Faster Optimal Coalition Structure Generation via Offline Coalition Selection and Graph-Based Search

Redha Taguelmimt[1], **Samir Aknine**[1], **Djamila Boukredera**[2], **Narayan Changder**[3] and **Tuomas Sandholm**[4,5,6,7]

[1]Univ Lyon, UCBL, CNRS, INSA Lyon, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Lyon, France
[2]Laboratory of Applied Mathematics, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria
[3]TCG Centres for Research and Education in Science and Technology, Kolkata, India
[4]Computer Science Department, Carnegie Mellon University, Pittsburgh, USA
[5]Strategy Robot, Inc.
[6]Strategic Machine, Inc.
[7]Optimized Markets, Inc.
redha.taguelmimt@gmail.com, samir.aknine@univ-lyon1.fr, djamila.boukredera@univ-bejaia.dz, narayan.changder@tcgcrest.org, sandholm@cs.cmu.edu

Abstract

Coalition formation is a key capability in multi-agent systems. An important problem in coalition formation is *coalition structure generation*: partitioning agents into coalitions to optimize the social welfare. This is a challenging problem that has been the subject of active research for the past three decades. In this paper, we present a novel algorithm, SMART, for the problem based on a hybridization of three innovative techniques. Two of these techniques are based on dynamic programming, where we show a powerful connection between the coalitions selected for evaluation and the performance of the algorithms. These algorithms use offline phases to optimize the choice of coalitions to evaluate. The third one uses branch-and-bound and integer partition graph search to explore the solution space. Our techniques bring a new way of approaching the problem and a new level of precision to the field. In experiments over several common value distributions, we show that the hybridization of these techniques in SMART is faster than the fastest prior algorithms (ODP-IP, BOSS) in generating optimal solutions across all the value distributions.

1 Introduction

One of the main challenges in coalition formation is the *coalition structure generation (CSG)* problem: partitioning the agents into disjoint exhaustive coalitions so as to maximize social welfare. (A *coalition structure* is a partitioning of agents into coalitions.) This is a central problem in artificial intelligence and game theory that captures a number of important applications such as collaboration among trucking companies [Sandholm and Lesser, 1997], distributed sensor networks [Dang *et al.*, 2006], etc.

Many algorithms have been developed for this problem. Dynamic programming algorithms [Yeh, 1986; Rahwan and Jennings, 2008; Michalak *et al.*, 2016; Changder *et al.*, 2019; Taguelmimt *et al.*, 2022b] find an optimal solution if it is computationally feasible to run them to completion. Anytime algorithms [Sandholm *et al.*, 1999; Dang and Jennings, 2004; Rahwan *et al.*, 2009; Ueda *et al.*, 2010; Taguelmimt *et al.*, 2022a] provide intermediate solutions during the execution and allow premature termination. Heuristic algorithms [Sen and Dutta, 2000; Ueda *et al.*, 2010; Krausburg *et al.*, 2021; Taguelmimt *et al.*, 2021] focus on speed and do not guarantee that an optimal solution is found.

Even though those algorithms perform well in practice in some cases, hybrid algorithms [Michalak *et al.*, 2016; Changder *et al.*, 2020; Changder *et al.*, 2021; Taguelmimt *et al.*, 2023; Taguelmimt *et al.*, 2024] that combine dynamic programming with integer partition graph search have emerged as the dominant approach to find optimal solutions to this problem. The fastest exact algorithms to date are hybrid solutions called ODP-IP [Michalak *et al.*, 2016], ODSS [Changder *et al.*, 2020], and BOSS [Changder *et al.*, 2021] that combine IDP [Rahwan and Jennings, 2008] and IP [Rahwan *et al.*, 2009]. IDP is based on dynamic programming and computes the optimal solution for n agents by computing an optimal partition of all the coalitions $\mathcal{C}$ of size $|\mathcal{C}| \in \{2, ..., \frac{2n}{3}, n\}$. In contrast, IP uses an integer representation of the search space and computes the optimal solution by traversing in a depth-first manner multiple search trees and uses branch-and-bound to speed up the search. However, the worst-case run time of the state-of-the-art hybrid algorithms is determined by their respective dynamic programming parts, which still need improvement. Also, the hybridization of IDP and IP in these algorithms relies heavily on the effectiveness of IP. Thus, the time required by the algorithms grows considerably when IP is not fast enough. Moreover, these algorithms exhibit very high run times for some distributions.

In light of this, and to enable faster generation of optimal

coalition structures, we develop a new algorithm that combines three complementary techniques to guide the search. The advantage of these techniques is threefold. The first technique, *Complementarity-Based Dynamic Programming (CDP)*, enables SMART to have the best worst-case time performance of all algorithms to date. *GRadual seArch with Dynamic Programming (GRAD)* enables it to find the optimal solution quickly by exploring a minimum number of solution subspaces which shortens the run time. *Distributed Integer Partition Search (DIPS)* further accelerates the search by exploring the subspaces that are most likely to contain the optimal solution. In short, our main contributions are:

- We develop a novel algorithm for optimal CSG that combines three new techniques, resulting in a significant performance improvement. Two of these techniques use offline phases to optimize the search. Moreover, we introduce a new complementarity principle in dynamic programming, where the optimal solution is found by combining the evaluation results of two distinct sets of coalitions. We also propose another principle of gradual search in dynamic programming, where percentages of solution subspaces are searched separately. These principles bring a new way of approaching the problem and a new level of precision to the field.
- We devise the fastest dynamic programming algorithm to date, which bounds the run time, and we propose a new way to speed the search for optimal solutions, while exploring only a part of the search space.
- We show that our algorithm outperforms existing algorithms when generating optimal solutions. We show that it is i) orders of magnitude faster in producing optimal solutions, and ii) more stable in the run time when varying the distributions and the numbers of agents.

2 Preliminaries

The input to a CSG problem is a set of agents $\mathcal{A}$ and a characteristic function v. We say that a CSG problem $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ is of size n. A coalition $\mathcal{C}$ in $\mathcal{A}$ is any non-empty subset of $\mathcal{A}$. The size of $\mathcal{C}$ is $|\mathcal{C}|$, which is the number of agents it contains. A size set is a set of coalition sizes. In a CSG problem, a characteristic function v assigns a real value to each coalition $\mathcal{C}$. A coalition structure $\mathcal{CS}$ is a partition of the set of agents $\mathcal{A}$ into disjoint coalitions. Given a set of non-empty coalitions $\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$, $\bigcup_{j=1}^{k} \mathcal{C}_i = \mathcal{A}$ and for all $i, j \in \{1, 2, ..., k\}$ where $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$. $\Pi(\mathcal{A})$ denotes the set of all coalition structures. The value of a coalition structure $\mathcal{CS}$ is $V(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$. The optimal solution of the CSG problem is the most valuable coalition structure $\mathcal{CS}^* \in \Pi(\mathcal{A})$, that is, $\mathcal{CS}^* = \text{argmax}_{\mathcal{CS} \in \Pi(\mathcal{A})} V(\mathcal{CS})$.

