Department of Computer Science

An Exact Algorithm for Coalition Structure Generation and Complete Set Partitioning

Talal Rahwan, Tomasz P. Michalak, Edith Elkind, Michael Wooldridge, and Nicholas R. Jennings

CS-RR-13-09

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

An Exact Algorithm for Coalition Structure Generation and Complete Set Partitioning

Talal Rahwan[1], Tomasz Michalak[2], Edith Elkind[3], Michael Wooldridge[2] and Nicholas R. Jennings[1]

[1] *School of Electronics and Computer Science, University of Southampton, UK.*
[2] *Department of Computer Science, University of Oxford, UK.*
[3] *Nanyang Technological University, Singapore.*

Abstract

Solving the *Coalition Structure Generation* problem is a major challenge in cooperative game theory. It involves partitioning the set of agents into subsets (or *coalitions*) such that the total reward is maximized. We study this problem in *Characteristic Function Games*, i.e., scenarios where every possible subset of agents is a potential coalition, and the outcome (or *value*) of every such subset is represented as a single, numerical value on which non-members have no influence. In this setting, the coalition structure generation problem becomes identical to the *Complete Set Partitioning* problem, where every subset of elements has a cost, and the goal is to find a partition in which the sum of subset costs is minimal. This is also identical to the *Winner Determination* problem in combinatorial auctions when a bid is placed on every possible bundle of goods, and the goal is to find a partition that maximizes the profit of the auctioneer.

To date, there are two state-of-the-art, *exact* algorithms for solving this problem: (1) a dynamic-programming algorithm called DP (Yeh, 1986; Rothkopf et al., 1995) and (2) a tree-search algorithm called IP (Rahwan et al., 2009). Each of these two algorithms has it relative strengths and weaknesses compared to the other. More specifically, in terms of worst-case performance, DP is significantly better as it runs in $O(3^n)$ time (given n elements), while IP runs in $O(n^n)$ time. However, when tested against popular value distributions, IP was shown to be often faster than DP (by several orders of magnitude in some cases). Furthermore, IP has the advantage of being *anytime*, meaning that its solution quality improves gradually over time, allowing it to return a valid solution at any moment during its execution. DP, on the other hand, is not an anytime algorithm; it can only return a solution once it has completed its execution.

The contribution of this article is twofold. Firstly, we show that some of DP's operations and memory requirements are redundant. Building upon this, we develop ODP—an optimal version of DP that avoids all redundancies. Secondly, we develop a hybrid algorithm, called ODP-IP, that has the best features of its constituent parts, ODP and IP. Specifically, it is an anytime algorithm just like IP, and runs in $O(3^n)$ just like ODP. Better still, when tested against a wide variety of value distributions, ODP-IP is empirically shown to significantly outperform both ODP and IP for all distributions.

1. Introduction

One of the most important aspects of multi-agent systems is the agents' ability to interact with one another in order to improve their performance and compensate for each other's deficiencies. One means of interaction that has been extensively studied in the literature is to form a *coalition*, i.e., a group of agents that agree to coordinate their activities (and possibly agree on how the

reward from cooperation should be divided among them) in order to achieve a certain objective. Such interaction can be useful both in cases where agents are *cooperative* (*i.e.*, their goal is to maximize the social welfare) as well as cases where they are *selfish* (*i.e.*, each agent maximizes its own reward, regardless of the consequences on others). A wide range of potential applications have been consider in the literature. For instance, by forming coalitions, autonomous sensors can improve their surveillance of certain areas (Han and Poor, 2009), green-energy generators can reduce their uncertainty regarding their expected energy output (Bitar et al., 2012), cognitive radio networks can increase their throughput (Khan et al., 2010), and buyers can obtain cheaper prices through bulk purchasing (Li et al., 2010).

A formal model of a coalition formation scenario is called a cooperative game. Here, given a set of agents, denoted by A, every subset of A is called a coalition, and every partition of A is called a *coalition structure*. Now, if the effectiveness of a coalition is influenced by other co-existing coalitions, then such a scenario is modelled as a *partition function game* (Lucas and Thrall, 1963). On the other hand, a *characteristic function game* models scenarios in which a coalition's effectiveness depends solely on the identities of its members. This assumption simplifies the research questions significantly, and holds in many settings such as those considered by (Han and Poor, 2009), Bitar et al. (2012), and (Khan et al., 2010).[1]

We focus in this article on characteristic function games, and adopt the classical representation of those games, where the worth (or *value*) of any coalition, $C \subseteq A$, is represented using a single, numerical value, returned by a *characteristic function*, $v : 2^A \rightarrow R$. The two research questions that are often studied under this representation are:

- **Coalition Structure Generation**. How to efficiently search through the many possible coalition structures, and find one that maximizes the sum of coalition values.
- **Payoff Distribution**. How should the gains from cooperation be distributed among the coalition members so as to meet certain criteria.

This article focuses on the former research question and, in particular, focuses on developing *exact* algorithms to solve it (i.e., algorithms that are guaranteed to find an optimal solution when run to completion). Interestingly, in characteristic function games, every possible subset of agents is a feasible coalition, and is assigned a real value, making the coalition structure generation problem identical to the *Complete Set Partitioning* problem, where every subset of elements has a cost, and the goal is to find a partition of those elements in which the sum of subset costs is minimal (Lin, 1975). This is also identical to the *Winner Determination* problem in combinatorial auctions when every possible bundle of goods has a (possibly zero-valued) bid placed on it, and the goal is to find a partition of goods that maximizes the profit of the auctioneer (Lehmann et al., 2006).

The main techniques used in exact algorithms for solving NP-hard problems are: (1) dynamic programming, (2) tree search, (3) data preprocessing and (4) local search (Woeginger, 2003). As far as our combinatorial optimization problem is concerned, the two state-of-the-art exact algorithms are based on techniques (1) and (2). Specifically:

- **DP**: This algorithm was originally proposed by Yeh (1986) to solve the *complete set partitioning* problem, and was later on re-discovered by Rothkopf et al. (1995) to solve the *winner determination* problem in combinatorial auctions. The algorithm is based on the *dynamic-programming* principle, which is suitable when the optimization problem can be broken down to smaller problems, and each one of those problems can be broken down to even smaller problems, and so on. A dynamic-programming algorithm solves the smallest problems first,

[1] See the works of Sandholm et al. (1999) and Rahwan et al. (2012) for further examples of both characteristic function game settings and partition function game settings, respectively.

and stores the solutions in memory so that they can be later on used to construct solutions for larger and larger problems, until the original (i.e., largest) problem is solved. Such an approach is efficient whenever the solution to a small subproblem is needed more than once (i.e., it is needed to solve multiple larger problems). The efficiency comes from the fact that such a solution is computed once and stored in memory, thereby avoiding the need to re-compute it every time it is needed (Bellman, 1957). In the coalition structure generation context, the original problem is to find an optimal partition of the set of agents, A, and a "smaller" problem is to find an optimal partition of a subset $C \subseteq A$. DP starts by computing an optimal partition of every subset $C \subseteq A : |C| = 2$, and then uses those to compute an optimal partition of every $C \subseteq A : |C| = 3$, and so on until an optimal partition of A is found.

- **IP**: This algorithm was proposed by Rahwan et al. (2009) based on a representation of the search space, whereby coalition structures are grouped into disjoint subspaces based on the sizes of the coalitions within each structure. With this representation, it is possible to compute upper and lower bounds on the quality of the best solution in each subspace. By comparing the bounds of different subspaces, it is possible to identify, and thus focus on, the promising subspaces. For every such space, the algorithm constructs multiple search trees, where every node represents a coalition, and every path (from a root node to a leaf node) represents a coalition structure. Every such tree is traversed in a depth-first manner. To speed up the search, IP applies a *branch-and-bound* technique to identify, and thus avoid, any branches that have no potential of containing an optimal solution.

As can be seen, the above two algorithms (i.e., DP and IP) are based on different design paradigms, and so exhibit different behaviours. More specifically, in what follows, we provide a comparison of the two algorithms from three different perspectives:

- **Worst case performance:** The time required to exhaustively enumerate all coalition structures is $O(n^n)$, given n agents (Sandholm et al., 1999). Such enumeration, however, involves repeating certain operations multiple times. As mentioned earlier, DP avoids such repetition by storing partial solutions in memory, thereby dropping the required time to $O(3^n)$. On the other hand, the techniques used by IP to speed up the search cannot drop the worst-case time below $O(n^n)$. This is because the effectiveness of IP is strongly influenced by the proportion of the search space that it identifies as being unpromising. This proportion, in turn, depends on the characteristic function at hand (i.e., the values of subsets). It is possible to construct a function for which IP searches the entire space exhaustively.

- **Average performance:** When tested against various characteristic functions drawn from popular value distributions, IP has been shown to be faster than DP, by several orders of magnitude for some distributions (Rahwan et al., 2009). This is because, in practice, IP is able to identify many (if not the vast majority of) subspaces and/or branches of the search trees as being unpromising. DP, on the other hand, is incapable of avoiding any of its operations, regardless of the characteristic function.

- **Being anytime:** IP has the advantage of being *anytime*, meaning that its solution quality improves gradually over time, allowing it to return a valid solution at any moment during its execution. DP, on the other hand, is not an anytime algorithm; it can only return a solution once it has completed its execution. Being anytime is important since the search space grows exponentially with the number of elements to be partitioned, meaning that there might not always be sufficient time to run the algorithm to completion. Moreover, being anytime makes the algorithm more robust against failure; if the execution is stopped before the algorithm

would normally have terminated, then it can still provide a solution that is better than the initial, or any other intermediate, one.

The above comparison shows that each algorithm has its relative strengths and weaknesses compared to the other. With this in mind, we set to develop a new algorithm that exploits the strengths of both DP and IP while, at the same time, avoiding all of their weaknesses. We achieve this goal through the following contributions:

- **Developing ODP**: We draw a link between the workings of DP and the *coalition structure graph*—a graphical representation of the search space due to Sandholm et al. (1999), where every node represents a coalition structure. This link provides an intuitive interpretation of DP's operations; the algorithm evaluates all the movements along the edges of the aforementioned graph, and stores the most beneficial movements in a table. Then, starting from the node where all agents are in one coalition, DP makes a series of movements until it reaches an optimal node. Interestingly, with this visualization, it becomes clear that certain movements are not needed to ensure that an optimal node is reachable. We formalize this observation and build upon it an *Optimal Dynamic Programming* algorithm, called ODP, that performs only one third of DP's operations, without losing the guarantee of finding an optimal solution. Furthermore, by adding a rather insignificant amount of computations, ODP uses only half of the memory required by DP.

- **Developing ODP-IP**: As we will show, ODP and IP approach the optimization problem in different ways. Nevertheless, instead of viewing these as two alternative choices, we develop a new search-space representation that draws a link between the two. Building upon this link, we refine both algorithms, and use the refined versions as building blocks to construct a hybrid approach, called ODP-IP. This algorithm has the best features of its constituent parts: it is an anytime algorithm just like IP, and runs in $O(3^n)$ just like ODP. Better still, when tested against a wide variety of value distributions, ODP-IP is empirically shown to significantly outperform both ODP and IP for all distributions.

