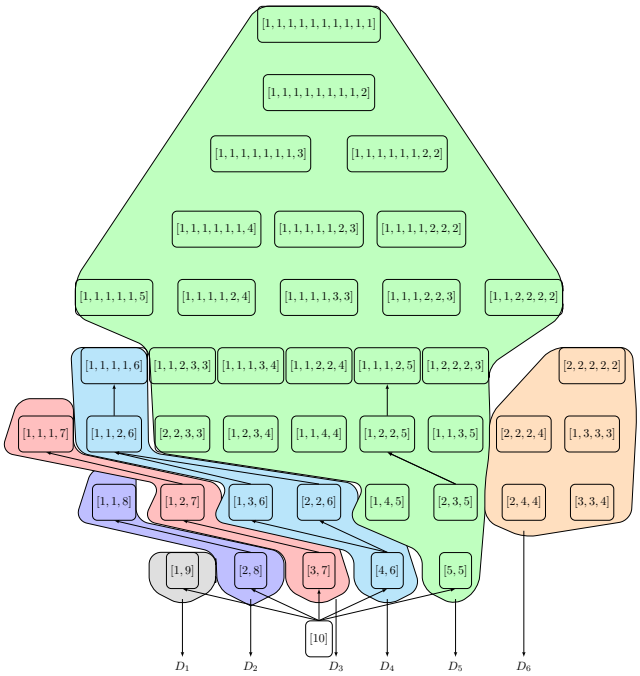


Figure 1: All the subspaces of CSG, given $n = 10$ agents. Each shaded area indicates the subspaces belonging to the set $D_i, \forall i \in \{1, 2, \ldots \lfloor \frac{2n}{3} \rfloor\}$. In BOSS, IDP starts searching the subspaces in an increasing order starting from the set $D_1$, then $D_2$ and so on, whereas IP starts searching the subspaces in a decreasing order starting from the set $D_{\lfloor \frac{2n}{3} \rfloor} = D_6$, then $D_{\lfloor \frac{2n}{3} \rfloor - 1} = D_5$, and so on.

Next, BOSS processes the rest of the subspaces in the integer partition graph and creates the set $D_{\lceil \frac{n}{2} \rceil + 1}$, then the set $D_{\lceil \frac{n}{2} \rceil + 2}$ and so on up to the set $D_{\lfloor \frac{2n}{3} \rfloor}$ as follows. For each $\lceil \frac{n}{2} \rceil < i <= \lfloor \frac{2n}{3} \rfloor$, if SUBSETSUM($\mathcal{SP}$) $= i$, then $\mathcal{SP}$ is added to the set $D_i$ (cf. Algorithm 1 lines 11-12). For example, in Figure 1, the subspace $[2, 4, 4]$ is added to the set $D_6$ because the SUBSETSUM($[2, 4, 4]$) is not equal to $\lceil \frac{n}{2} \rceil = 5$, and the highest part, which is 4, is less than $\lceil \frac{n}{2} \rceil = 5$.

We now formalize the number of subspaces assigned to each set $D_i$, $\forall i \in \{1, 2, \ldots \lfloor \frac{2n}{3} \rfloor\}$. Each set $D_i, i < \lceil \frac{n}{2} \rceil$, contains the subspaces, the highest part of which in the integer partitions is $n - k$. The set $D_{\lceil \frac{n}{2} \rceil}$ contains all the subspaces where the subset-sum of their integer partition is $\lceil \frac{n}{2} \rceil$. This means that the integer partition of each subspace in $D_{\lceil \frac{n}{2} \rceil}$ has the highest part $k \leq \lceil \frac{n}{2} \rceil$. 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 7 are $[3, 7]$, $[1, 2, 7]$, and $[1, 1, 1, 7]$. In all these subspaces, if we delete 7 from their integer partitions, then these partitions are exactly the integer partitions of 3.

**Algorithm 1** Subspace division technique

**Input:** The set $\mathcal{S}$ of all possible subspaces of size 2,3,... $n$, given $n$ agents.
**Output:** $\lfloor \frac{2n}{3} \rfloor$ disjoint sets of subspaces.