The *integer partition graph* [Rahwan *et al.*, 2007] (see Figure 1) divides the search space into subspaces that are represented by integer partitions of n. Given n agents, each integer partition of n is represented by a node, where the nodes are divided into levels. Each level $l \in \{1, 2, .., n\}$ contains nodes representing integer partitions of n that contain l parts. For instance, level 3 contains nodes where integer partitions of n have 3 parts. Two adjacent nodes are connected if the integer partition in level l can be reached from the one in level $l-1$ by splitting only an integer. Each integer partition $\mathcal{P}$ represents a *set* of coalition structures in which the sizes of the coalitions match the parts of $\mathcal{P}$. For example, the node [1,1,2] represents all coalition structures that contain two coalitions of size 1 and one coalition of size 2. Figure 1 shows a four-agent example of the integer partition graph.

For the remainder of this paper, we use the terms solution subspace and node interchangeably.

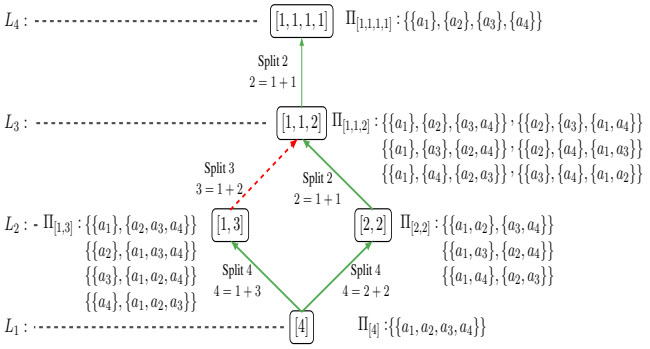

Figure 1: A four-agent integer partition graph.

3 SMART: A Novel CSG Algorithm

The SMART algorithm is based on three techniques (CDP, GRAD and DIPS) that combine dynamic programming with integer partition graph search. SMART introduces new ways of searching the integer partition graph of solutions.

3.1 Complementarity-Based Dynamic Programming (CDP)

CDP is an algorithm that determines the optimal coalition structure. To compute the optimal structure, CDP evaluates different sets of coalitions through two processes (Figure 2), and computes the best partition of each coalition, meaning the best way to split it into potentially multiple subcoalitions. The highest valued coalition structure returned by these processes is an optimal solution. To determine the coalitions to evaluate (that is, for which to compute the best partitions) and ensure that the optimal solution is found, the CDP algorithm uses an offline phase of preprocessing. This phase defines the best pair of coalition size sets to evaluate, such that when combined, the entire solution space is searched. This means that by evaluating these specific sets of coalitions, the CDP algorithm can guarantee that it has considered every possible grouping of agents.

CDP's Offline Phase

The offline phase is one of the key components of the CDP algorithm, which is responsible for determining the coalitions to evaluate in the two processes of the algorithm. This is done by considering the coalition sizes, as illustrated on the integer partition graph (Figure 1). To understand how this works, let us consider an example of four agents. Dividing a coalition of size 2 into two coalitions of size 1, when searching for the

solutions, corresponds to an upward movement in the integer partition graph from the node $[2, 2]$ to the node $[1, 1, 2]$ (2=1+1). By choosing to split in this graph a subset of integers starting from the bottom node, that is, by considering the edges that result from splitting a subset of sizes, a subset of nodes in the integer partition graph becomes reachable from the bottom node, which means that the nodes are connected to the bottom node through a series of edges. Thus, to search a certain number of subspaces, several sets of coalition sizes could be considered, with a different run time for each set. For example, by splitting only the sizes 2 and 4 starting from the bottom node, that is, by deciding to evaluate all the coalitions of sizes 2 and 4 and not those of size 3, all the nodes in the integer partition graph are reachable from the bottom node through a series of edges that result from splitting the coalition sizes 2 and 4. Hence, the set of sizes $\{2, 4\}$ generates 100% of subspaces, as does the set $\{2, 3, 4\}$, but with a lower run time. Hence, some sets of sizes may be more beneficial than others. To find the best size set pair, we propose the *Size Sets Definition (SSD)* algorithm. The SSD algorithm starts by estimating the time required to evaluate the coalitions of each size from 2 to n. This time corresponds to the evaluation time of all the different ways of splitting the coalitions of each size. The estimated time of a splitting is the computational cost associated with this hardware operation, which is a fixed value like any other operation, such as an addition or a subtraction. Then, SSD computes the best pair of coalition size sets that searches all the subspaces with minimum run time.

Then, SSD goes through each possible pair of coalition size sets, and for each pair, it constructs two integer partition graphs by only dividing the integers that belong to each set of sizes (Figure 2). The edges that result from dividing the sizes of the set connect a number of nodes to the bottom node. For a particular pair of sets, in case the generated nodes of the second set of sizes cover all the missed nodes of the first set, meaning that all the solution subspaces are obtained by the first or second set, the SSD algorithm tests whether the pair minimizes the run time. If so, this pair becomes the best pair. The run time of a set of sizes is the sum of the evaluation times of the coalitions of the size set. Given this, the run time of a pair of size sets is the highest run time of the size sets that comprise it.

Algorithm 1 shows how SSD computes the best pair of coalition size sets. The sets of sizes are represented in binary format, with numbers of $n - 2$ bits that represent sizes between 2 and $n - 1$. The coalitions of size 1 and n are not evaluated because the size 1 coalitions are never split and the coalition of size n is always split. For example with five agents, the sets of sizes are $\{2, 3\}$ and $\{2, 4\}$, are represented by the binary numbers 011_2 and 101_2, respectively.

These steps are all executed offline, meaning that we run the SSD algorithm only once for each problem size n (not once for each problem instance) to set up CDP. For example with ten agents, the best pair of coalition size sets that SSD returns is $\mathcal{BS}_1 = \{2, 4, 6, 10\}$ and $\mathcal{BS}_2 = \{2, 8, 10\}$, which together search all the subspaces.

Algorithm 1: Size Sets Definition (SSD) Algorithm

Input: A problem size n, the required time to evaluate the coalitions of each size.
Output: The best pair $\mathcal{BS}_1, \mathcal{BS}_2$ of coalition size sets to search the entire solution subspaces. $\mathcal{BS}_1$ and $\mathcal{BS}_2$ are in binary format.

$\mathcal{BS}_1 \leftarrow 2^{n-2} - 1$ ▷ Initially, the best sizes include all the coalition sizes, i.e. $\{2, .., n-1\}$
$\mathcal{BS}_2 \leftarrow 2^{n-2} - 1$
$x \leftarrow GeneratedSubspaces(2^{n-2} - 1)$ ▷ $GeneratedSubspaces(2^{n-2} - 1)$ returns the set of subspaces for a problem of n agents
$t_1^* \leftarrow SetTime(2^{n-2} - 1)$ ▷ $SetTime$ returns the run time for a set of sizes. The best time for searching the subspaces is initialized to the run time of considering all the sizes
$t_2^* \leftarrow SetTime(2^{n-2} - 1)$
for $i = 1$ *to* $2^{n-2} - 1$ **do** ▷ i corresponds to a set of coalition sizes represented in binary format
  $y \leftarrow GeneratedSubspaces(i)$
  $v \leftarrow x \setminus y$ ▷ v contains the missed nodes when considering the set i
  $t_1 \leftarrow SetTime(i)$
  if $t_1 < t_1^*$ *or* $t_1 < t_2^*$ **then** ▷ the set i has a chance to improve the result
    for $j = i + 1$ *to* $2^{n-2} - 1$ **do**
      $y \leftarrow GeneratedSubspaces(j)$
      if $v \subseteq z$ **then** ▷ the set j generates all the missed nodes of the set i
        $t_2 \leftarrow SetTime(j)$
        if $t_1 < t_1^*$ *and* $t_2 \leq t_2^*$ *or* $t_2 < t_2^*$ *and* $t_1 \leq t_1^*$ **then** ▷ the pair of sets $\{i, j\}$ improves the result
          $\mathcal{BS}_1 \leftarrow y, \mathcal{BS}_2 \leftarrow z$
          $t_1^* \leftarrow t_1, t_2^* \leftarrow t_2$
Add n to $\mathcal{BS}_1$ and $\mathcal{BS}_2$ ▷ n is always considered.
Return $\mathcal{BS}_1, \mathcal{BS}_2$