For convenience, The remainder of this article is structured as follows. Section 2 formalizes the coalition structure generation problem. Section 3 provides detailed descriptions of IP and DP. Section 4 presents ODP—our optimal version of DP, while Section 5 presents our hybrid algorithm, ODP-IP. The two new algorithms are then evaluated in Section 6. The related literature is discussed in Section 7. Section 8 concludes the article and outlines future work. Appendix A provides a summary of the main notations used throughout the article. Appendices B to K provide proofs of our theorems, while Appendix L discusses a certain aspect of ODP-IP in detail.

2. Preliminaries

In this section, we formally introduce the key definitions and notations used throughout the article. To this end, let $A = \{a_1, a_2, \cdots, a_n\}$ denote the set to be partitioned. When viewed from the *Complete Set Partitioning* perspective, A would be the set of *"elements"*, and when viewed from the *Winner Determination* perspective, A would be the set of *"goods"*. However, in the remainder of this article, we adopt the terminology from the field of *Algorithmic Game Theory*, and refer to A as the set of *"agents"*. Furthermore, we refer to every non-empty subset of A as a *"coalition"*. We denote by $\mathcal{C}^A$ the set of coalitions over A, i.e., $\mathcal{C}^A = \{C : C \subseteq A, C \neq \emptyset\}$. As such, $\left|\mathcal{C}^A\right| = 2^n - 1$. For any two coalitions, $C_1, C_2 \in \mathcal{C}^A$, we write $C_1 < C_2$ when C_1 precedes C_2 lexicographically, e.g., we write $\{a_1, a_3, a_9\} < \{a_1, a_4, a_5\}$ and $\{a_4\} < \{a_4, a_5\}$.

Characteristic function games are formally defined as follows:

Definition 1. *A characteristic function game is a tuple, $\langle A, v\rangle$, where A is the set of agents and v is a characteristic function that assigns a real value to every possible coalition, i.e., $v : \mathcal{C}^A \to \mathbb{R}$.*

A partition of all the agents in the game into disjoint and exhaustive coalitions is called a *coalition structure*.[2] Formally:

Definition 2. *A coalition structure over A is a collection of coalitions $CS = \{C_1, \ldots, C_{|CS|}\}$ that satisfies the following conditions: $\bigcup_{j=1}^{|CS|} C_j = A$, and $\forall i, j \in \{1, \cdots, |CS|\} : i \neq j, C_i \cap C_j = \emptyset$.*

We will denote by Π^A the set of all coalition structures over A. Furthermore, for any coalition structure, $CS \in \Pi^A$, the sum of the values of all the coalitions in CS will be called the *value of CS*, denoted by $V(CS)$. Formally, $V(CS) = \sum_{C' \in CS} v(C')$. Now, we are ready to state our optimization problem formally:

Definition 3. *The coalition structure generation problem in characteristic function games is to find an optimal coalition structure $CS^* \in \Pi^A$, i.e., an (arbitrary) element of the set*

$$\arg\max_{CS \in \Pi^A} V(CS).$$

This problem is computationally hard. It resists brute-force search, as the number of possible coalition structures over n players, which is known as the *Bell number*, B_n (Bell, 1934), satisfies $\alpha n^{n/2} \leq B_n \leq n^n$ for some positive constant α (see, e.g., the work by Sandholm et al., 1999, for proofs of these bounds, and the work by de Bruijn, 1981, for an asymptotically tight bound). Moreover, it is NP-hard to find an optimal coalition structure given oracle access to the characteristic function (Sandholm et al., 1999).

Since every coalition structure represents a possible solution to the coalition structure generation problem, the terms *"coalition structure"* and *"solution"* will be used interchangeably. Furthermore, the set of possible coalition structures will often be referred to as the *"search space"*.

3. The IP Algorithm vs. the DP Algorithm

In this section we provide a detailed description of the main exact algorithms in the literature: (1) IP—the anytime, depth-first search, algorithm by Rahwan et al. (2007), and (2) DP—the dynamic programming algorithm by Yeh (1986).

3.1. The IP Algorithm

The IP algorithm is based on the *integer partition-based representation* (Rahwan et al., 2007) of the space of possible coalition structures. This representation divides the space into disjoint subspaces that are each represented by an *integer partition* of n. Recall that an integer partition of n is a multiset of positive integers, or *parts*, whose sum (with multiplicities) is equal to n (Andrews and Eriksson, 2004). We denote the set of all such integer partitions as $\mathcal{I}^n$. For instance, $\mathcal{I}^4 = \{\{4\}, \{1,3\}, \{2,2\}, \{1,1,2\}, \{1,1,1,1\}\}$. In the IP algorithm, every integer partition, $I \in \mathcal{I}^n$, corresponds to a *subspace*, $\Pi_I^A \subseteq \Pi^A$, consisting of all the coalition structures within which the sizes of the coalitions match the parts of I. For instance, $\Pi_{\{1,1,2\}}^{\{a_1,a_2,a_3,a_4\}}$ is the subspace consisting of all the coalition structures within which two coalitions are of size 1 and one coalition is of size 2. A four-agent example is shown in Figure 1.

[2] We note that some recent studies have considered games where a coalition structure may contain overlapping coalitions; see, e.g. (Chalkiadakis et al., 2010).

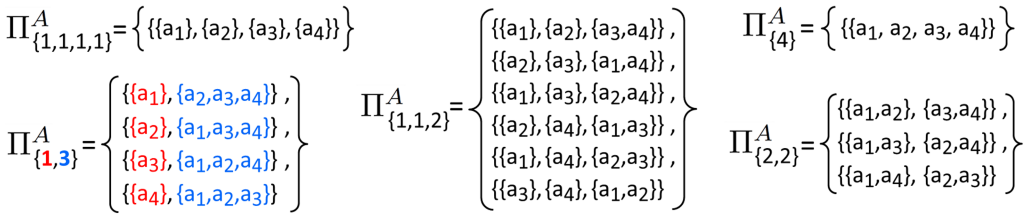

Figure 1: A four-agent example of the integer partition-based representation, where $A = \{a_1, a_2, a_3, a_4\}$.

With this representation, it is possible to compute upper and lower bounds on the value of the best coalition structure that can be found in each subspace. More formally, for every coalition size $s \in \{1, 2, \ldots, n\}$, let $\mathcal{C}^A_s$ denote the set of all the possible coalitions of size s. Furthermore, let Max_s and Avg_s be the maximum and average values of the coalitions in $\mathcal{C}^A_s$, respectively. It turns out that, by computing Avg_s for all $s \in \{1, 2, \ldots, n\}$, it is possible to compute the average value of the coalition structures in each subspace $\Pi^A_I : I \in \mathcal{I}^n$ using the following theorem (the proof can be found in (Rahwan et al., 2009)):

Theorem 1. *For any $I \in \mathcal{I}^n$, let $I(i)$ be the multiplicity of i in I, then:*

$$\frac{\sum_{CS \in \Pi^A_I} V(CS)}{|\Pi^A_I|} = \sum_{i \in I} I(i) \cdot Avg_i$$

Based on this, for every $I \in \mathcal{I}^n$, it is possible to compute a *lower bound* LB_I on the value of the best coalition structure in Π^A_I. This bound is simply equal to the average coalition structure value in Π^A_I, which (according to Theorem 1) can be computed easily as follows: $LB_I = \sum_{i \in I} I(i) Avg_i$. Similarly, it is possible to compute an *upper bound* UB_I on the value of the best coalition structure in Π^A_I as follows: $UB_I = \sum_{i \in I} I(i) Max_i$. Using these bounds, the algorithm computes an upper bound $UB^* = \max_{I \in \mathcal{I}^n} UB_I$ and a lower bound $LB^* = \max_{I \in \mathcal{I}^n} LB_I$ on the value of the optimal coalition structure CS^*. Computing UB^* allows for establishing a bound on the quality of CS^{**}—the best coalition structure found by the algorithm at any point in time; this bound is $\beta = UB^*/V(CS^{**})$. On the other hand, computing LB^* allows for identifying any subspaces that have no potential of containing an optimal coalition structure, which are $\Pi^A_I : UB_I < LB^*$. These subspaces are pruned from the search space. As for the remaining subspaces, the algorithm searches them one at a time. During this search, if a solution is found whose value is greater than $V(CS^{**})$, then the algorithm updates CS^{**} (by setting it to the newly found solution) and updates LB^* (by setting it to $V(CS^{**})$ if necessary, i.e., iff $LB^* < V(CS^{**})$). Now if LB^* is updated, then the algorithm repeats the attempt of pruning unpromising subspaces, i.e., those whose upper bounds are smaller than the updated LB^*. The order by which the subspaces are searched is always based on the upper bounds (i.e., out of all the remaining subspaces, the one with the highest upper bound is searched first). Next, we explain how a subspace is searched.

The process of searching a subspace, say $\Pi^A_I : I = \{i_1, \ldots, i_{|I|}\}$, is done in a *depth-first* manner: the algorithm iterates over the coalitions in $\mathcal{C}^A_{i_1}$ and, for every $C_1 \in \mathcal{C}^A_{i_1}$ that the algorithm encounters, it iterates over the coalitions in $\mathcal{C}^A_{i_2}$ that do not overlap with C_1. Similarly, for every $C_2 \in \mathcal{C}^A_{i_2}$ that the algorithm encounters, it iterates over the coalitions in $\mathcal{C}^A_{i_3}$ that do not overlap with $C_1 \cup C_2$, and so on. This process is repeated until the last set, $\mathcal{C}^A_{i_{|I|}}$, is reached. For every $C_{|I|} \in \mathcal{C}^A_{i_{|I|}}$ that the algorithm encounters, it would have selected a combination of $|I|$ coalitions, namely $\{C_1, C_2, \ldots, C_{|I|}\}$, which is guaranteed to be a coalition structure in Π^A_I. The algorithm

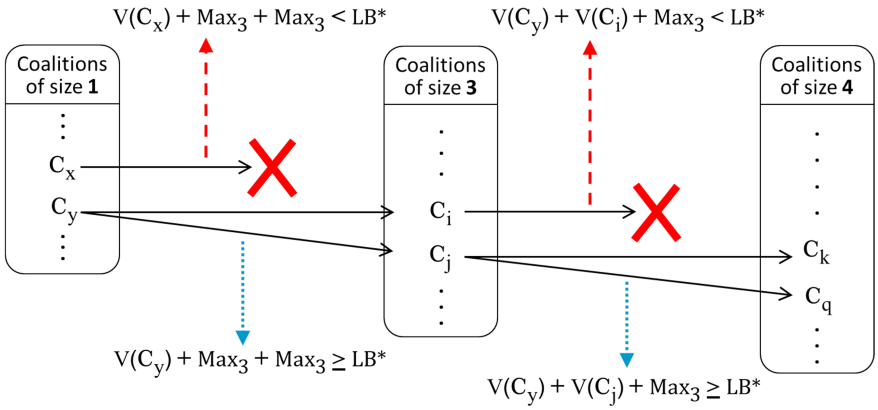

Figure 2: An illustration of IP's branch-and-bound technique when searching $\Pi^A_{\{1,3,4\}}$. Here, the algorithm recognizes that the coalition structures containing C_x or $\{C_y, C_i\}$ cannot be optimal, and so IP does not proceed deeper into the search tree.

repeats this process so that, eventually, every coalition structure in Π^A_I is examined. Here, it should be noted that a straight forward repetition of the aforementioned process would not be efficient, because some of the coalition structures will be examined multiple times. For instance, every coalition structure $\{C_1, C_2, C_3\} \in \Pi^A_{\{2,2,3\}}$ will be examined twice, once as $\{C_1, C_2, C_3\}$ and once as $\{C_2, C_1, C_3\}$ (because in this example we have $|C1| = |C_2|$). For more details on how IP avoids such redundant operations, see (Rahwan et al., 2009).