1: **for** $i = 1$ to $\lfloor \frac{2n}{3} \rfloor$ **do**
2: **for** each subspace $\mathcal{SP} \in \mathcal{S}$ **do**
// MAXINTEGER($\mathcal{SP}$) returns the highest part in the integer partitions associated with the subspace $\mathcal{SP}$.
3: **if** MAXINTEGER($\mathcal{SP}$) $= n - i$ and $i < \lceil \frac{n}{2} \rceil$ **then**
4: $D_i \leftarrow \mathcal{SP}$
5: $S \leftarrow \mathcal{S} \setminus \mathcal{SP}$
6: **else**
7: **if** SUBSETSUM($\mathcal{SP}$) or MAXINTEGER ($\mathcal{SP}$) $= \lceil \frac{n}{2} \rceil$ and $i = \lceil \frac{n}{2} \rceil$) **then**
8: $D_{\lceil \frac{n}{2} \rceil} \leftarrow \mathcal{SP}$
9: $S \leftarrow \mathcal{S} \setminus \mathcal{SP}$
10: **else**
11: **if** SUBSETSUM($\mathcal{SP}$) $= i$ and $i > \lceil \frac{n}{2} \rceil$ **then**
12: $D_i \leftarrow \mathcal{SP}$
13: $S \leftarrow \mathcal{S} \setminus \mathcal{SP}$
14: **end if**
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: Return $D_1, D_2, \ldots, D_{\lfloor \frac{2n}{3} \rfloor}$.

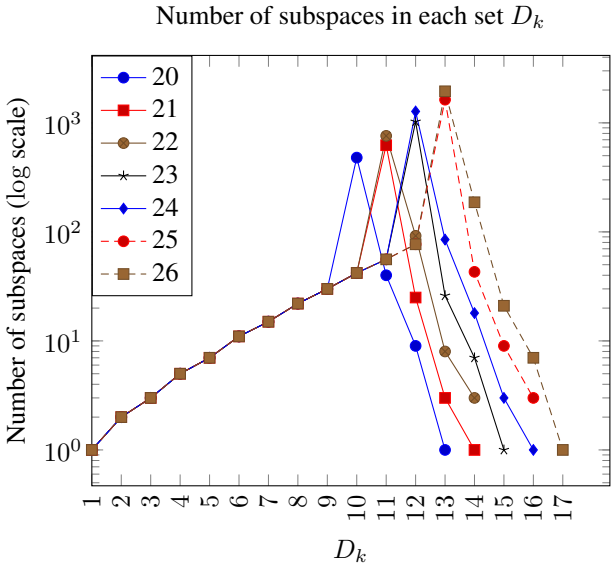


Figure 2: Number of subspaces in each set $D_k$ for different numbers of agents.

If the highest part in the integer partition of the subspace is greater than $\lceil \frac{n}{2} \rceil$, then the number of subspaces assigned to the set $D_i$ is defined by Equation 1. Here, $p(i)$ represents the number of distinct partitions of the integer $i$.

$$|D_i| = p(i),\ \forall i < \lceil \frac{n}{2} \rceil \tag{1}$$

In the case of the subspaces in the set $D_{\lceil \frac{n}{2} \rceil}$, where the integer partitions of the subspaces in $D_{\lceil \frac{n}{2} \rceil}$ have the highest part less than or equal to $\lceil \frac{n}{2} \rceil$, we compute the number of subspaces in each case. Consider the subspace whose integer partition has the highest part $k = \lceil \frac{n}{2} \rceil$. In this case, the integer $n - \lceil \frac{n}{2} \rceil$ is partitioned in all the possible ways. Hence, the number of subspaces in the set $D_{\lceil \frac{n}{2} \rceil}$ whose highest part is $\lceil \frac{n}{2} \rceil$ is computed by $p(n - \lceil \frac{n}{2} \rceil)$ (cf. Figure 1, there are seven subspaces whose maximum part is five, $p(5) = 7$). The subspaces in the set $D_{\lceil \frac{n}{2} \rceil}$ also have the subset-sum equal to $\lceil \frac{n}{2} \rceil$. This means that the same subspaces also have the subset-sum equal to $n - \lceil \frac{n}{2} \rceil$. Hence, the number of subspaces in the set $D_{\lceil \frac{n}{2} \rceil}$ is at the most $p(\lceil \frac{n}{2} \rceil) \times p(n - \lceil \frac{n}{2} \rceil)$.