CDP's Online Phase

The CDP algorithm uses these sets to compute the optimal coalition structure each time a problem instance is to be solved. CDP starts, in a first step, by constructing two tables, the partition table P_t that stores the optimal partition of each coalition $\mathcal{C}$ in $P_t(\mathcal{C})$ and the value table V_t that stores the optimal value of each coalition $\mathcal{C}$ in $V_t(\mathcal{C})$. $P_t(\mathcal{C})$ and $V_t(\mathcal{C})$ are computed for each coalition $\mathcal{C}$ by evaluating all possible ways of splitting $\mathcal{C}$ into two coalitions and checking whether it is beneficial to split it or not. For example, for a coalition of size 4, we evaluate its splitting into a coalition of size 1 and a coalition of size 3 (4=1+3) and into two coalitions of size 2 (4=2+2). This evaluation is done by the two CDP processes, which each consider the coalitions whose sizes belong to the sets returned by SSD. In each process, CDP starts evaluating the smallest coalitions first, as the result of this evaluation is used for evaluating larger coalitions (see Algorithms 2 and 3). In a second step, each process of CDP computes the best

Algorithm 2: The CDP algorithm

Input: Set of all possible coalitions and the value $V_t(\mathcal{C})$ of each coalition $\mathcal{C}$. A number of agents n. Sets of coalition sizes $\mathcal{BS}_1$ and $\mathcal{BS}_2$ to consider by CDP.
Output: An optimal coalition structure $\mathcal{CS}^*$ and its value.
▷ Begin parallel
▷ CDP runs in parallel given $\mathcal{BS}_1$
1 $\mathcal{CS}_1^*, \mathcal{V}_1^* \leftarrow Computing\_Optimal\_CS(\mathcal{V}_t, n, \mathcal{BS}_1)$
▷ CDP runs in parallel given $\mathcal{BS}_2$
2 $\mathcal{CS}_2^*, \mathcal{V}_2^* \leftarrow Computing\_Optimal\_CS(\mathcal{V}_t, n, \mathcal{BS}_2)$
▷ End parallel
3 **if** $\mathcal{V}_1^* > \mathcal{V}_2^*$ **then**
4 $\quad \mathcal{V}^* \leftarrow \mathcal{V}_1^*, \mathcal{CS}^* \leftarrow \mathcal{CS}_1^*$
5 **else**
6 $\quad \mathcal{V}^* \leftarrow \mathcal{V}_2^*, \mathcal{CS}^* \leftarrow \mathcal{CS}_2^*$
7 Return $\mathcal{CS}^*, \mathcal{V}^*$

coalition structure, among the searched subspaces, by computing the best partition of the grand coalition A. Hence, the optimal solution that CDP finds is the highest-valued coalition structure produced by these processes (Figure 2).

Theorem 1 establishes that when considering any pair of coalition size sets, the presence of a path between each node and the bottom node in one of the two integer partition graphs associated with the respective size sets guarantees finding an optimal coalition structure.

Theorem 1. *When considering any pair of coalition size sets to evaluate, if there is a path between each node and the bottom node of one of the two integer partition graphs generated by the two size sets, CDP will fully search the solution subspaces. Thus it finds an optimal coalition structure.*

Proof. The splitting operations of CDP are represented with edges in the integer partition graph (Figure 2). An edge that connects two adjacent nodes, and results from splitting an integer x into two, represents the evaluation of all coalitions of size x by CDP. If there is a path between the bottom node of the integer partition graph and a node $\mathcal{N}$, then all the coalitions that need to be split to find the best solution in $\mathcal{N}$ are evaluated. As all the nodes are at least connected to one of the bottom nodes of the two integer partition graphs generated by the pair of size sets, CDP fully searches each subspace. □

Algorithms 2 and 3 detail the pseudocode of CDP. CDP runs in parallel on the two coalition size sets obtained from the offline phase. In Algorithm 3, CDP computes the optimal coalition structure that belongs to the subspaces searched considering each set. Then, the optimal solution is the highest valued solution of the two (see lines 3-7 of Algorithm 2).

3.2 Gradual Search with Dynamic Programming (GRAD)

The GRAD algorithm uses multiple parallel processes to search for the optimal solution, each with a set of coalition sizes as input with which it explores a certain percentage of the search space. These percentages that we detail in Section 6 are hyperparameters that can be adjusted to fine-tune the algorithm.

GRAD's Offline Phase

GRAD also uses an offline phase to compute, for each considered percentage ω, the best coalition size set that allows one to search this percentage of subspaces with the shortest run time. To find these sets, we introduce the *Size Optimization for diFferent percenTages (SOFT)* algorithm. For each size set $\mathcal{S}$, SOFT constructs the corresponding integer partition graph $\mathcal{G}_\mathcal{S}$ by only dividing the integers that belong to the set $\mathcal{S}$. $\mathcal{G}_\mathcal{S}$ is thus partial, as shown in the example of Figure 2. If this number of generated nodes in $\mathcal{G}_\mathcal{S}$ is at least an ω fraction of the total number of subspaces and $\mathcal{S}$ minimizes the run time, then $\mathcal{S}$ becomes the best set. Algorithm 4 shows how SOFT computes the best coalition size sets. For example with $n = 10$ agents and $\omega = 90\%$, the best coalition size set that SOFT returns is $\{2, 4, 6, 10\}$; it searches 92.86% of subspaces (Figure 2.a).

GRAD's Online Phase

Once these size sets are computed by the SOFT algorithm, each process of GRAD is tuned with the corresponding size set. To solve the problems, GRAD builds a partial integer partition graph with all subspaces and no edges and launches each process with its size set and this partial graph as input (Algorithm 5). Each process of GRAD evaluates all the coalitions whose sizes belong to the best size set obtained from the offline phase and computes their best partitions. The GRAD process evaluates all possible ways of splitting each coalition of the selected sizes into two coalitions and tests whether it is beneficial to split or not. The coalitions are evaluated starting with the smallest ones (Figure 2.a). The result of this evaluation is stored in the partition table P_t and the value table V_t. Once all the coalitions have been evaluated, the GRAD process returns the best coalition structure among the searched subspaces. This is determined by computing the best partition of the grand coalition A using the partition and value tables generated during the evaluation process.