To speed up the search, IP applies a *branch-and-bound* technique at every depth $d < |I|$. Specifically, after fixing d coalitions, $C_1 \in \mathcal{C}^A_{i_1}, \ldots, C_d \in \mathcal{C}^A_{i_d}$, and before iterating over the relevant coalitions in $\mathcal{C}^A_{i_{d+1}}, \ldots, \mathcal{C}^A_{i_{|I|}}$, it checks whether the following inequality holds:

$$\sum_{j=1}^{d} v(C_j) + \sum_{j=d+1}^{|I|} Max_{i_j} < V(CS^{**}) \tag{1}$$

Now if the inequality in (1) holds, it means that every coalition structure containing $C_1, \ldots, C_d$ can be skipped during the search, because its value cannot be greater than $V(CS^{**})$—the value of the best coalition structure found by the algorithm so far. Figure 2 provides an illustration of how IP searches $\Pi^A_{\{1,3,4\}}$ given 8 agents.

As mentioned earlier, before IP can use the branch-and-bound technique, it needs to first compute Max_i and Avg_i for all $i \in \{1, \ldots, n\}$. To do so, the algorithm needs to iterate over all the coalition values (to compute the average and maximum values of every coalition size). One way to perform this iteration is to first go through all coalitions of size 1 (to compute Max_1 and Avg_1), then through all those of size 2 (to compute Max_2 and Avg_2), then size 3 and so on. However, to allow for certain subspaces to be searched during the iteration process, IP goes through the coalitions in a different order. More specifically, it iterates over all coalitions of size $s \in \{1, \ldots, \lfloor n/2 \rfloor\}$ in a *lexicographic* order, and *simultaneously* iterates over all coalitions of size $n - s$ in an *anti-lexicographic* order.[3] With this order, the i^{th} coalition of size s, together with the i^{th} coalition of size $n - s$, form a coalition structure in $\Pi^A_{\{s,n-s\}}$. By going through every such pair, IP examines every coalition structure in $\Pi^A_{\{s,n-s\}}$. By the end of this iteration process, every

[3]Such iteration can be carried out efficiently, e.g., using the techniques in (Rahwan and Jennings, 2007).

subspace, $\Pi^A_I : |I| = 2$, would have been searched.

The IP algorithm runs in $O(n^n)$ time, and could in the worst case end up constructing every possible coalition structure. In practice, however, IP has been shown to run significantly faster than DP given popular coalition-value distributions. Furthermore, the bound that IP generates, i.e., $\beta = UB^*/V(CS^{**})$, has been shown to improve rapidly during the search process, e.g., reaching 90% by searching less than 10^{-9} of the search space for 25 agents (given certain value distributions).

3.2. The DP Algorithm

The DP algorithm is based on the following theorem.

Theorem 2. *Given a coalition $C \subseteq A$, the value of an optimal partition of C, denoted as $f(C)$, can be computed recursively as follows:*

$$f(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\left\{v(C)\ ,\ \max_{\{C',C''\}\in\Pi^C}\left(f(C') + f(C'')\right)\right\} & \text{otherwise.} \end{cases} \tag{2}$$

The pseudo code of DP is given in Algorithm 1. Basically, for every coalition, $C \subseteq A$, the algorithm computes $f(C)$ as well as $t(C)$—a variable that provides an indication of the optimal partition of C. In more detail, DP starts with the coalitions of size 1. For every such coalition, C, it sets $f(C) = v(C)$ and $t(C) = \{C\}$ to indicate that the optimal partition of C is $\{C\}$ (see lines 1 to 3 in the pseudo code). After that, it iterates over all the coalitions of size $s = 2, \ldots, n$ (see lines 4 and 5). For every such coalition, C, it uses two temporary variables, namely ϑ^* and C^*; these variables are computed in lines 6 to 11 such that:

$$\vartheta^* = \max_{\{C',C''\}\in\Pi^C} f(C') + f(C'') \tag{3}$$

$$\{C^*, C\backslash C^*\} = \arg\max_{\{C',C''\}\in\Pi^C} f(C') + f(C'') \tag{4}$$

Now since $|C| \neq 1$, then according to equation (2) we have $f(C) = \max\{v(C), \vartheta^*\}$. Based on this, DP checks in line 12 whether $v(C) > \vartheta^*$, and then sets $f(C)$ accordingly. In more detail:

- if $v(C) > \vartheta^*$, then DP sets $f(C) = v(C)$. This implies that $\{C\}$ is an optimal partition of C (because $f(C)$ is, by definition, the value of the optimal partition of C). Based on this, DP sets $t(C) = \{C\}$ so that it can later on remember that it is not beneficial to split C into smaller coalitions (see line 14).

- Otherwise, DP sets $f(C) = \vartheta^*$. This basically means that $f(C) = f(C^*) + f(C\backslash C^*)$ (because we know from equations (3) and (4) that $\vartheta^* = f(C^*) + f(C\backslash C^*)$). This, in turn, means that an optimal partition of C can be obtained by first splitting C into two coalitions, C^* and $C\backslash C^*$, and then replacing each one of those two coalitions with its optimal partition. Based on this, DP sets $t(C) = \{C^*, C\backslash C^*\}$ to remember the best way of splitting C into two coalitions (see line 17).

Once $f(C)$ and $t(C)$ are computed for every $C \subseteq A$, an optimal coalition structure CS^* can be computed recursively (lines 18 to 22). In more detail, DP initially sets $CS^* = \{A\}$, and then repeats the process of replacing some coalition $C \in CS^*$ with $t(C)$ as long as it is beneficial to do so (i.e., as long as $\{C\}$ is not an optimal partition of C). A four-agent example is illustrated in Figure 3. Here, DP first computes $t(C)$ and $f(C)$ for every $C : |C| = 1, \ldots, 4$. After that, it determines the optimal partition of $\{a_1, a_2, a_3, a_4\}$ as follows. It first checks $t(\{a_1, a_2, a_3, a_4\})$ and

Algorithm 1: The DP algorithm.

Input: $v(C)$ for all $C \subseteq A$.
Output: the optimal coalition structure, CS^*.

```
1  foreach C ⊆ A : |C| = 1 do // for every coalition of size 1
2      f(C) ← v(C)
3      t(C) ← {C}
4  foreach s = 2 to n do
5      foreach C ⊆ A : |C| = s do // for every coalition of size s
6          ϑ* ← −∞ // initialization
7          foreach {C', C''} ∈ Π^C do // for every possible way of splitting C in two
8              ϑ' ← f(C') + f(C'')
9              if ϑ* < ϑ' then
10                 ϑ* ← ϑ' // to ensure that ϑ* = max_{{C',C''}∈Π^C} (f(C') + f(C''))
11                 C* ← C' // to ensure {C*, C\C*} = arg max_{{C',C''}∈Π^C} (f(C') + f(C''))
12         if v(C) > ϑ* then // i.e., if v(C) > max_{{C',C''}∈Π^C} (f(C') + f(C''))
13             f(C) ← v(C)
14             t(C) ← {C} // to remember that it is not beneficial to split C
15         else
16             f(C) ← ϑ*
17             t(C) ← {C*, C\C*} // to remember the best way to split C in two
   // Having computed t(C) and f(C) for every C ⊆ A, the remaining lines compute CS*
18 CS* ← {A}
19 foreach C ∈ CS* do
20     if t(C) ≠ {C} then // i.e., if {C} is not an optimal partition of C
21         CS* ← (CS*\{C}) ∪ t(C) // replace C with the two coalitions in t(C)
22         Go to line 19 and start with the new CS*.
23 return CS*
```

realizes that it is more beneficial to split $\{a_1, a_2, a_3, a_4\}$ into $\{a_1, a_2\}$ and $\{a_3, a_4\}$. After that, it checks $t(\{a_1, a_2\})$ and realizes that it is more beneficial to split $\{a_1, a_2\}$ into $\{a_1\}$ and $\{a_2\}$ (see the dashed arrow in Figure 3). Finally, it checks $t(\{a_3, a_4\})$ and realizes that it is more beneficial to keep $\{a_3, a_4\}$ as it is (see the dotted arrow in the figure). The optimal coalition structure is, therefore, $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$.

DP requires storing a total of 2^{n+1} values, namely $f(C)$ and $t(C)$ for every $C \subseteq A$. The running time of DP has been shown to be $O(3^n)$ (Yeh, 1986). This is significantly less than $\omega(n^{n/2})$—the time required to exhaustively enumerate all coalition structures. However, the disadvantage is that DP provides no interim solution before completion, meaning that it is not possible to trade computation time for solution quality.