We observe that the set $D_{\lceil \frac{n}{2} \rceil}$ always contains the highest number of subspaces. For example, given 26 agents, the set $D_{13}$ contains 1948 subspaces. Figure 2 shows the number of subspaces in each $D_k$ for different numbers of agents. In Figure 2, we see a spike for the point $\lceil \frac{n}{2} \rceil$ on the $x$ axis because the subset-sum of a large number of subspaces is equal to $\lceil \frac{n}{2} \rceil$. We also found that $|D_i| \leq |D_{\lceil \frac{n}{2} \rceil}|,\ \forall i \in \{1, 2, \ldots \lfloor \frac{2n}{3} \rfloor\}$ (cf.Figure 2).

## 4.1 The BOSS Search Process

Given the $\lfloor \frac{2n}{3} \rfloor$ disjoint sets of subspaces, IDP starts exploring the sets $D_1, D_2 \ldots$ and so on, while IP starts searching the sets $D_{\lfloor \frac{2n}{3} \rfloor}, D_{\lfloor \frac{2n}{3} \rfloor - 1} \ldots$ and so on. In the midway when IDP and IP meet, BOSS deals with the two following possible cases:

**Case:**1 IP finishes searching all the subspaces in the set $D_i$, and IDP finishes evaluating all the coalitions of size $i - 1$, where $i \leq \lfloor \frac{2n}{3} \rfloor$. Hence, all the subspaces have been explored. BOSS stops and returns the optimal solution. No duplicated work is carried out by IP and IDP because IP and IDP finished searching all the disjoint subspaces: IDP finished searching all the subspaces in the sets $D_1, D_2, \ldots, D_{i-1}$ and IP finished searching all the subspaces in the sets $D_i, D_{i+1}, \ldots, D_{\lfloor \frac{2n}{3} \rfloor}$.

**Case:**2 Both IP and IDP are searching the subspaces in a set $D_i$, where $i \leq \lfloor \frac{2n}{3} \rfloor$. In this case, there are still duplicated operations in the set $D_i$ carried out by IP and IDP. As soon as one of them (IP or IDP) finishes searching the subspaces in $D_i$, BOSS returns the optimal solution.

**Example 1** *Given 10 agents, BOSS divides all the subspaces into* $\lfloor \frac{2n}{3} \rfloor = \lfloor \frac{20}{3} \rfloor = 6$ *disjoint sets of subspaces:* $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, *and* $D_6$. *Let* $\mathcal{S}$ *be the set of all the subspaces in CSG. BOSS creates the following sets:* $D_1$=*{[1,9]},* $D_2$=*{[2,8], [1,1,8]},* $D_3$=*{ [3,7], [1,2,7], [1,1,1,7] },* $D_4$=*{[4,6], [1,3,6], [2,2,6], [1,1,2,6], [1,1,1,1,6]},* $D_6$=*{[2,4,4], [3,3,4], [2,2,2,4], [1,3,3,3], [2,2,2,2,2]}, and* $D_5 = \mathcal{S} \setminus (D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_6)$.

**Algorithm 2** BOSS Algorithm

**Input:** Set of all possible non-empty subsets $(2^n - 1)$ of $n$ agents. The value of a coalition $\mathcal{C}$ is $v(\mathcal{C})$. $\lfloor \frac{2n}{3} \rfloor$ disjoint sets of subspaces $D_i$, where $i \leq \lfloor \frac{2n}{3} \rfloor$ is also given.
**Output:** Optimal coalition structure $\mathcal{CS}^*$ and its value.