When the optimal solution is in the subspaces explored by a process of GRAD that searches a specific percentage of sub-

Algorithm 3: Computing Optimal $\mathcal{CS}$

Input: The value table $\mathcal{V}_t$. A number of agents n. Set of coalition sizes $\mathcal{BS}$ to consider by CDP.
Output: An optimal coalition structure CS^* and its value.
1 **for** $s \in \mathcal{BS}$ **do**
2 $\quad$ **foreach** $C \subseteq A$, *where* $|C| = s$ **do**
3 $\quad\quad$ **foreach** $C_1, C_2 \subseteq C$, *where* $C_1 \cup C_2 = C$ *and* $C_1 \cap C_2 = \emptyset$ **do**
4 $\quad\quad\quad$ **if** $V_t(C_1) + V_t(C_2) > V_t(C)$ **then**
5 $\quad\quad\quad\quad V_t(C) \leftarrow V_t(C_1) + V_t(C_2)$
6 $\quad\quad\quad\quad P_t(C) \leftarrow \{C_1, C_2\}$
7 $\mathcal{CS}^* \leftarrow$Partition$(A, P_t)$, $\mathcal{V}^* \leftarrow V_t(A)$ ▷ the pseudocode of Partition is in the appendix
8 Return $\mathcal{CS}^*, \mathcal{V}^*$

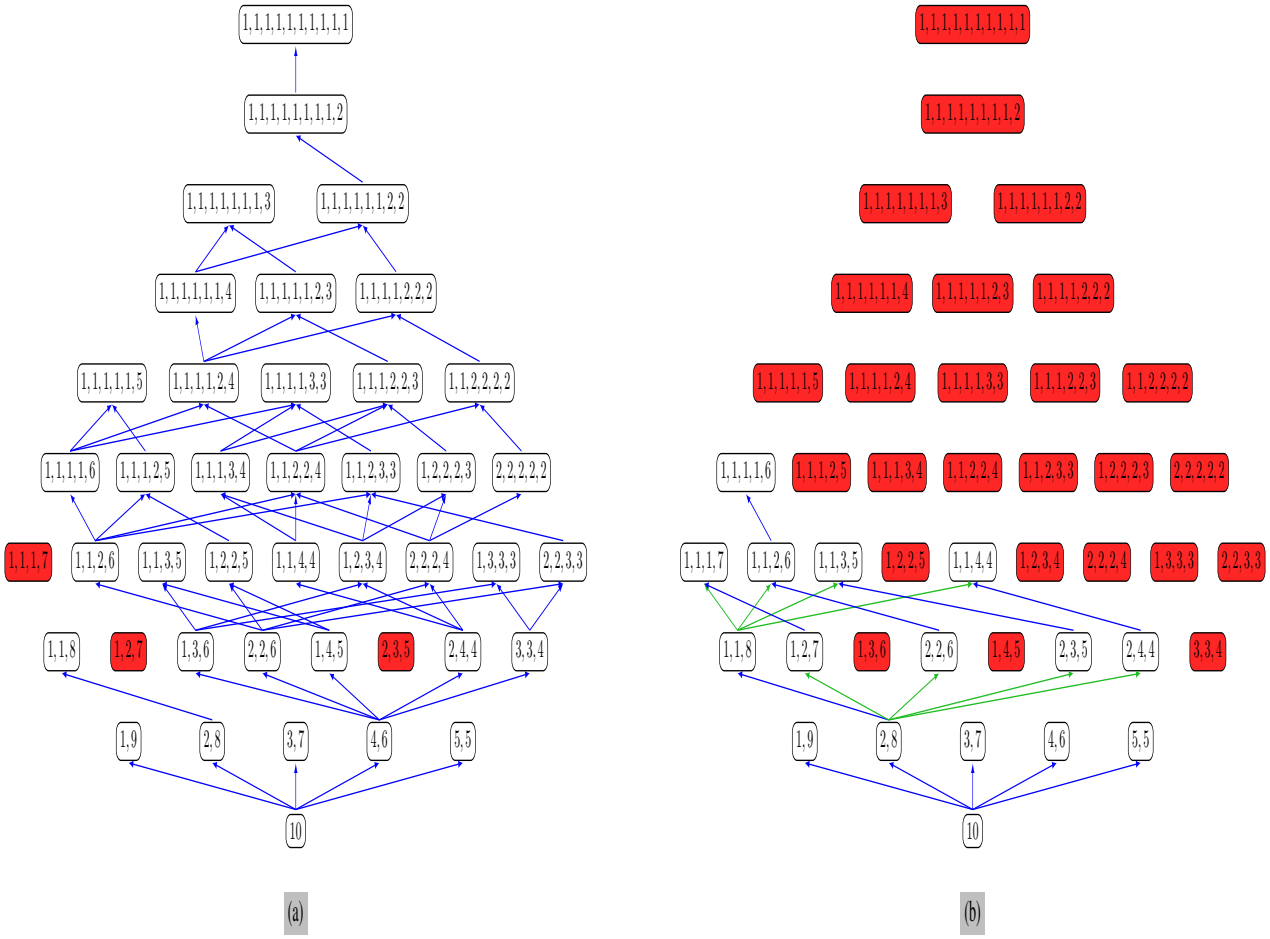

Figure 2: Illustration of CDP on a 10-agent integer partition graph. CDP evaluates the coalitions of size $s \in \mathcal{BS}_1$=$\{2, 4, 6, 10\}$ (Figure 2.a) and those of size $s \in \mathcal{BS}_2$=$\{2, 8, 10\}$ (Figure 2.b) in parallel. With the set $\mathcal{BS}_1$ (resp. $\mathcal{BS}_2$), CDP explores all the subspaces in Figure 2.a (resp. Figure 2.b), except the red ones. Nevertheless, CDP covers all the subspaces using $\mathcal{BS}_1$ and $\mathcal{BS}_2$. Hence, no node is missed by both sets, that is, no node is red in both figures.

spaces $\omega < 100\%$, the process finds it with the shortest run time and enables the other GRAD processes to instantly prune certain subspaces without exploring them. To prune the subspaces, we introduce the upper bound $\mathcal{UB}(\mathcal{N})$ of a subspace $\mathcal{N}$, which is the highest value a coalition structure of this subspace can possibly reach. $\mathcal{UB}(\mathcal{N}) = \sum_{i \in Integers(\mathcal{N})} Max_i$, where Max_i is the maximum value a coalition of size i can take and $Integers(\mathcal{N})$ is the set of integers that form the corresponding integer partition of the subspace $\mathcal{N}$. For instance, for $\mathcal{N} = [1, 4, 5]$, $Integers(\mathcal{N}) = \{1, 4, 5\}$ and $\mathcal{UB}(\mathcal{N}) = \sum_{i \in \{1,4,5\}} Max_i = Max_1 + Max_4 + Max_5$. By comparing the upper bounds of the subspaces, the GRAD processes identify those that have no chance of improving the solution quality and prune the corresponding nodes that do not have a better upper bound than the last best solution found (Line 10 in Algorithm 6). Moreover, after evaluating all the coalitions of size x, a GRAD process updates the integer partition graph by adding the edges that result from splitting x into two integers (Line 8 in Algorithm 6). For example in Figure 2.b, after evaluating all the coalitions of size 8, all the green edges are added to the graph. Hence, a number of nodes become reachable from the bottom node through a series of edges, and the corresponding subspaces are fully searched. Thus, the GRAD process prunes them (Line 9 in Algorithm 6). The subspace pruning, using the upper bounds and the connection to the bottom node, are repeated each time a process of GRAD finishes evaluating the coalitions of each size. Hence, as the size x increases, more subspaces are pruned from the graph. Thus, when all subspaces are pruned, GRAD finishes and returns the optimal solution.