4. Improving the DP Algorithm

In this section, we present the first contribution of this article, which is to develop an optimal version of DP. More specifically, in Section 4.1, we demonstrate that there exists a strong link between the way DP works and the way nodes are connected in a certain graph. Based on this link, we analyse in Section 4.2 the effect of avoiding certain operations of DP. Building upon

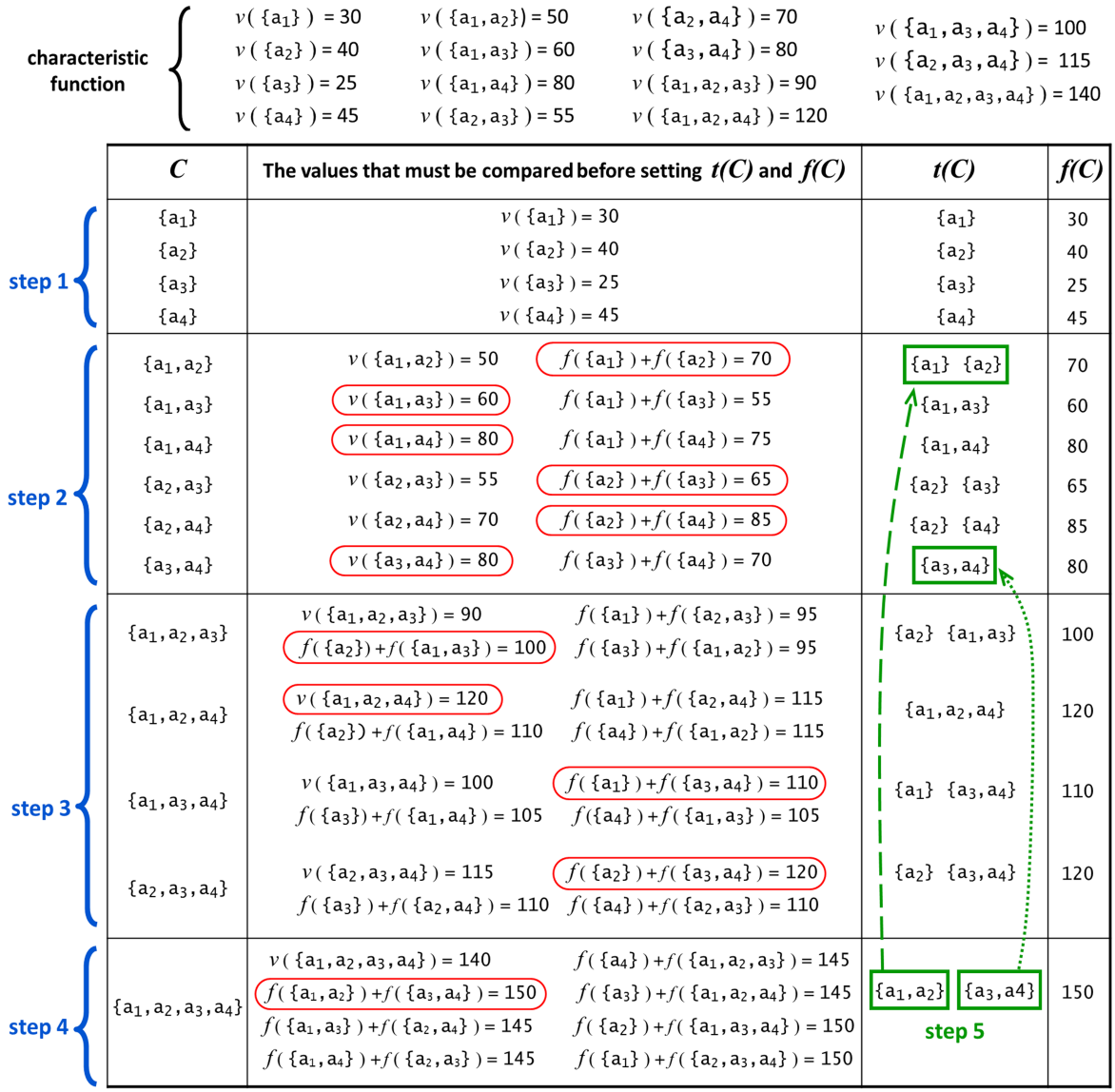

Figure 3: A four-agent example of how DP computes $t(C)$ and $f(C)$ for every $C \subseteq A$.

this analysis, we present in Section 4.3 our optimal dynamic programming (ODP) algorithm—a modified version of DP that avoids all the redundant operations and memory requirements of DP, without losing the guarantee of finding an optimal solution.

4.1. The Link Between DP and the Coalition Structure Graph

To obtain a deeper understanding of how DP works, we looked at the *coalition structure graph* (Sandholm et al., 1999). Specifically, in this undirected graph, every node represents a coalition structure. These nodes are categorized into n levels, namely $\Pi_1^A, \ldots, \Pi_n^A$, such that level Π_i^A contains the nodes that represent all coalition structures containing exactly i coalitions each. An edge connects two coalition structures if and only if: (1) they belong to two consecutive levels Π_i^A and Π_{i-1}^A, and (2)

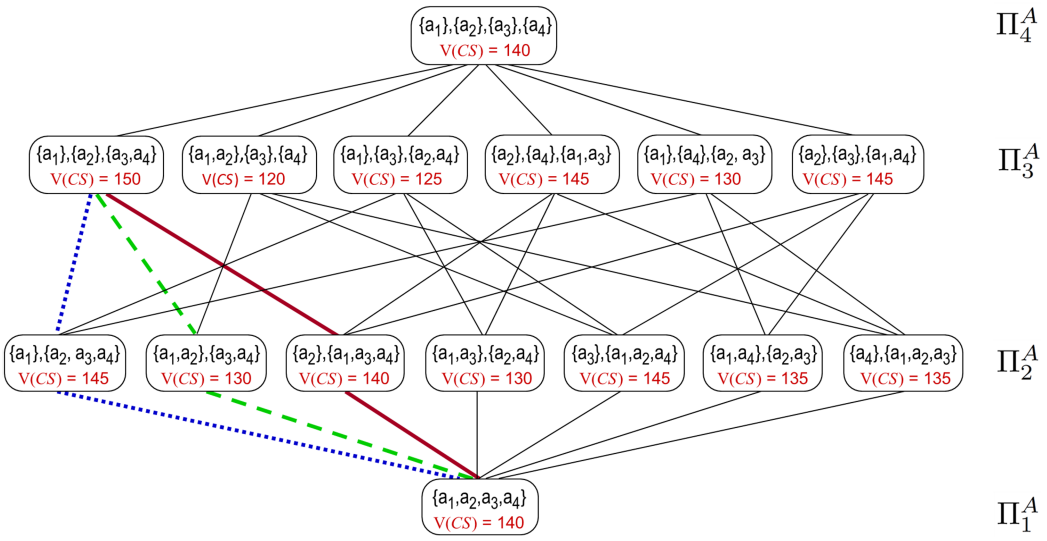

Figure 4: The coalition structure graph of four agents. The figure also shows the value of every coalition structure based on the characteristic function from Figure 3.

the coalition structure in Π_i^A can be obtained from the one in Π_{i-1}^A by splitting one coalition into two. Figure 4 shows a four-agent example of the coalition structure graph. It also shows every coalition structure's value based on the characteristic function from Figure 3.

Looking at this graph enables us to visualize how DP works. To this end, observe that every *movement* upwards in the graph (between adjacent nodes) corresponds to the splitting of one coalition into two (see Figure 4). Based on this observation, the work of DP can be divided into three main tasks that can all be seen on the graph:

1. **Task 1: evaluate all the movements in the graph**: For every coalition $C : |C| \geq 2$, the algorithm *evaluates every partition* $\{C', C''\} \in \Pi^C$, and that is by computing $f(C') + f(C'')$ (see lines 7 and 8 of the pseudo code). This can be interpreted as *evaluating every movement* that involves splitting C in two. Since the algorithm does this for every possible coalition of size $s \geq 2$, all the movements in the graph are eventually evaluated.

2. **Task 2: store the most beneficial movements**: What DP actually does in lines 12 to 17 is determine, for every coalition C, whether it is beneficial to make a movement that involves splitting C and, if so, what is the best such movement (this decision is stored in $t(C)$). To see how this is the case, let us analyse how DP sets $t(C)$. From the pseudo code, we can see that DP sets $t(C)$ to either $\{C\}$ or $\{C^*, C \backslash C^*\}$. As mentioned earlier in section 3.2, setting $t(C) = \{C\}$ only happens when $\{C\}$ is an optimal partition of C. On the other hand, setting $t(C) = \{C^*, C \backslash C^*\}$ only happens when an optimal partition of C can be obtained by first splitting it into C^* and $C \backslash C^*$, and then replacing each one of those two coalitions with its optimal partition. Let us interpret this differently based on the coalition structure graph. Setting $t(C) = \{C\}$ means that, from any node representing a coalition structure $CS \ni C$, it is not beneficial to *make a movement* that involves splitting C. On the other hand, setting $t(C) = \{C^*, C \backslash C^*\}$ means that, from any node representing $CS \ni C$, *the most beneficial movement*—out of all those that involve splitting C—is the one in which C is split into C^* and $C \backslash C^*$.

3. **Task 3: move upwards in the graph**: This occurs in lines 18 to 22. Here, DP first initializes CS^* by setting $CS^* = \{A\}$. This means that DP is actually starting at the node that represents $\{A\}$, i.e., the bottom node in the graph. After that, DP selects some coalition $C \in CS^* : t(C) \neq \{C\}$ (if such a coalition exists), and replaces it with $t(C)$. By doing this, DP is actually making a movement that involves splitting C into the two coalitions that are stored in $t(C)$. This process is repeated until $t(C) = \{C\}$ for all $C \in CS$. In other words, DP keeps moving upwards in the graph through a series of connected nodes—a *"path"*—until it reaches a node after which no movement is beneficial. For instance, in our example from Figure 3, the way DP reached $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ can be visualized as movements through the dashed path in Figure 4, where the first movement involved splitting $\{a_1, a_2, a_3, a_4\}$ into $\{a_1, a_2\}$ and $\{a_3, a_4\}$, and the second movement involved splitting $\{a_1, a_2\}$ into $\{a_1\}$ and $\{a_2\}$.

It should be noted that DP has the ability to look ahead before deciding on its next movement. In Figure 4, for example, DP did not move from the node representing $\{a_1, a_2, a_3, a_4\}$ to the one representing $\{\{a_3\}, \{a_1, a_2, a_4\}\}$ (which has a value of 145). Instead, it moved through the dashed edge to the node that represents $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ (which has a value of 130). The way DP made that decision was based on the movements that will follow. As can be seen from the figure, the movement made by DP leads to the optimal coalition structure (which has a value of 150), while the other movement does not lead to any coalition structure with a value greater than 145. This ability to look ahead is a result of the way DP evaluates the movements. More specifically, given a movement in which a coalition is split into C' and C'', the evaluation that DP assigns to this movement is equal to $f(C') + f(C'')$ (see line 8 of the pseudo code). In other words, the evaluation is based on the values of *the optimal partitions* of C' and C''.

From this visualization it is clear that, for any coalition structure $CS : |CS| > 2$, there are multiple paths that start from the bottom node of the graph, and end with the node that contains CS. In Figure 4, for example, one could reach $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ through three different paths, which are highlighted using dotted, dashed, and bold edges respectively. This raises the following question: *"whenever there are multiple paths that lead to the same optimal node, can DP reach this node through any of those paths?"* We will show that the answer is *"yes"*. To this end, observe that whenever there are multiple paths that lead to an optimal node, DP has no preference on which path to take. This can be seen in line 17 of the pseudo code, where $t(C)$ is set to $\{C^*, C \backslash C^*\}$, which is basically equal to $\arg\max_{\{C', C''\} \in \Pi^C} \big(f(C') + f(C'')\big)$. This implies that whenever there are multiple arguments $\{C', C''\} \in \Pi^C$ that maximize $f(C') + f(C'')$, DP has no preference on which argument to store in $t(C)$. In Figure 3, for example, $t(\{a_1, a_2, a_3, a_4\})$ was set to $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ because it had *one of* the highest evaluations, which is 150 (see how $f(\{\{a_1, a_2\}) + f(\{a_3, a_4\}) = 150$ in Figure 3). However, $t(\{a_1, a_2, a_3, a_4\})$ could have been set to $\{\{a_1\}, \{a_2, a_3, a_4\}\}$ instead (since it also has an evaluation of 150). If that happened, then DP would have found, based on $t(\{a_2, a_3, a_4\})$, that it is more beneficial to split $\{a_2, a_3, a_4\}$ into $\{a_2\}$ and $\{a_3, a_4\}$. As a result, the same optimal solution (i.e., $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$) would have been found, but through the dotted path rather than the dashed one in Figure 4.