The IP algorithm sorts the subspaces inside each set $D_i$, where $1 \leq i \leq \lfloor \frac{2n}{3} \rfloor$ according to the upper bound values of the subspaces in $D_i$.
//Begin parallel (IDP algorithm runs in parallel)
**for** $i = 2$ to $\lfloor \frac{2n}{3} \rfloor$ **do**
  **for** each $\mathcal{C}, \mathcal{C} \subset \mathcal{A}$, where $|\mathcal{C}| = i$ **do**
    **for** each $\mathcal{C}', \mathcal{C}' \subset \mathcal{C}$, where $1 \leq |\mathcal{C}'| \leq \frac{|\mathcal{C}|}{2}$ **do**
      **if** $v(\mathcal{C}') + v(\mathcal{C} \setminus \mathcal{C}') > v(\mathcal{C})$ **then**
        $v(\mathcal{C}) \leftarrow v(\mathcal{C}') + v(\mathcal{C} \setminus \mathcal{C}')$
      **end if**
    **end for**
  **end for**
**end for**
**for** each $\mathcal{C}', \mathcal{C}' \subset \mathcal{A}$, where $1 \leq |\mathcal{C}'| \leq \frac{|\mathcal{A}|}{2}$ **do**
  **if** $v(\mathcal{C}') + v(\mathcal{A} \setminus \mathcal{C}') > v(\mathcal{A})$ **then**
    $v(\mathcal{A}) \leftarrow v(\mathcal{C}') + v(\mathcal{A} \setminus \mathcal{C}')$
  **end if**
**end for**
//End parallel
//Begin parallel (IP algorithm runs in parallel)
**for** $i = \lfloor \frac{2n}{3} \rfloor$ down-to 1 **do**
  **for** each promising subspace $\mathcal{SP} \in D_i$ **do**
    IP searches the subspace $\mathcal{SP}$
  **end for**
**end for**
Return $\mathcal{CS}^*$, $v(\mathcal{CS}^*)$
//End parallel

## 4.2 Subspace Shrinking in BOSS

We now detail how in BOSS, IDP helps IP to reduce the size of the subspace. Assume that IP is now searching the subspace $\mathcal{Y} = [i_1, i_2, \ldots, i_k]$ and that, at the same time, IDP has already finished evaluating all the coalitions of size $s \in \{2, 3, \ldots s^*\}$ ($s^*$ is the maximum coalition size evaluated by IDP). IP 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. The term reachability refers to the relation between the nodes in the integer partition graph. For example, in Figure 1, both the nodes $[1, 1, 1, 7]$ and $[1, 2, 7]$ are reachable from the node $[3, 7]$ because the integer 3 in the node $[3, 7]$ can be split to 1,2 $(3 = 1 + 2)$, which are parts of the node $[1, 2, 7]$, and the integer 2 in the node $[1, 2, 7]$ can be split to 1,1 $(2 = 1 + 1)$, which are parts of the node $[1, 1, 1, 7]$. Splitting an integer means dividing it into several parts so that the sum of these parts yields this number. For example, 1.2 and 1.1.1 are two splits of the integer 3. If IP finds such a

node $\mathcal{T}$, then it searches the node $\mathcal{T}$ and still guarantees that IP will explore the node $\mathcal{Y}$ because IP searches 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. IP picks only one integer $x \in \mathcal{Y}$ so that splitting $x$ makes it possible to reach the largest number of subspaces from $\mathcal{Y}$. For instance, given $\mathcal{Y} = [2, 4, 4]$, by splitting the integer 4, all the subspaces $[1, 2, 3, 4]$, $[2, 2, 2, 4]$, $[1, 1, 2, 2, 4]$, and $[1, 1, 1, 1, 2, 4]$ are reachable from the subspace $[2, 4, 4]$. To better understand the process of splitting exactly one integer, let us consider an example.

**Example 2** *In Figure 1, assume that IDP has evaluated all the coalitions of size $s \in \{2, 3\}$ and the current upper bound subspace to be searched by IP is $\mathcal{Y} = [1, 2, 2, 5]$. IP visits all the nodes $\mathcal{T}$ below the node $\mathcal{Y} = [1, 2, 2, 5]$ in the integer partition graph (cf. Figure 1) and checks whether the subspace $[1, 2, 2, 5]$ is reachable from the node $\mathcal{T}$. IP finds that the node $\mathcal{Y} = [1, 2, 2, 5]$ is reachable from the node $\mathcal{T} = [2, 3, 5]$ by splitting the integer 3. Now, if IP searches the subspace $\mathcal{T} = [2, 3, 5]$, IP will simultaneously search the subspaces $[2, 3, 5]$, $[1, 2, 2, 5]$, and $[1, 1, 1, 2, 5]$. The improvement in this subspace shrinking method is twofold: First, the subspace size is reduced (i.e. the size of the subspace $[1, 2, 2, 5]$ was 4, after subspace shrinking the size of the subspace $[2, 3, 5]$ is 3), thus speeding up IP's depth-first search; second, IP is now searching more subspaces i.e. before subspace shrinking, IP was searching 2 subspaces (i.e.$[1, 1, 1, 2, 5]$, and $[1, 2, 2, 5]$ by splitting the integer 2) simultaneously but after subspace shrinking IP is now searching 3 subspaces simultaneously.*