3.3 Distributed Integer Partition Graph Search (DIPS)

DIPS searches the solution subspaces using the integer partition graph. First, DIPS computes the upper bounds of the subspaces and searches them based on their upper bounds. Then, whenever a CDP or GRAD process finishes, while there are still unexplored nodes, DIPS uses that process for a different problem space to parallelize its search. The new process uses the same search technique in DIPS. Thus, the subspaces of

Algorithm 4: The SOFT Algorithm

Input: A CSG problem size n, the required times to evaluate the coalitions of sizes 2 to n-1 for splitting, and the percentage ω of solution subspaces.
Output: The best set of coalition sizes $\mathcal{BS}$ to search at least the percentage ω of the solution subspaces. $\mathcal{BS}$ is in binary format.

$\mathcal{BS} \leftarrow 2^{n-2} - 1$ ▷ Initially, the best set includes all the coalition sizes, i.e. $\{2, .., n-1\}$. $2^{n-2} - 1$ is the binary representation of this set.
$x \leftarrow NumberOfSubspaces(2^{n-2} - 1)$ ▷ The function$NumberOfSubspaces$ returns the number of subspaces for a CSG problem with n agents
$t^* \leftarrow SetTime(2^{n-2} - 1)$ ▷ $SetTime$ returns the run time for a set of sizes. t^* is initialized to the run time when considering all the coalition sizes, which is the worst time
for $i = 1$ *to* $2^{n-2} - 1$ **do** ▷ i corresponds to a set of coalition sizes represented in binary format
  $y \leftarrow NumberOfSubspaces(i)$ ▷ y represents the number of generated subspaces in i
  if $y \geq x \times \omega$ ***and*** $SetTime(i) < t^*$ **then** ▷ the set i generates at least the needed percentage of subspaces
    $\mathcal{BS} \leftarrow i$
    $t^* \leftarrow SetTime(i)$
Add n to $\mathcal{BS}$ ▷ n is always considered.
Return $\mathcal{BS}$

Algorithm 5: The GRAD algorithm

Input: Set of all possible coalitions and the value $V_t(\mathcal{C})$ and $P_t(\mathcal{C})$ of each coalition $\mathcal{C}$. A number of agents n. Sets of coalition sizes $\mathcal{BS}_i$ to consider by GRAD.
Output: An optimal coalition structure $\mathcal{CS}^*$ and its value.

for $i = 1$ ***to*** $|\mathcal{BS}|$ **do** ▷ $|\mathcal{BS}|$ is the number of considered percentages.
  Generate a partial integer partition graph $\mathcal{I}_\mathcal{G}$ with all subspaces and with no edges
  ▷ Begin parallel
  ▷ GRAD process runs in parallel given $\mathcal{BS}_i$
  $\mathcal{CS}^*, \mathcal{V}^* \leftarrow Search\_Process(\mathcal{P}_t, \mathcal{V}_t, n, \mathcal{BS}_i, \mathcal{I}_\mathcal{G})$
  ▷ End parallel
Return $\mathcal{CS}^*, \mathcal{V}^*$

solutions will gradually be distributed between several processes as they are released by CDP or GRAD, which share the subspaces, their upper bounds, and their sorting. This way, each subspace is searched by only one process.

DIPS progressively prunes the subspaces that do not have a better upper bound than the last best solution found. To search a subspace of solutions, a DIPS process constructs several search trees to explore the coalition structures. The

Algorithm 6: Search Process

Input: The partition and value tables $\mathcal{P}_t$ and $\mathcal{V}_t$. A number of agents n. Set of coalition sizes $\mathcal{BS}$ to consider by the process. A partial integer partition graph.
Output: An optimal coalition structure CS^* and its value.

for $s \in \mathcal{BS}$ **do**
  foreach $C \subseteq A$, *where* $|C| = s$ **do**
    foreach $C_1, C_2 \subseteq C$, *where* $C_1 \cup C_2 = C$ *and* $C_1 \cap C_2 = \emptyset$ **do**
      if $V_t(C_1) + V_t(C_2) > V_t(C)$ **then**
        $V_t(C) \leftarrow V_t(C_1) + V_t(C_2)$
        $P_t(C) \leftarrow \{C_1, C_2\}$
  Compute the best solution, $\mathcal{CS}^*, \mathcal{V}^*$, from P_t and V_t
  Add to the integer partition graph the edges that result from splitting the size s
  Prune the subspaces connected to the bottom node ▷ these subspaces are already explored
  Prune the subspaces that do not have a better upper bound than the last best solution found
  if *all the subspaces are pruned* **then**
    Return $\mathcal{CS}^*, \mathcal{V}^*$

nodes of these trees represent coalitions and each path from the root to a leaf represents a coalition structure. Moreover, DIPS applies a branch-and-bound technique to identify and avoid branches that have no chance of containing an optimal solution. An example of this step is given in Figure 3 in the appendix.

4 Hybridization: The SMART Algorithm

We combine CDP, GRAD, and DIPS to make the *coalition-**S**ize opti**M**ization and subsp**A**ce **R**econfigura**T**ion (SMART)* algorithm. Initially, SMART sorts the subspaces by their upper bounds. The DIPS algorithm starts searching with the subspace that has the highest upper bound. Then, DIPS prunes out the subspaces that are either already searched by CDP or GRAD, or that do not have a better upper bound than the last best solution found. CDP and GRAD evaluate the coalitions of the computed sizes obtained from their respective offline phases and allow subspace pruning through intermediate solutions. Whenever a process in CDP or GRAD finishes evaluating the coalitions of any size, they prune out the subspaces that are connected to the bottom node of the integer partition graph through a series of edges because the optimal coalition structure among these subspaces is found by CDP or GRAD. Hence, DIPS does not need to search them. Figure 3 shows how DIPS distributes the search.

Algorithm 7 shows the pseudocode of SMART. We now introduce the following results.

Lemma 1. *CDP is faster than or at least as fast as IDP.*

Proof. Let $\mathcal{S}_{IDP}$ be the set of sizes used by IDP [Rahwan and Jennings, 2008]. $\mathcal{S}_{IDP}$ is hand tuned and is always equal to $\{2, 3, .., \frac{2n}{3}, n\}$ for n agents. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the sets of sizes used by CDP, which are tuned automatically by the

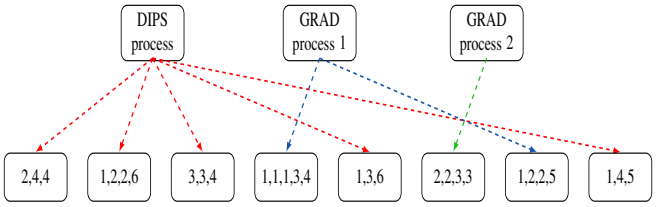

Figure 3: Illustration of the subspace distribution technique. The subspaces are sorted according to their upper bounds. The subspace [2,4,4] is the highest upper bound node and [1,4,5] is the lowest upper bound node. First, DIPS starts by searching the highest upper bound subspaces. Then, each time a GRAD or CDP process is released, DIPS uses the process to expand the parallelism of its search. For example in this Figure, when the released "GRAD process 1" is used by DIPS to search the node [1,1,1,3,4], the "DIPS process" searches another node.