4.2. Analysing the Effect of Avoiding Certain Operations in DP

In the previous subsection, we showed that DP evaluates all the movements in the coalition structure graph, stores the best ones in the table t, and then selects from t the movements that together form a path from the bottom node to an optimal node. We also showed that DP has no preference on any alternative paths that lead to the same optimal node. All of these observations raise yet another important question: *"what happens if DP is modified such that it only evaluates some of the movements in the graph?"* Suppose that for a certain coalition, C, the algorithm did not evaluate a particular movement that involves splitting C into two coalitions, namely C_1 and C_2.

More formally, suppose that the term Π^C in line 7 of the pseudo code was replaced with the term $\Pi^C \backslash \{C_1, C_2\}$. In this case, the movement stored in $t(C)$ would be the best out of all the movements that DP has evaluated (i.e., excluding the one in which C is split into C_1 and C_2). As a result, whenever a coalition structure $CS \ni C$ is reached, the movement to $CS' = (CS \backslash \{C\}) \cup \{C_1, C_2\}$ would no longer be an option (since DP always selects its movements from the table t). In other words, DP would ignore the existence of the edge that connects CS to CS', and would evaluate the movements through the remaining edges, and decide on its path accordingly. This can be visualized on the graph by *removing the edge* that connects CS to CS'. Now if CS' happened to be the only optimal solution in the graph, and if the removed edge happened to be the only one leading to CS', then DP would no longer be able to find the optimal solution. We formalize this observation in the remainder of this subsection.

For any two *disjoint* coalitions, C_1 and C_2, let m^{C_1,C_2} denote the movement that corresponds to splitting $C = C_1 \cup C_2$ into C_1 and C_2.[4] Moreover, let $\mathcal{M}$ denote the set of all possible movements in the coalition structure graph, i.e., $\mathcal{M} = \{m^{C_1,C_2} : C_1, C_2 \subseteq A, C_1 \cap C_2 = \emptyset\}$. Now, given an arbitrary subset of movements, $M \subseteq \mathcal{M}$, and given two arbitrary partitions, $\pi, \pi' \in \Pi$, we write $\pi \xrightarrow{M} \pi'$ if and only if the partition π' can be reached from π via *a single movement* in M. Formally:

$$\pi \xrightarrow{M} \pi' \quad \Leftrightarrow \quad \exists m^{C_1,C_2} \in M : \pi' = \pi \backslash \{C_1 \cup C_2\} \cup \{C_1, C_2\}$$

While $\xrightarrow{M}$ expresses the notion of reachability with respect to single movements from M, the following definition generalizes this notion to multiple movements.

Definition 4. *Given an arbitrary subset of movements, $M \subseteq \mathcal{M}$, and given two arbitrary partitions, $\pi, \pi' \in \Pi$, we say that π'* is **reachable** *from π via M, and write: $\pi \overset{M}{\rightsquigarrow} \pi'$, if and only if π' is either equal to π, or can be reached from π via one or more movements in M. More formally:*

$$\pi \overset{M}{\rightsquigarrow} \pi' \quad \Leftrightarrow \quad \big(\pi = \pi' \big) \vee \big(\pi \xrightarrow{M} \pi' \big) \vee \big(\exists \{\pi_1, \ldots, \pi_k\} \subseteq \Pi : \pi \xrightarrow{M} \pi_1 \xrightarrow{M} \ldots \xrightarrow{M} \pi_k \xrightarrow{M} \pi' \big)$$

Let us denote by R^π_M the set of *all partitions that are reachable from π via M*. More formally, $R^\pi_M = \{\pi' \in \Pi : \pi \overset{M}{\rightsquigarrow} \pi'\}$. Observe that every partition in R^π_M is either equal to π, or reachable from π via *at least one movement* in M, in which case it must also be reachable from at least one of the partitions in $\{\pi' \in \Pi : \pi \xrightarrow{M} \pi'\}$. Based on this observation, the set R^π_M can be computed recursively as follows:

$$R^\pi_M \;=\; \{\pi\} \;\cup \bigcup_{\pi' \in \Pi : \pi \xrightarrow{M} \pi'} R^{\pi'}_M \tag{5}$$

Theorem 3. *Given an arbitrary partition, $\{C_1, \ldots, C_k\} \in \Pi$, and an arbitrary subset of movements, $M \subseteq \mathcal{M}$, the following holds:*

$$R^{\{C_1,\ldots,C_k\}}_M \;=\; R^{\{C_1\}}_M \;\times\; \ldots \;\times\; R^{\{C_k\}}_M$$

Proof. See Appendix B.

[4] Observe that the same movement, m^{C_1,C_2}, can be made through different edges in the coalition structure graph. More precisely, it can be made through any edge that connects a coalition structure, $CS \ni C$, to another coalition structure, $CS' = (CS \backslash \{C\}) \cup \{C_1, C_2\}$.

Now, let us define $f_M(C)$ as the value of an optimal partition in $R_M^{\{C\}}$. More formally, $f_M(C) = \max_{\pi \in R_M^{\{C\}}} V(\pi)$. With this definition, we are ready to generalize Theorem 2 (the main theorem behind DP), and that is by replacing $f(C)$ and Π^C with $f_M(C)$ and $R_M^{\{C\}}$, respectively.

Theorem 4. *For any coalition $C \subseteq A$, and for any subset of movements, $M \subseteq \mathcal{M}$, the following holds:*

$$f_M(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\Big\{v(C)\ ,\ \ \max_{\{C',C''\} \in R_M^{\{C\}}} \big(f_M(C') + f_M(C'')\big)\Big\} & \text{otherwise.} \end{cases} \tag{6}$$

Proof. See Appendix C.

Now, we can analyse the effect of replacing every $f(C)$ and Π^C in DP with $f_M(C)$ and $R_M^{\{C\}}$, respectively. Let us call the resulting algorithm DP_M. Based on Theorem 4, it is easy to see how DP_M will compute $f_M(C)$ recursively for every $C \subseteq A$ (in the same way that DP computes $f(C)$ recursively based on Theorem 2). By the end of this process, DP_M would have computed $f_M(A)$—the value of the best coalition structure reachable from $\{A\}$ via M. To identify this coalition structure, DP_M uses the table t in the same way that DP uses it. This leads to the following corollary:

Corollary 1. *Let $M \subseteq \mathcal{M}$ be an arbitrary subset of movements, and let DP_M be the version of DP in which every $f(C)$ is replaced with $f_M(C)$, and every Π^C is replaced with $R_M^{\{C\}}$. Then, the output of DP_M is a coalition structure in:* $\arg\max_{CS \in R_M^{\{A\}}} V(CS)$.

This corollary can be useful in settings where only certain movements are of interest. For example, if a coalition is allowed to form only when its size exceeds a certain threshold, s, then instead of using DP, one can use DP_M where $M = \{m^{C_1,C_2} \in \mathcal{M} : |C_1| > s, |C_2| > s\}$. This is relevant to settings like `www.groupon.com`, where *"deals"* are activated only when the number of people signing up for that deal exceeds a certain threshold.

Having analysed the effect of avoiding the evaluation of certain movements in the coalition structure graph, the following subsection shows how this analysis can help design an optimal version of DP.

4.3. The ODP Algorithm

In this subsection, we present our *optimal dynamic programming (ODP)* algorithm—a modified version of DP that avoids all the redundant operations and memory requirements of DP, while maintaining the guarantee of finding an optimal coalition structure.

First, let us show how the redundant operations are avoided. As mentioned earlier in Corollary 1, for any given $M \subseteq \mathcal{M}$, the only coalition structures that are searched by DP_M are those in $R_M^{\{A\}}$, i.e., those that are reachable from $\{A\}$ via M. Thus, DP_M is guaranteed to find an optimal coalition structure if and only if $R_M^{\{A\}} = \Pi^A$. Based on this observation, we now present the main theorems behind ODP.

Theorem 5. *For any two coalitions $C', C'' \in \mathcal{C}^A$, let us write $C' < C''$ if and only if C' precedes C'' lexicographically. Now, if we define $M^* \subseteq \mathcal{M}$ as:*

$$M^* = \Big\{m^{C',C''} \in \mathcal{M} : \big(C' \cup C'' = A\big) \wedge \big(C' < C'' < A \setminus (C' \cup C'')\big)\Big\} \tag{7}$$

Then, the following holds:

$$R_{M^*}^{\{A\}} = \Pi^A \tag{8}$$

Proof. See Appendix D.

Based on Theorem 5, our algorithm, ODP, *is the version of DP that only evaluates the movements in M^*, i.e., it is the one that uses f_{M^*} instead of f.* Formally, $ODP = DP_{M^*}$.

Theorem 6. *For any coalition $C \in \mathcal{C}^A$, if $\{a_1, a_2\} \not\subseteq C$, then ODP does not evaluate any of the possible ways of splitting C.*

Proof. See Appendix E.

Theorem 7. *It is not possible to evaluate fewer splits compared to ODP, and still be guaranteed to find an optimal coalition structure.*

Proof. See Appendix F.

Theorem 8. *The number of movements in M^* is equal to the number of coalition structures in $\Pi_2^A \cup \Pi_3^A$—levels 2 and 3 of the coalition structure graph. That is:*

$$\left|M^*\right| \quad = \quad \left|\Pi_2^A\right| + \left|\Pi_3^A\right|$$

Proof. See Appendix G.

Theorem 9. *Given n agents, ODP runs in $O(3^n)$ time.*

Proof. See Appendix H.

Having explained how ODP avoids all redundant operations of DP, we now show how it uses a smaller amount of memory compared to DP. In particular, we will show how ODP avoids maintaining the table t all together.

To this end, recall that for any coalition $C \subseteq A : |C| \geq 2$, DP computes $t(C)$ is done as follows, where $\{C^*, C \backslash C^*\} = \arg\max_{\{C', C''\} \in \Pi^C} f(C') + f(C'')$:

$$t(C) = \begin{cases} \{C\} & \text{if } v(C) \ > \ f(C^*) + f(C \backslash C^*) \\ \{C^*, C \backslash C^*\} & \text{otherwise.} \end{cases}$$

In other words, to compute $t(C)$, one needs to first compute $\{C^*, C \backslash C^*\}$, and then compare $f(C^*) + f(C \backslash C^*)$ with $v(C)$. However, those same operations are also needed in order to compute $f(C)$. Therefore, to avoid repeating those operations twice, one must assign $t(C)$ and $f(C)$ in the same loop (see lines 5 to 17 of the pseudo code of DP). While this approach is computationally efficient, it requires an exponential amount of memory (because it involves storing $t(C)$ for every $C \subseteq A$). In order to avoid this drawback, we need to further analyse the way DP uses the table t.