### 4.3 Time complexity of the search space division

In Algorithm 1, SUBSETSUM($\mathcal{SP}$) $= y$ returns true if the sum of any subset of parts associated with the subspace $\mathcal{SP}$ is same as $y$.

Using the property $P_1$ of integer partition, if the highest part $k$ of the integer partition associated with a subspace is greater than $\lceil \frac{n}{2} \rceil$, then this subspace is stored in the set $D_i$, $\forall i \in \{1, 2, \dots \lceil \frac{n}{2} \rceil - 1\}$. 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 $\mathcal{SP} = [1, 2, 7]$, the last part of the integer partitions is 7. Hence, the algorithm takes constant time to return the highest part of the integer partitions in the subspace $\mathcal{SP}$.

We now prove that our algorithm always finds the optimal solution and we analyze the computational complexity of BOSS.

**Theorem 1** *Given $n$ agents, BOSS 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 $\mathcal{CS}^*$. We prove the correctness of BOSS by using the previously established algorithms IDP and IP. The correctness of the algorithms IDP (Rahwan and Jennings 2008) and IP (Rahwan et al. 2009) is well established.

In BOSS, IDP and IP start working on the sets $D_i,\ \forall i \in \{1, 2, \dots \lfloor \frac{2n}{3} \rfloor\}$ but in opposite directions. 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 in exactly one $D_i,\ \forall i \in \{1, 2, \dots \lfloor \frac{2n}{3} \rfloor\}$. BOSS stops if all the subspaces in $D_i,\ \forall i \in \{1, 2, \dots \lfloor \frac{2n}{3} \rfloor\}$ are searched by IDP or IP. It follows that the node $P$ containing the optimal coalition structure is always found by IDP or IP. □

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

**Proof:** In BOSS, IDP and IP run simultaneously in two different processors and as soon as any of them returns the optimal result, BOSS terminates. Worst case running times of IDP and IP algorithms are $O(3^n)$ and $O(n^n)$. Hence, the time complexity of BOSS is $min(O(3^n), O(n^n)) = O(3^n)$. □

## 5 Empirical Evaluation

To evaluate the algorithms, we used for ODP-IP and ODSS the code provided by the authors (Michalak et al. 2016; Changder et al. 2020). ODP-IP, ODSS, and BOSS were implemented in Java, and the experiments were run on an Intel Xeon E5-2640V2 2.0GHz, 96GB RAM, and 600GB Hard Disk DELL servers. We considered the following distributions from the literature: agent-based uniform (ABU) (Rahwan, Michalak, and Jennings 2012), agent-based normal (ABN) (Michalak et al. 2016), beta, modified normal (MN) (Rahwan, Michalak, and Jennings 2012), modified uniform (MU) (Service and Adams 2010), normal (N) (Larson and Sandholm 2000), and Single Valuable Agent with $\beta$ (SVA-$\beta$) (Travis 2012) distributions. Moreover, we reused the following five distributions from statistics for evaluation of ODP-IP, ODSS, and BOSS algorithms.