SSD algorithm. $\mathcal{S}_1$ and $\mathcal{S}_2$ are configured to be the pair of sets with the shortest resulting run time. We now show that IDP can not have a better run time than CDP. Let $time(\mathcal{S}_i)$ be the run time produced by the set i. Suppose now that there exists a number of agents for which $\mathcal{S}_{IDP}$ is the best set of sizes to consider and that no pair of sets can produce a better run time. In this case, as the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ find the shortest run time, they must coincide with $\mathcal{S}_{IDP}$, meaning that $max(time(\mathcal{S}_1), time(\mathcal{S}_2)) = time(\mathcal{S}_{IDP})$. Hence, the SSD algorithm would find the sets $\mathcal{S}_1 = \mathcal{S}_{IDP}$ and $\mathcal{S}_2$ is not necessary because $\mathcal{S}_{IDP}$ is sufficient to search the entire solution space and the statement follows. □

Lemma 1 shows a notable property of CDP, and hence of SMART. In particular, it enables us to prove the following theorem. To the best of our knowledge, ODP-IP [Michalak *et al.*, 2016], ODSS [Changder *et al.*, 2020], and BOSS [Changder *et al.*, 2021] are the fastest prior optimal algorithms for the CSG problem.

Theorem 2. *In the worst case, SMART is faster than or at least as fast as ODP-IP, ODSS, and BOSS.*

Proof. SMART uses the CDP algorithm, which relies on having the best pair of coalition size sets that enables fast search of optimal results. By Lemma 1, CDP is faster than IDP. Recall that the fastest exact algorithms are the hybrid solutions, ODP-IP, ODSS, and BOSS, that combine IDP and an integer partition-based algorithm. However, this combination is highly dependent on the efficiency of the integer partition-based algorithm, which in the worst case requires searching all the coalition structures in $O(n^n)$ time [Rahwan *et al.*, 2009], which is infeasible in a reasonable time. Thus, in the worst case, the run time of such hybrid algorithms is determined by the dynamic programming approach. Hence, for hard problems, SMART represents the fastest solution as the dynamic programming algorithm used (CDP) is faster than IDP used by the other algorithms. Formally, let T_{SMART} be the time complexity of SMART, where $T_{SMART} = min(O(n^n), time(GRAD), time(CDP)) \leq time(CDP)$ and let T_{Other} be the time complexity of the other algorithms, where $T_{Other} = min(O(n^n), time(IDP)) = time(IDP)$. Given that $time(CDP) \leq time(IDP)$ by Lemma 1, $T_{SMART} \leq T_{Other}$. □

The result of this hybridization is threefold: (1) CDP is faster than the dynamic programming algorithm IDP (results reported in Section 6). This allows SMART to be faster in the worst case than the state-of-the-art algorithms ODP-IP [Michalak *et al.*, 2016] and BOSS [Changder *et al.*, 2021] because the CDP part of SMART is faster than the IDP algorithm used by ODP-IP and BOSS, and the worst case time performance of these algorithms is determined by their dynamic programming parts; (2) GRAD gradually searches the best size sets for each percentage of solution subspaces. This allows SMART to reach the number of subspaces needed to guarantee finding an optimal solution with the best run time; (3) The integer partitions are distributed among several processes, enabling an efficient and faster search in the integer partition graph.

Algorithm 7: The SMART algorithm

Input: Set of all possible coalitions and the value $v(C)$ of each coalition $\mathcal{C}$ for n agents. Sets of coalition sizes $\mathcal{BS}_1$, $\mathcal{BS}_2$ to consider by CDP and $\mathcal{BS}_i$ to consider by GRAD.
Output: An optimal coalition structure CS^* and its value.

Generate a partial integer partition graph with all subspaces and with no edges
Sort the subspaces by their upper bounds
▷ Begin parallel
▷ CDP runs in parallel with DIPS and GRAD
$\mathcal{CS}^*, \mathcal{V}^* \leftarrow$ CDP$(v, n, \mathcal{BS}_1, \mathcal{BS}_2)$
Return $\mathcal{CS}^*, \mathcal{V}^*$
▷ GRAD runs in parallel with DIPS and CDP
$\mathcal{CS}^*, \mathcal{V}^* \leftarrow$ GRAD$(v, n, \mathcal{BS}_i)$
Return $\mathcal{CS}^*, \mathcal{V}^*$
▷ DIPS runs in parallel with CDP and GRAD
foreach *promising subspace* $\mathcal{PS}$ **do**
  DIPS searches the subspace $\mathcal{PS}$
Return $\mathcal{CS}^*, \mathcal{V}^*$
▷ End parallel

5 Analysis of SMART

In this section, we prove that the SMART algorithm is complete in Theorem 3. Then, we analyze the computational complexity of the algorithms in detail.

Theorem 3. *The SMART algorithm always finds the optimal solution.*

Proof. Each node in the integer partition graph is searched by SMART using the CDP, GRAD, and DIPS algorithms. A node represents a subspace, which contains a number of coalition structures that match the parts of the subspace. The SMART algorithm returns the final solution when all nodes have been searched or pruned. For a particular node that contains the optimal coalition structure, the only way for SMART

not to search it is for DIPS or GRAD to prune it without any of the three algorithms searching it. However, DIPS or GRAD will only prune nodes that have no chance of containing the optimal solution. Thus, such a node would never be pruned, and one of the three algorithms would always completely search the node that contains an optimal solution. □

Time complexity of the SSD algorithm. For each problem size n, the SSD algorithm tests all possible pairs of coalition size sets. For each set of coalition sizes, SSD reconstructs the integer partition graph by dividing only the integers that belong to that set and tests whether it is beneficial to pair that set with another set or not. To test this for one set, CDP pairs it with all the other sets. The total number of sets that SSD evaluates is $2^{n-2}-1$. We denote by $\mathcal{I}(s)$ the number of integer splits performed for a set s and by $p(n,s)$, the number of subspaces generated by the set s. The total number of operations performed by SSD for one set is $\mathcal{T}(n) = \sum_{s=1}^{2^{n-2}-1} \mathcal{I}(s) = \sum_{s=1}^{2^{n-2}-1} \sum_{\mathcal{N}} \mathcal{I}(\mathcal{N},s)$, where $\mathcal{I}(\mathcal{N},s)$ is the number of integer splits performed on node $\mathcal{N}$. The number of splits into two for a certain integer i is $\frac{i}{2}$, the highest integer to split is n, and the highest possible number of integers in a single node is n, which is the number of integers of the node that represents the singleton coalition structure. Thus, $\mathcal{I}(\mathcal{N},s) \leq n \times \frac{n}{2}$, and $\mathcal{T}(n) \leq \sum_{s=1}^{2^{n-2}-1} p(n,s) \times n \times \frac{n}{2}$. However, the growth rate of the number of nodes in the integer partition graph, which is the same as the growth rate of integer partitions of n, is $\mathcal{O}(\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n})$ [Wilf, 2000]. Hence,

$\mathcal{T}(n) \leq \sum_{s=1}^{2^{n-2}-1} \mathcal{O}(\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n}) \times n \times \frac{n}{2} \leq (2^{n-2}-1) \times \mathcal{O}(\frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n}) \times n \times \frac{n}{2}$.