To this end, observe that t is only used in the final stage of DP (lines 18 to 22 of the pseudo code). In this stage, DP initially sets $CS^* = \{A\}$, and then repeatedly splits some coalition $C \in CS^*$ in two as long as it is beneficial to do so. Here, DP needs to know $t(A)$ in order to

determine whether it is beneficial to split A and, if so, determine how to split A in two in the most beneficial way. Similarly, every time DP splits a coalition $C \in CS^*$ into two coalitions, C' and C'', it needs to know $t(C')$ and $t(C'')$ to determine whether, and how, to split each one of those two coalitions. Based on this, it is easy to show that, out of the $2^n - 1$ entries in the table t, DP uses at most $2n - 3$ entries.

Based on the above analysis, in ODP, we avoid storing all $2^n - 1$ entries of t and instead compute the required $2n - 3$ entries. This results in a significant reduction in memory requirement at the expense of an insignificant addition in computation.

Algorithm 2 provides the pseudo code of ODP, while Algorithm 3 provides the pseudo code of the function `getBestPartition`—a function used in line 32 of Algorithm 2.

5. The ODP-IP Algorithm

The previous section presented the first contribution of this article—the development of ODP. This section presents the second contribution—the development of ODP-IP. Here, the basic idea is to modify ODP and IP such that they can assist one another when running in parallel (i.e., when they are simultaneously used to solve the same problem instant).

This section is structured as follows. Section 5.1 presents the link between IP and DP. Section 5.2 shows how to modify ODP so that it searches subspaces of the integer partition graph. Section 5.3 shows how to use the information provided by ODP to speed up IP's depth-first search. Section 5.4 show how to modify IP such that it searches multiple subspaces simultaneously. Finally, Section 5.5 provides a summary of ODP-IP, and analyses its complexity.

5.1. The Link Between DP and IP

Given the differences between ODP and IP, both in terms of the search-space representation as well as the search techniques that are being used, it is not trivial to determine how these two algorithms can be combined. First, let us draw a link between IP and DP (not ODP). In order to draw this link, we developed what we call the *integer partition graph*. This is an undirected graph where every node represents an integer partition, and two nodes representing $I, I' \in \mathcal{I}^n$ are connected via an edge if and only if there exists two parts $i, j \in I$ such that $I' = (I \setminus \{i, j\}) \uplus \{i + j\}$ (here $\uplus$ denotes the multiset union operation). A four-agent example is shown in Figure 5(A).

By looking at this graph, we can visualize the way DP searches the subspaces that are represented by different integer partitions. To this end, recall that the operation of DP can be visualized as the evaluation of the possible movements in the *coalition structure graph*. Furthermore, avoiding the evaluation of some of these movements can be visualized by *removing the edges* through which these movements are made. Importantly, these same operations can also be visualized on the *integer partition graph*. Specifically:

- By making a movement from one coalition structure CS' to another CS'' (in the coalition structure graph), DP is actually making a movement (in the integer partition graph) from one integer partition $I' : \Pi^A_{I'} \ni CS'$ to another $I'' : \Pi^A_{I''} \ni CS''$. For example, see how the movement from $\{\{a_1\}, \{a_2, a_3, a_4\}\}$ to $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ in Figure 5(B) corresponds to the movement from $\Pi^A_{\{1,3\}}$ to $\Pi^A_{\{1,1,2\}}$ in Figure 5(A).

- If we remove all the edges (in the coalition structure graph) that correspond to splitting a coalition of size s into two coalitions of sizes s' and s'', then we are essentially removing every edge (in the integer partition graph) that connects an integer partition, $I : I \ni s$, to another, $I' = (I \setminus \{s\}) \uplus \{s', s''\}$. For instance, removing the dotted edges in Figure 5(B) corresponds to the removal of the dotted edge that connects $\Pi^A_{\{2,2\}}$ to $\Pi^A_{\{2,1,1\}}$ in Figure 5(A). This is

Algorithm 2: The Optimal Dynamic Programming algorithm (ODP).

Input: $v(C)$ for all $C \subseteq A$.
Output: the optimal coalition structure, CS^*.

```
// First, compute f_{M*}(C) for every singleton coalition
foreach C ⊆ A : |C| = 1 do
    f_{M*}(C) ← v(C)
// Second, compute f_{M*}(C) for every coalition of size 2
foreach C ⊆ A : |C| = 2 do
    f_{M*}(C) ← v(C)
if f_{M*}({a1, a2}) < v({a1}) + v({a2}) then
    f_{M*}({a1, a2}) ← v({a1}) + v({a2})
// Third, compute f_{M*}(C) for every coalition of size s = 3, ..., n − 1
foreach s = 3 to n − 1 do
    foreach S ⊆ A \ {a1, a2} : |S| = s − 2 do  // S is our desired coalition before adding {a1, a2}
        C ← S ∪ {a1, a2} // Observe that |C| = s
        f_{M*}(C) ← v(C)
        foreach {S', S''} ∈ (Π^S ∪ {∅, S}) do
            j ← min_{a_i ∈ A\C} i   // a_j is the ''smallest'' agent outside C
            A^j ← {a3, ..., a_{j−1}}
            if f_{M*} < v(S' ∪ {a1}) + v(S'' ∪ {a2}) then
                f_{M*} ← v(S' ∪ {a1}) + v(S'' ∪ {a2})
            if f_{M*} < v(S' ∪ {a2}) + v(S'' ∪ {a1}) then
                f_{M*} ← v(S' ∪ {a2}) + v(S'' ∪ {a1})
            if S' ∩ A^j ≠ ∅ then  // to ensure that S'' ∪ {a1, a2} < S' < A \ C
                if f_{M*} < v(S') + v(S'' ∪ {a1, a2}) then
                    f_{M*} ← v(S') + v(S'' ∪ {a1, a2})
            if S'' ∩ A^j ≠ ∅ then  // to ensure that S' ∪ {a1, a2} < S'' < A \ C
                if f_{M*} < v(S' ∪ {a1, a2}) + v(S'') then
                    f_{M*} ← v(S' ∪ {a1, a2}) + v(S'')
    foreach C ⊆ A : |C| = s, {a1, a2} ⊈ C do
        f_{M*}(C) ← v(C)
// Fourth, compute f_{M*(A)} and t(A)
f_{M*}(A) ← v(A)
t(A) ← {A}
foreach {C', C''} ∈ Π_2^A do
    if f_{M*}(A) < f_{M*}(C') + f_{M*}(C'') then
        f_{M*}(A) ← f_{M*}(C') + f_{M*}(C'')
        t(A) ← {C', C''}
// Finally, set CS* to be an optimal split of A
CS* ← getBestPartition(A, t(A)) // see the pseudo code in Algorithm 3
return CS*
```

Algorithm 3: `getBestPartition`—a function used in ODP.

Input: C and $t(C)$, where $C \subseteq A$. It is assumed that the table f_{M^*} has been computed.

Output: the best partition of C that is reachable via M^*, i.e., the best partition in $R^{\{C\}}_{M^*}$.

```
// Check whether {C} is an optimal partition of C
if t(C) = {C} then
    return {C}
else
    // In this case, there are two coalitions in t(C), let us denote them as C_1 and C_2
    π ← ∅ // initialization
    foreach C_i ∈ t(C) do  // for each one of the two coalitions in t(C)
        // First, compute t(C_i) (in lines 6 to 12)
        if f_{M*}(C_i) = v(C_i) then  // i.e., if {C_i} is an optimal partition of C_i
            t(C_i) ← {C_i}
        else
            // Identify two coalitions {C'_i, C''_i} ∈ Π^{C_i} such that f_{M*}(C'_i) + f_{M*}(C''_i) = f_{M*}(C_i)
            foreach {C'_i, C''_i} ∈ Π^{C_i} ∪ ({∅, C_i}) do
                if f_{M*}(C'_i) + f_{M*}(C''_i) = f_{M*}(C_i) then
                    t(C_i) ← {C'_i, C''_i}
                    Break
        // Having computed t(C_i), we now use it to compute an optimal partition of C_i, and then add that
        partition to π
        π ← π ∪ getBestPartition(C_i, t(C_i))
    // Now, π is the union of the optimal partitions of C_1 and C_2
    return π
```

because it is no longer possible to move from a coalition structure in $\Pi^A_{\{2,2\}}$ to another in $\Pi^A_{\{2,1,1\}}$.

This visualization provides the link between DP and IP since the latter deals with subspaces that are represented by integer partitions.

5.2. Searching Subspaces Using ODP

In the previous section, we showed that avoiding the evaluation of *all* possible movements from *all* coalitions of a particular size, s, into two coalitions of particular sizes, s' and s'', corresponds to removing edges from the integer partition graph—the graph that links DP and IP. The problem with ODP is that it avoids the evaluation of *only some* of the movements from coalitions of a given size. For instance, given $n = 4$ and $s = 2$, ODP avoids evaluating the movements from $\{a_1, a_3\}$, $\{a_1, a_4\}$, $\{a_2, a_3\}$, $\{a_2, a_4\}$ and $\{a_3, a_4\}$, but evaluates the movement from $\{a_1, a_2\}$. Because of this single movement from a coalition of size 2, we cannot remove the dotted edge from Figure 5(A). To circumvent this, we will now present a *size-based version* of ODP that, for any three sizes, $s, s', s'' \in \{1, \ldots, n\} : s = s' + s''$, either evaluates *all* or *none* of the movements in which a coalition of size s is split into coalitions of sizes s' and s''. While, with this restriction, not all redundant evaluations can be avoided, we show later on in Section 6 that most of the redundant evaluations are actually avoided.