1. **Weibull (W):** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim |C| \times$ Weibull $(|\mathcal{C}|)$.
2. **Rayleigh (R):** The value of each coalition $\mathcal{C}$ is drawn as $v(\mathcal{C}) \sim$ Rayleigh $(Modevalue)$, where $Modevalue$ is defined as $10 * \sqrt{\frac{2}{\pi}} * |C|$.
3. **Weighted random with Chisquare (WRC):** First the value of each coalition $\mathcal{C}$ is drawn as $v(C) \sim$ Random $(1, |\mathcal{C}|)$, the value $v \sim \chi^2(\nu)$ is added to $v(C)$, where $\nu = |\mathcal{C}|$ is the degree of freedom.
4. **F:** each coalition value is calculated using F-distribution with degrees of freedom in numerator $Dfnum = 1$ and degrees of freedom in denominator $Dfden_1 = |\mathcal{C}| + 1$.
5. **Laplace or double exponential (LAP):** The value of each coalition $\mathcal{C}$ is drawn as $v(C) \sim$ Laplace $(\mu, \lambda)$ where $\mu = 10 \times |\mathcal{C}|$ and $\lambda = 0.1$, the exponential decay.

Through these experiments, we observed that the number of operations performed by IDP depends only on the number

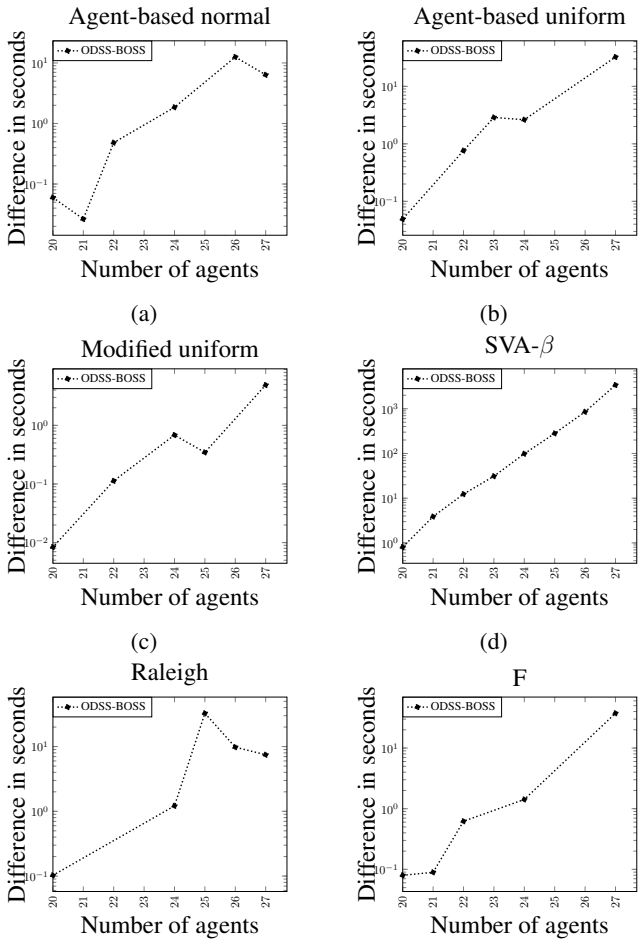


Figure 3: Time difference in seconds. The difference is calculated as ODSS time- BOSS time.

of agents. It is not influenced by the characteristic function. However, the number of operations performed by IP (and consequently by ODP-IP, ODSS and BOSS) depends on the effectiveness of the IP's branch-and-bound technique, which in turn depends on the characteristic function.

We plotted the difference of the termination times of ODSS and BOSS 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 25 runs for all distributions, and imposed a limit of 5 instances when the data sets became large. As can be observed, for all the aforementioned distributions, BOSS is faster. The maximum time gain by BOSS over ODSS is 3392 seconds (cf. Table 1, row 1).

The results show that the search space division technique in the BOSS algorithm provides more positive synergy. For example, given 27 agents, with SVA-$\beta$, F, agent-based uniform, raleigh, agent-based normal, and modified uniform distributions, the time gain with BOSS is 3392, 37, 33, 7, 6, and 6 seconds, respectively, compared to the minimum time taken by ODP-IP and ODSS. With these distributions, BOSS performs well (cf. Table 1). In the case of beta, and weibull

| | Time in seconds | | | |
|---|---|---|---|---|
| Distribution | ODP-IP | ODSS | BOSS | $\Delta t$ |
| **SVA-$\beta$** | 10700 | 10087 | 6695 | 3392 |
| **F** | 682 | 227 | 190 | 37 |
| **ABU** | 2780 | 1455 | 1422 | 33 |
| **R** | 407 | 355 | 348 | 7 |
| **ABN** | 3075 | 1573 | 1567 | 6 |
| **MU** | 1244 | 102 | 96 | 6 |
| **Beta** | 1 | 1 | 1 | 0 |
| **Weibull** | 3 | 3 | 3 | 0 |

Table 1: Evaluating the effectiveness of ODP-IP, ODSS, and BOSS. The table shows the runtime (in seconds) for 27 agents. $\Delta t$ represents $min$(ODP-IP time, ODSS time)-BOSS time in seconds.