As a result, the total number of operations of SSD when testing one set is $\mathcal{O}(n^2 \times 2^n \times \frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n})$. SSD tests $2^{n-2}-1$ different sets. Thus, the time complexity of SSD is $\mathcal{O}(2^{2n} \times n^2 \times \frac{e^{\pi\sqrt{\frac{2n}{3}}}}{n})$.

Time complexity of the SMART algorithm. SMART combines three algorithms–CDP, GRAD, and DIPS, and runs them in parallel. The worst-case run time of dynamic programming on this problem is $\mathcal{O}(3^n)$ [Yeh, 1986]. CDP and GRAD are based on dynamic programming. They run in parallel on several sets of sizes and terminate when all sets are fully evaluated. Thus, the time complexity of both CDP and GRAD is $\mathcal{O}(3^n)$. The worst-case run time of DIPS, which, in the worst-case, requires us to search all coalition structures, is $\mathcal{O}(n^n)$. As a result, the time complexity of SMART is $\min(\mathcal{O}(3^n), \mathcal{O}(3^n), \mathcal{O}(n^n)) = \mathcal{O}(3^n)$.

6 Empirical Evaluation

We now evaluate the effectiveness of SMART by comparing it to the prior state-of-the-art algorithms ODP-IP and BOSS. We implemented SMART in Java and for ODP-IP and BOSS, we used the codes provided by their authors for the comparisons. They are also written in Java. The algorithms were run on an Intel Xeon 2.30GHz E5-2650 CPU with 256GB of RAM. For GRAD, we considered values of $\omega \in \{10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%, 100\%\}$. We also designed and tested a different version of ODP-IP, namely POI (Parallel ODP-IP), that we developed to integrate parallelism in the baseline version of ODP-IP in order to improve its performance. It uses the same number of processes as SMART (see the appendix for more details). This does not affect the theoretical guarantees but improves the practical performances of the algorithm.

We conducted the experiments on common benchmark problems. We show results on nine value distributions. Results on other distributions are in the appendix. We compared the algorithms using the following value distributions: Modified Normal [Rahwan *et al.*, 2012], Beta, Exponential, Gamma [Michalak *et al.*, 2016], Normal [Rahwan *et al.*, 2007], Uniform [Larson and Sandholm, 2000], Modified Uniform [Service and Adams, 2010], Zipf, SVA Beta and Weibull [Changder *et al.*, 2020]. The experiments shown in the remainder of the paper are also representative of those in the appendix. For each distribution and number of agents, we ran each algorithm 50 times. Figure 4 reports the run times of SMART, BOSS, ODP-IP and POI. On all distributions, SMART was the fastest for all numbers of agents. For example, after 2 seconds, with the Normal distribution for 24 agents, SMART returns optimal solutions roughly 92% faster than BOSS, 91% faster than ODP-IP and 54% faster than POI, while outperforming them by multiple orders of magnitude as can be seen in Figure 4. The reason for this is twofold. First, when problems are hard to solve (see, for instance, the results for Exponential), the CDP part of SMART finishes before the other algorithms as it presents the best worst-case time performance. The second reason is that for problems where the search of a specific percentage of the solution subspaces is sufficient to find the optimal solution, the combination of GRAD and DIPS achieves the best run time. On one hand, GRAD searches that percentage of subspaces with the best run time. On the other hand, DIPS distributes the search to further accelerate it. Notice that the relative contribution of each technique depends on the specific problem instance. Generally, for easier-to-solve instances, DIPS and GRAD play a more significant role in finding the optimal solution, as they target specific subspaces with the upper bound for DIPS and percentages for GRAD. As the problem becomes more difficult, CDP becomes increasingly important for searching a larger portion of the solution space, as it aims to search the entire solution space. In the worst-case scenario, CDP is the fastest technique to search the entire solution space. Hence, all algorithms have a goal and help each other achieve it as explained in Section 4. Additional experimental insights are described in the appendix.

We also report the empirical performance of CDP, which, as discussed earlier in this paper, determines the worst-case run time of SMART. We compared CDP to the dynamic programming algorithm IDP [Rahwan and Jennings, 2008] used by prior hybrid algorithms, and to the fastest dynamic programming algorithm to date, ODP [Michalak *et al.*, 2016]. Notice that ODP, which stands for Optimal DP, is optimal in the sense that it evaluates a minimum number of coalitions to find the optimal solution, not in the sense of run time, mean-

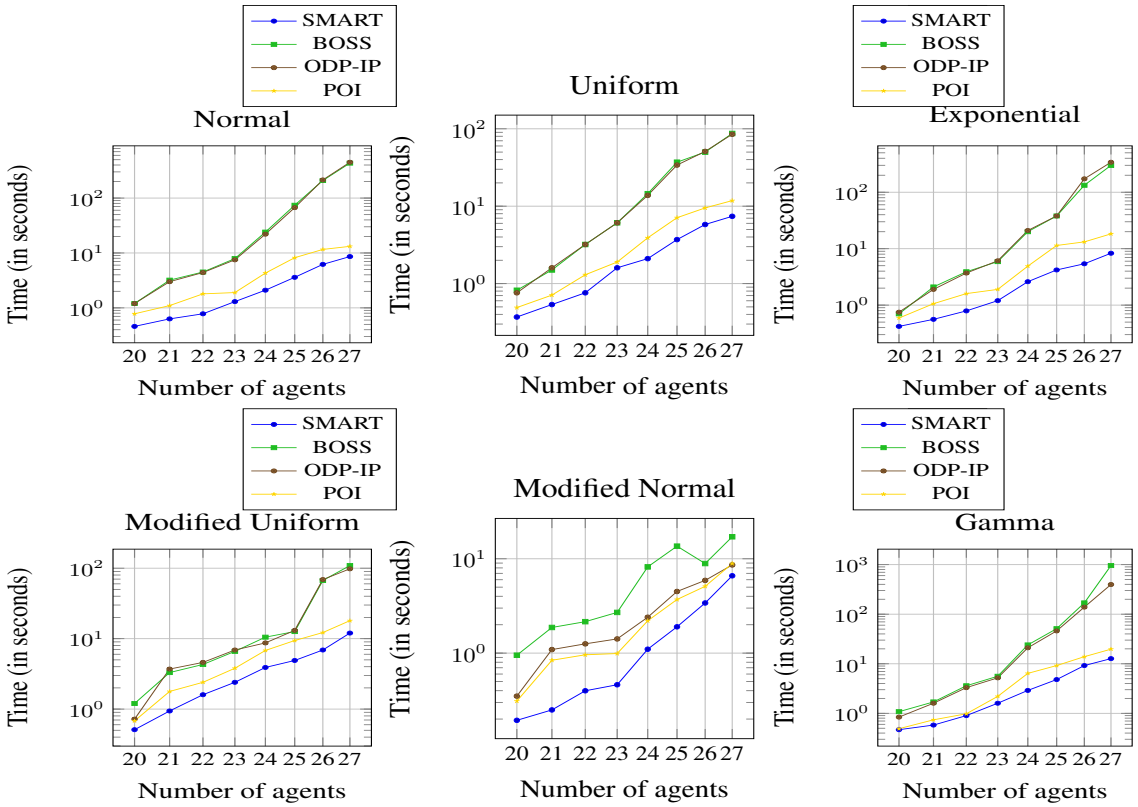

Figure 4: Run time of SMART, BOSS, ODP-IP, and POI.