Theorem 10. *For any two positive integers, $s', s'' \in \mathbb{Z}^+$, let us define $M^{s',s''} \subseteq \mathcal{M}$ as the set that consists of every movement in which a coalition of size $(s' + s'')$ is split into two coalitions of sizes*

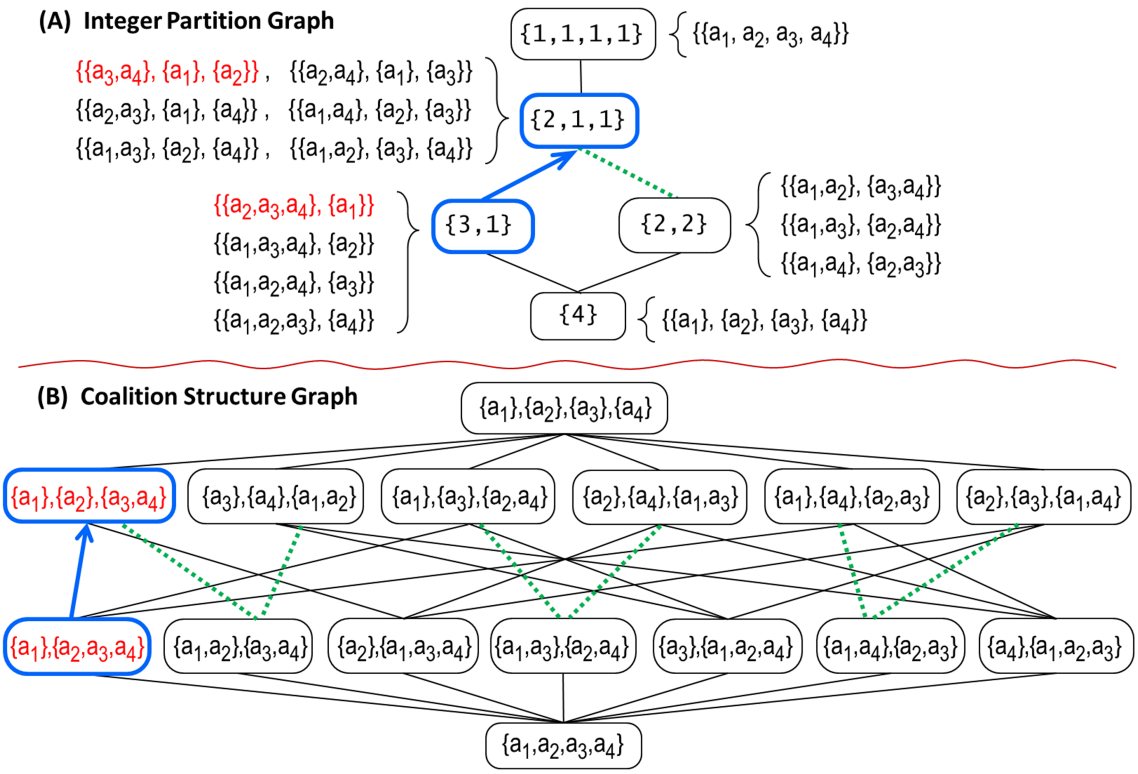

Figure 5: Given 4 agents, (A) shows the integer partition graph while (B) shows the coalition structure graph. The Figure also shows how a movement in (B) corresponds to a movement in (A), and the removal of the dotted edges in (B) corresponds to the removal of the dotted edge in (A).

s' and s''. More formally, $M^{s',s''} = \{m^{C',C''} \in \mathcal{M} : |C'| = s', |C''| = s''\}$. Furthermore, let us define $M^{**} \subseteq \mathcal{M}$ as follows:

$$M^{**} = \left(\bigcup_{s',s'' \in \mathbb{Z}^+ : \max\{s',s''\} \leq n-s'-s''} M^{s',s''} \right) \cup \left(\bigcup_{s',s'' \in \mathbb{Z}^+ : s'+s''=n} M^{s',s''} \right) \tag{9}$$

Then, the following holds:

$$R^{\{A\}}_{M^{**}} = \Pi^A \tag{10}$$

Proof. See Appendix I.

Based on Theorem 10, the size-based version of ODP only evaluates the movements in M^{**}, i.e., it uses $f_{M^{**}}$. This way, all movements that are evaluated by ODP can be expressed in terms of coalition sizes (see equation (9)). Moreover, none of the movements from a coalition of size $s = \lfloor \frac{2\times n}{3} \rfloor + 1, \ldots, n-1$ will be evaluated (see the following theorem).

Theorem 11. *The size-based version of ODP (i.e., the one that evaluates M^{**}) does not evaluate any of the possible ways of splitting a coalition of size $s \in \{\lfloor \frac{2\times n}{3} \rfloor + 1, \ldots, n-1\}$.*

Proof. See Appendix J.

Next, we show how to further modify ODP so that it searches subspaces of the integer partition graph. To this end, observe that the size-based version of ODP works in three main steps:

- for $s = 2, \ldots, \lfloor \frac{2 \times n}{3} \rfloor$, evaluate $m^{C',C''} \in M^{**} : |C'| + |C''| = s$;
- for $s = n$, evaluate $m^{C',C''} \in M^{**} : |C'| + |C''| = s$ and compute $t(A)$.
- make the best movements from $\{A\}$ using the function `getBestPartition`$(A, t(A))$.

Our modification involves changing this sequence of operation; the new sequence is:

- initialize $t(A) \leftarrow \{A\}$ and $f_{M^{**}}(C) \leftarrow v(C)$ for all $C \subseteq A$;
- for $s = 2, \ldots, \lfloor \frac{2 \times n}{3} \rfloor$: (1) evaluate $m^{C',C''} \in M^{**} : |C'| + |C''| = s$, (2) evaluate $m^{C',C''} \in M^{**} : \{|C'|, |C''|\} = \{s, n-s\}$, (3) update $t(A)$, and (4) make the best movements from $\{A\}$ using the function `getBestPartition`$(A, t(A))$;

The pseudo code of the size-based ODP algorithm, including the aforementioned modifications, can be found in Algorithm 4. To understand the intuition behind these latter modifications, let us consider an example of 10 agents, where the size-based ODP has just finished evaluating $m^{C',C''} \in M^{**} : |C'| + |C''| = s$ for $s = 2$ and $s = 3$. At this moment, although some movements in M^{**} were not yet evaluated, ODP *can reach some subspaces in the integer partition graph.* This is illustrated in Figure 6(A), where every movement not evaluated by ODP has been removed from the graph. As can be seen, some subspaces are reachable from $\Pi^A_{\{10\}}$—the bottom node in the graph. Consequently, based on corollary 1, the best coalition structure in those subspaces can easily be identified: simply repeat the process of splitting the coalition(s) in $\{A\}$ in the best way (out of all the ways that were evaluated by ODP thus far) until no such splitting is beneficial. Similarly, as soon as the following movements are evaluated: $m^{C',C''} \in M^{**} : |C'| + |C''| = 4$, more edges will be added to the graph, and so more subspaces will become reachable from the bottom subspace (see Figure 6(B)). Just as before, the best coalition structure in all of those subspaces can easily be identified. By repeating this process for every size s, ODP gradually evaluates more and more subspaces, until it eventually searched the entire space.

So far in this subsection, we have developed a size-based version of ODP, and shown how to modify it such that is searches integer partition-based subspaces. This has the following important advantage: at any point in time during execution, the part of the space that is *yet to be searched* can also be represented as the union of integer partition-based subspaces. As a result IP can be readily used to search the remaining part of the space. This is the key idea behind ODP-IP—a hybrid algorithm where IP runs in parallel with ODP[5] to solve the same problem instance. This division of work (between ODP and IP) gives ODP-IP the ability to calibrate itself automatically such that the amount of search assigned to each of its two constituent parts (ODP and IP) reflects the relative strength of that part with respect to the problem instance at hand. This is particularly important since IP could be significantly faster than ODP in some cases, while in some other cases it could be significantly slower (see Section 6 for more details).

Now that we have shown how IP and ODP can work jointly on the same search space, in the following subsections we show how this combination can be enhanced further. In particular, we will be focusing on how IP's performance can be improved by using the information that ODP has computed at any point in time.

[5]Whenever we are talking about ODP-IP, we mean the combination between IP and the size-based version of ODP. However, for simplicity, we will write "ODP" instead of "size-based version of ODP".

Algorithm 4: The size-based version of ODP.

Input: $v(C)$ for all $C \subseteq A$.
Output: CS^{**}—the best solution found at any point in time.

$t(A) \leftarrow \{A\}$; $CS^{**} \leftarrow \{A\}$ // initialization
// First, initialize $f_{M^{**}}(C)$ for every coalition, not just singletons
foreach $C \subseteq A$ **do**
  $f_{M^{**}}(C) \leftarrow v(C)$
// Second, compute $f_{M^{**}}(C)$ for every coalition of size $s = 2, \ldots, \lfloor \frac{2\times n}{3} \rfloor$
foreach $s = 2$ ***to*** $\lfloor \frac{2\times n}{3} \rfloor$ **do**
  foreach $C \subseteq A : |C| = s$ **do**
    foreach $s', s'' \in \mathbb{Z}^+$ **such that** $(s' + s'' = s) \wedge (\max\{s', s''\} \leq n - s' - s'')$ **do**
      foreach $\{C', C''\} :\in \Pi^C : \{|C'|, |C''|\} = \{s', s''\}$ **do**
        if $f_{M^{**}}(C) < f_{M^{**}}(C') + f_{M^{**}}(C'')$ **then**
          $f_{M^{**}}(C) \leftarrow f_{M^{**}}(C') + f_{M^{**}}(C'')$
  // update $f_{M^{**}(A)}$ and $t(A)$
  $temp \leftarrow t(A)$
  foreach $\{C', C''\} \in \Pi_2^A : \{|C'|, |C''|\} = \{s, n - s\}$ **do**
    if $f_{M^{**}}(A) < f_{M^{**}}(C') + f_{M^{**}}(C'')$ **then**
      $f_{M^{**}}(A) \leftarrow f_{M^{**}}(C') + f_{M^{**}}(C'')$
      $temp \leftarrow \{C', C''\}$
  // if $t(A)$ was updated, then update CS^{**}
  if $temp \neq t(A)$ **then**
    $CS^{**} \leftarrow$ `getBestPartition`$(A, t(A))$ // see the pseudo code in Algorithm 3
return CS^{**}

5.3. Speeding up IP's Depth-First Search

As mentioned earlier in Section 3.1, every time IP reaches a certain depth d in the search tree of a subspace Π_I^A, it adds a coalition, C_d, to a set of disjoint coalitions, $\{C_1, \cdots, C_{d-1}\}$. After that, it determines whether it is worthwhile to go deeper in the search tree, and that is by checking whether the inequality in (1) holds. If not, then the coalition structures in Π_I^A that start with $\{C_1, \cdots, C_{d-1}\}$ are considered **promising**, i.e., one of them could potentially have a value greater than $V(CS^{**})$—the value of the best coalition structure found so far. In this case, IP goes deeper in the search tree. However, we will now show how, with the help of ODP, some coalition structures can still be pruned even if they are promising.