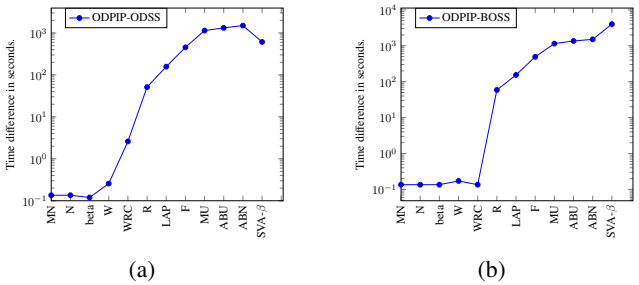


Figure 4: Figure (a) and Figure (b) show the time difference in seconds between ODP-IP and ODSS, ODP-IP and BOSS, respectively, for 27 agents with 12 different distributions.

distributions, the performance of ODP-IP, ODSS, and BOSS is the same.

To show the effectiveness of the new search space division technique in the BOSS algorithm, we compare the runtime differences for 27 agents among ODP-IP, ODSS, and BOSS separately for different value distributions. For clarity we show first in Figure 4a the time difference between ODP-IP and ODSS given 27 agents. This figure reveals that the maximum time gain of ODSS compared to ODP-IP is for agent-based normal distribution.

On the other hand, Figure 4b shows the time difference between ODP-IP and BOSS for all the aforementioned value distributions. We observe that the maximum time gain of BOSS compared to ODP-IP is for SVA-$\beta$ distribution.

The time difference between ODSS and BOSS for 27 agents is shown in Figure 5. This figure shows that the maximum time gain of BOSS over ODSS is also for SVA-$\beta$ distribution. SVA-$\beta$ distribution is a difficult problem for the IP algorithm. In SVA-$\beta$ distribution, every subspace has an upper bound greater than the value of the optimal solution. Hence, the IP algorithm cannot prune the subspaces. Due to the ability of bi-directional search, BOSS gains a huge amount of time in the case of SVA-$\beta$ distribution over ODP-IP and ODSS.

Figure 6 shows the mean of the time difference between ODSS and BOSS for all the problem instances for agents 5 to 27. The maximum difference is for the SVA-$\beta$ distribu-

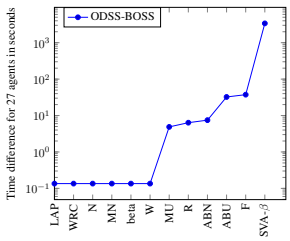


Figure 5: Time difference in seconds between ODSS and BOSS for 27 agents with 12 different distributions.

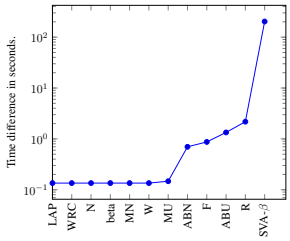


Figure 6: The mean of the time difference in seconds for ODSS and BOSS. The mean is calculated from all the problem instances for agents 5 to 27.

tion.

From the above discussion, we can conclude that BOSS outperforms ODP-IP and ODSS for certain value distributions where IP is unable to generate good upper bound estimates on the value of solutions within each subspace.

To determine whether there are any statistically significant differences between the means of the runtime of ODP-IP, ODSS, and BOSS, we perform one-way analysis of variance (ANOVA). Given 27 agents, for each of the above distributions we calculate the runtime mean for ODP-IP, ODSS, and BOSS. For a comparison of more than two group means, ANOVA is the appropriate method (Green and Salkind 2011). Therefore, a one-way ANOVA was conducted to try to answer the research question: Does a statistical significant relationship exist between the mean of the runtimes for ODP-IP, ODSS, and BOSS for 27 agents. Our null hypothesis $H_0$: There is no statistically significant difference between the runtimes of ODP-IP, ODSS, and BOSS.