Number of Agents	Execution Time			
	CDP (t_1)	IDP (t_2)	ODP	P-IDP
20	1.7	3.7	2.2	2.0
21	7.4	15.9	10.3	7.9
22	12.3	24.7	19.8	15.7
23	57	131	79	69
24	205	507	382	257
25	427	887	675	593
26	1781	3659	2846	2267
27	3390	7078	5508	5196

Table 1: Time in seconds of CDP, IDP, ODP, and P-IDP. The time gain is shown in the appendix.

ing that the number of evaluated coalitions is optimal. This does not mean that the resulting run time is optimal. We also compared CDP to a parallel version of IDP (P-IDP). P-IDP uses the same technique presented in [Cruz *et al.*, 2017] and uses two processes to evaluate the coalitions. The evaluation of the coalitions of the same size can be distributed because the coalitions of the same size are independent of each other. Hence, for every coalition size s, each process of P-IDP evaluates half of the coalitions of size s. Table 1 shows the results. The run time of these algorithms depends only on the number of agents. As can be seen, CDP outperforms IDP by at least 50%. Moreover, CDP is also faster than ODP and P-IDP (See the appendix for the time difference between CDP and P-IDP). This experimentally confirms that SMART offers the best worst-case run time. This superior speed appears to translate into the superior practical performance of the SMART algorithm as well. With the most difficult distributions, such as Gamma (see Figure 4), the SMART algorithm is significantly faster than the other algorithms.

7 Conclusion

In this paper, we developed an optimal algorithm, SMART, for the coalition structure generation problem. Our method contributes and combines a number of ideas and techniques. First, we introduced several results concerning the choice of coalitions to evaluate. We used those results to build offline phases to optimize the choice of coalitions to evaluate. Second, we developed three techniques that have different pros. Two of them use the results of the offline phases. The third one uses branch-and-bound and integer partition graph search to explore the solution space. Finally, we combined these techniques by showing how they can assist one another during the search process. Experiments showed that SMART is faster than the fastest prior algorithms on all of the instance distributions for all numbers of agents.

Acknowledgments

Tuomas Sandholm's research is supported by the Vannevar Bush Faculty Fellowship ONR N00014-23-1-2876, National Science Foundation grant RI-2312342, ARO award W911NF2210266, and NIH award A240108S001.

References

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

[Changder *et al.*, 2020] Narayan Changder, Samir Aknine, Sarvapali D Ramchurn, and Animesh Dutta. Odss: Efficient hybridization for optimal coalition structure generation. In *Proc. of AAAI*, pages 7079–7086, 2020.

[Changder *et al.*, 2021] Narayan Changder, Samir Aknine, Sarvapali D. Ramchurn, and Animesh Dutta. Boss: A bidirectional search technique for optimal coalition structure generation with minimal overlapping (student abstract). In *Proc. of AAAI*, volume 35, pages 15765–15766, May 2021.

[Cruz *et al.*, 2017] Francisco Cruz, Antonio Espinosa, Juan C Moure, Jesus Cerquides, Juan A Rodriguez-Aguilar, Kim Svensson, and Sarvapali D Ramchurn. Coalition structure generation problems: optimization and parallelization of the idp algorithm in multicore systems. *Concurrency and computation: Practice and experience*, 29(5):e3969, 2017.

[Dang and Jennings, 2004] Viet Dung Dang and Nicholas R Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 564–571. IEEE Computer Society, 2004.

[Dang *et al.*, 2006] Viet Dung Dang, Rajdeep K Dash, Alex Rogers, and Nicholas R Jennings. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In *Proc. of AAAI*, volume 6, pages 635–640, 2006.

[Krausburg *et al.*, 2021] Tabajara Krausburg, Jürgen Dix, and Rafael H. Bordini. *Feasible Coalition Sequences*, page 719–727. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2021.

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

[Michalak *et al.*, 2016] Tomasz Michalak, Talal Rahwan, Edith Elkind, Michael Wooldridge, and Nicholas R Jennings. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230:14–50, 2016.

[Rahwan and Jennings, 2008] Talal Rahwan and Nicholas R Jennings. An improved dynamic programming algorithm for coalition structure generation. In *Proc. of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[Rahwan *et al.*, 2007] Talal Rahwan, Sarvapali D Ramchurn, Viet Dung Dang, and Nicholas R Jennings. Near-optimal anytime coalition structure generation. In *Proc. of IJCAI*, volume 7, pages 2365–2371, 2007.

[Rahwan *et al.*, 2009] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, and Andrea Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of artificial intelligence research*, 34:521–567, 2009.

[Rahwan *et al.*, 2012] Talal Rahwan, Tomasz Michalak, and Nicholas R Jennings. A hybrid algorithm for coalition structure generation. In *Proc. of AAAI*, pages 1443–1449, 2012.

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

[Sandholm *et al.*, 1999] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1):209–238, 1999.

[Sen and Dutta, 2000] Sandip Sen and Partha Sarathi Dutta. Searching for optimal coalition structures. In *Proceedings Fourth International Conference on MultiAgent Systems*, pages 287–292. IEEE, 2000.

[Service and Adams, 2010] Travis Service and Julie Adams. Approximate coalition structure generation. In *Proc. of AAAI*, pages 854–859, 2010.

[Taguelmimt *et al.*, 2021] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Code-based algorithm for coalition structure generation. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1075–1082, 2021.

[Taguelmimt *et al.*, 2022a] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Pics: Parallel index-based search algorithm for coalition structure generation. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 739–746, 2022.

[Taguelmimt *et al.*, 2022b] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, and Narayan Changder. Subspace-focused search method for optimal coalition structure generation. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1435–1440, 2022.

[Taguelmimt *et al.*, 2023] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, Narayan Changder, and Tuomas Sandholm. Optimal anytime coalition structure generation utilizing compact solution space representation. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 309–316, 8 2023. Main Track.

[Taguelmimt *et al.*, 2024] Redha Taguelmimt, Samir Aknine, Djamila Boukredera, Narayan Changder, and Tuomas Sandholm. Efficient size-based hybrid algorithm for optimal coalition structure generation. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '24, page 2492–2494, 2024.

[Ueda *et al.*, 2010] Suguru Ueda, Atsushi Iwasaki, Makoto Yokoo, Marius Silaghi, Katsutoshi Hirayama, and Toshihiro Matsui. Coalition structure generation based on distributed constraint optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1), Jul. 2010.

[Wilf, 2000] Herbert Wilf. Lectures on integer partitions. 09 2000.

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