Table 2 shows the important ANOVA information for each distribution. Due to the space requirement, we have not shown the exact table of ANOVA for each distribution. We only report three important values obtained from the ANOVA test: $P$, $F$, and $F_c$. The $F$ value is obtained as follows:

$$F = \frac{\text{variance between groups}}{\text{variance within groups}}$$

The critical $F$ value $F_c$ is 3.88 in the F table when the degrees of freedom of numerator and denominator are 2 and 12, respectively, at the $\alpha$ error level 0.05. $P < 0.05$ and $F > F_c$ indicate that there is a very strong evidence against $H_0$.

The results in Table 2 may be interpreted as follows:

| | $F$ | $P$ | $P < \alpha$ | $F > F_c$ |
|---|---|---|---|---|
| **ABU** | 61.79 | 4.8e−07 | Yes | Yes |
| **ABN** | 175.32 | 1.31e−09 | Yes | Yes |
| **SVA-$\beta$** | 82.29 | 9.8e−08 | Yes | Yes |
| **MU** | 5.05 | 0.02 | Yes | Yes |
| **Beta** | 0.90 | 0.431 | No | No |
| **W** | 0.04 | 0.96 | No | No |
| **R** | 0.28 | 0.75 | No | No |
| **F** | 2.55 | 0.11 | No | No |

Table 2: Important ANOVA information for 8 different data distributions with $\alpha$ error level set to 0.05.

- For agent-based uniform, agent-based normal, SVA-$\beta$, and modified uniform data distributions, there is a statistically significant difference among the means of the runtime of the ODP-IP, ODSS and BOSS algorithms at the $\alpha$ error level 0.05. The results suggest us to reject the null hypothesis ($H_0$) that all the runtime means of ODP-IP, ODSS, BOSS are the same, and supports that at least one mean differs from other means.
- For beta, weibull, Raleigh, and F distributions, we failed to reject the null hypothesis $H_0$. This means that for these distributions, we have no evidence to suggest that the runtime means are different.

The results in Table 1 and Table 2 show that there are problem instances where the subspace division technique provides more positive synergies.

It is clear that the performance of BOSS is better for the most challenging input distributions. Using the subspace division technique in BOSS, IDP and IP always work on disjoint subspaces except for the meeting point. Hence, in BOSS, duplicated operations performed by IDP and IP are minimized. Furthermore, the subspace shrinking method used in the IP algorithm makes it possible to search more subspaces simultaneously.

## 6 Conclusion

In this paper, we focused on the CSG problem. Specifically, we proposed a new algorithm for optimal CSG, namely BOSS. This algorithm is based on the parallelism between the IDP and IP algorithms. BOSS uses a new bi-directional search method that limits the overlap between IP and IDP processing. We introduced an effective search space division technique, which produces $\lfloor \frac{2n}{3} \rfloor$ disjoint sets of subspaces and forces IDP and IP to always work on disjoint subspaces except for their meeting point. We also found that, by reducing the size of the subspace with the help of IDP, IP can explore more subspaces simultaneously. When experimented over 12 different value distributions, we found that, for most coalition value distributions, BOSS outperforms the fastest exact algorithms for CSG in the state-of-the-art, ODP-IP and ODSS.

## References

Changder, N.; Aknine, S.; Ramchurn, S. D.; and Dutta, A. 2020. ODSS: Efficient Hybridization for Optimal Coalition

Structure Generation. In *AAAI*, 7079–7086.

Dang, V. D.; Dash, R. K.; Rogers, A.; and Jennings, N. R. 2006. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In *AAAI*, volume 6, 635–640.

Green, S. B.; and Salkind, N. J. 2011. Using SPSS for the MacIntosh and Windows: Analyzing and understanding data.

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.

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.; Michalak, T. P.; and Jennings, N. R. 2012. A Hybrid Algorithm for Coalition Structure Generation. In *AAAI*, 1443–1449.

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.

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.

Travis, S. 2012. *Coalition structure generation in characteristic function games*. Ph.D. thesis, Graduate School of Vanderbilt University.

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