The basic idea is to modify IP so that, for any subset of agents $C \subseteq A$, it keeps track of the value of the best partition of C that it has encountered so far. This is done using a table, w, with an entry for every possible coalition. In more detail, IP initially sets $w(C) = v(C)$ for all $C \subseteq A$. After that, every time IP reaches a certain depth, d, it performs the following operation:

$$\textbf{if} \quad w(\bigcup_{i=1}^{d} C_i) < \sum_{i=1}^{d} v(C_i) \quad \textbf{then} \quad w(\bigcup_{i=1}^{d} C_i) \leftarrow \sum_{i=1}^{d} v(C_i) \tag{11}$$

Since IP performs this operation every time it takes one step deeper in the search tree, the information in w is kept up-to-date throughout the search. Now, to use this information, IP is modified

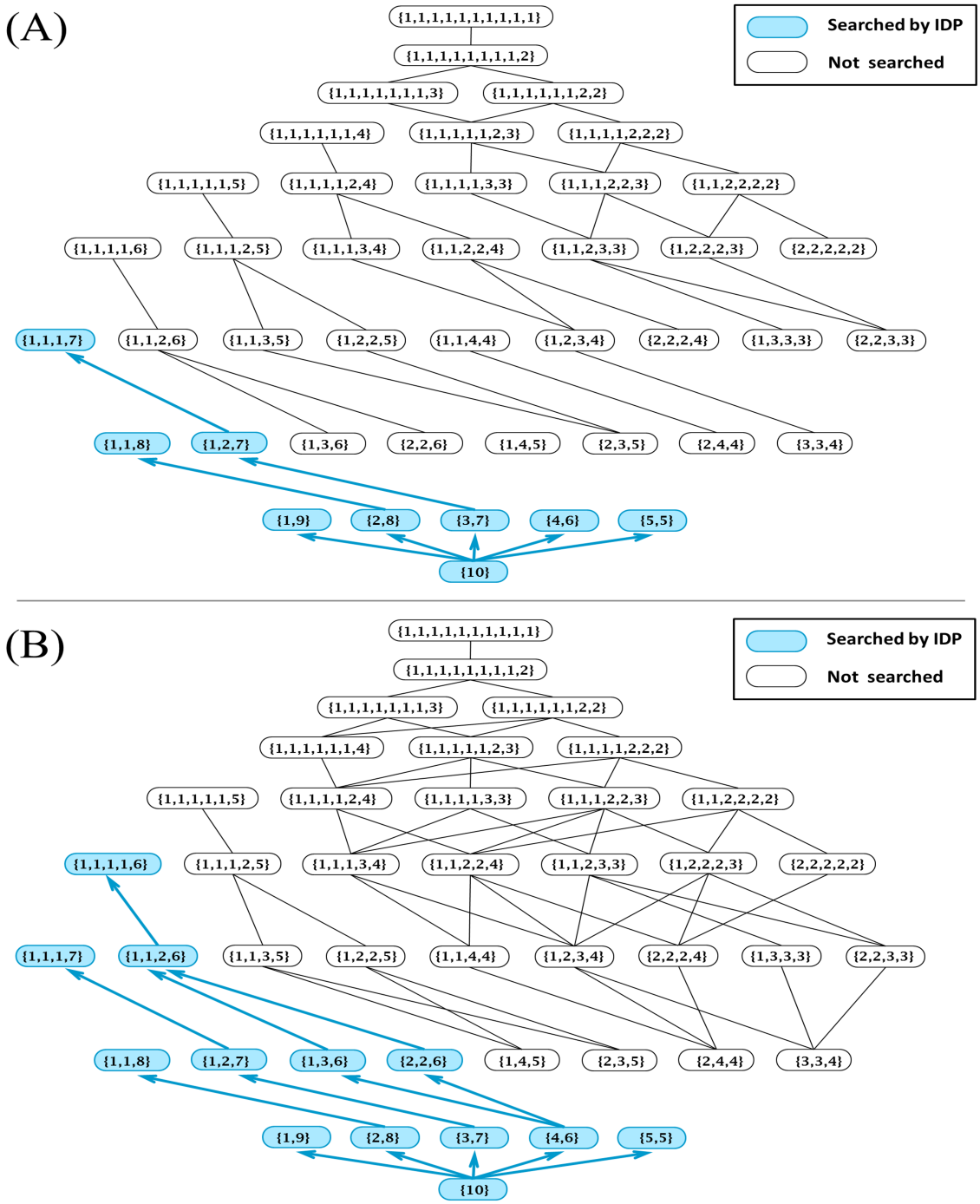

Figure 6: Illustration of how ODP gradually covers all subspaces given 10 agents. Figure 6(A) illustrates the subspaces that are reachable the moment ODP finishes evaluating the movements: $m^{C,C'} \in M^{**} : (|C| + |C'|) \in \{2, 3\}$. Figure 6(B) provides the same illustration, but with the evaluated movements being $m^{C,C'} \in M^{**} : (|C| + |C'|) = \{2, 3, \mathbf{4}\}$.

so that, at depth d, it checks whether any of the following inequalities holds:

$$w(\bigcup_{j=1}^{d} C_j) > \sum_{j=1}^{d} v(C_j) \tag{12}$$

$$w(C_d) > v(C_d) \tag{13}$$

If (12) holds, then $\{C_1, \ldots, C_d\}$ is not an optimal partition of $C_1 \cup \cdots \cup C_d$, and so there does not exist an optimal coalition structure, CS^*, such that $\{C_1, \ldots, C_d\} \subseteq CS^*$. Similarly, if (13) holds, then $\{C_d\}$ is not an optimal partition of C_d, and so there does not exist CS^* such that $C_d \in CS^*$. In either case, every coalition structure containing $\{C_1, \cdots, C_d\}$ can be skipped during the search (since we are only interested in finding CS^*). Note that this pruning occurs *even if the coalition structures containing* $\{C_1, \cdots, C_d\}$ *are promising*. This is because the pruning here occurs whenever CS^* cannot possibly belong to the coalition structures containing $\{C_1, \cdots, C_d\}$, *even if one of those coalition structures was indeed better than* CS^{**}—the best coalition structure found so far.

Now, knowing that ODP runs in parallel with IP, we can improve the above technique as follows. Instead of having IP use a table, w, and ODP use another table, $f_{M^{**}}$, we modify IP so that it uses the same table as ODP. Formally, we replace w with $f_{M^{**}}$ in (11), (12) and (13). This implicitly means that IP will not only check the best partitions that it had encountered; it will also check those encountered by ODP.

To better understand the effect that ODP has on the new branch-and-bound technique, let us consider an example of 19 agents. With such a relatively small number of agents, ODP can compute the optimal partitions of all the coalitions of sizes 2 to 9 in a very short time (e.g., less than 0.2 second on our standard desktop PC, see Section 6 for more details). Now suppose that, after this short time, IP started searching subspace $\Pi^A_{\{2,2,2,2,1,1,3,3,3\}}$. As mentioned earlier, IP goes deeper in the search tree as long as it encounters promising coalitions. For instance, suppose that all coalition structures containing $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}, \{a_9\}$ happened to be promising. With the new branch-and-bound technique, and with the information now provided by ODP, this combination of coalitions would not be reached by IP, *even if they were indeed promising*, unless all of the following hold:

- $\{\{a_1, a_2\}\}$ happens to be an optimal partition of $\{a_1, a_2\}$
- $\{\{a_3, a_4\}\}$ happens to be an optimal partition of $\{a_3, a_4\}$
- $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ happens to be an optimal partition of $\{a_1, \ldots, a_4\}$
- $\{\{a_5, a_6\}\}$ happens to be an optimal partition of $\{a_5, a_6\}$
- $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}$ happens to be an optimal partition of $\{a_1, \ldots, a_6\}$
- $\{\{a_7, a_8\}\}$ happens to be an optimal partition of $\{a_7, a_8\}$
- $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}$ happens to be an optimal partition of $\{a_1, \ldots, a_8\}$
- $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}, \{a_9\}$ happens to be an optimal partition of $\{a_1, \ldots, a_9\}$

The probability of this happening (assuming that every partition has an equal probability of being an optimal one) is only $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{15} \times \frac{1}{2} \times \frac{1}{203} \times \frac{1}{2} \times \frac{1}{4140} \times \frac{1}{21147} = 2.3 \times 10^{-13}$. This example clearly demonstrates the great potential that this new branch-and-bound technique has in terms of speeding up the search.

5.4. Searching Multiple Subspaces Simultaneously

In this subsection, we show how to modify IP such that it searches multiple subspaces simultaneously to avoid repeating certain operations. For presentation clarity, we will postpone the formal description of our technique until after we have presented the basic idea through an example of five subspaces. To this end, recall that IP searches any subspace in a depth-first manner. As such, for any given subspace, Π^A_I, the shape of the search tree depends on the way in which the integers in I are ordered. With this in mind, we note that, given any particular ordering of the integers, the first few levels of a search tree can be *exactly the same* as those of several other search trees. For instance, the first two levels are exactly the same in the search trees of the subspaces that are represented by the following ordered integer partitions: $I_1 = \{\mathbf{2, 4}, 4\}$, $I_2 = \{\mathbf{2, 4}, 1, 3\}$, $I_3 = \{\mathbf{2, 4}, 2, 2\}$, $I_4 = \{\mathbf{2, 4}, 1, 1, 2\}$, and $I_5 = \{\mathbf{2, 4}, 1, 1, 1, 1\}$. Searching any of those subspaces in a depth first manner (as is the case with IP) involves constructing pairs of disjoint coalitions, $C_1, C_2 : |C_1| = \mathbf{2}, |C_2| = \mathbf{4}$ (for more details, see Section 3.1). Now, instead of repeating this process for every one of those five subspaces, one can perform it only once. More specifically, for every pair, $C_1, C_2 : |C_1| = \mathbf{2}, |C_2| = \mathbf{4}$, one can perform the following steps:

1. Compute the value of $\{C_1, C_2, A \backslash (C_1 \cup C_2)\}$—the only coalition structure in $\Pi^A_{I_1}$ that contains C_1 and C_2.
2. Find the best partition of $A \setminus (C_1 \cup C_2)$ into two coalitions of sizes 1, 3, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi^A_{I_2}$ that contains C_1 and C_2.
3. Find the best partition of $A \setminus (C_1 \cup C_2)$ into two coalitions of sizes 2, 2, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi^A_{I_3}$ that contains C_1 and C_2.
4. Find the best partition of $A \setminus (C_1 \cup C_2)$ into three coalitions of sizes 1, 1, 2, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi^A_{I_4}$ that contains C_1 and C_2.
5. Compute the value of $\{C_1, C_2\} \cup_{a_i \in A \setminus (C_1 \cup C_2)} \{\{a_i\}\}$—the only coalition structure in $\Pi^A_{I_5}$ that contains C_1 and C_2.
6. Select the best out of the coalition structures that were found in the above five steps.

The above six steps return the best coalition structure containing C_1 and C_2 in: $\cup_{i=1}^5 \Pi^A_{I_i}$. By performing these steps for every pair, $C_1, C_2 : |C_1| = \mathbf{2}, |C_2| = \mathbf{4}$, we find the best coalition structure in $\cup_{i=1}^5 \Pi^A_{I_i}$.

Next, we will show how to significantly speed up the above technique using the information provided by ODP. To this end, consider an example where IP started searching $\Pi^A_{I_1}, \ldots, \Pi^A_{I_5}$ after ODP has finished evaluating the movements $m^{C,C'} \in M^{**} : (|C| + |C'|) \in \{2, 3, 4\}$. This means that, for all $C \subseteq A : |C| \in \{2, 3, 4\}$, ODP has finished computing $f_{M^{**}}(C)$. In this case, for any pair, $C_1, C_2 : |C_1| = 2, |C_2| = 4$, it is possible to find the value of the best coalition structure containing C_1 and C_2 in $\cup_{i=1}^5 \Pi^A_{I_i}$ without having to examine the different partitions of $A \setminus (C_1 \cup C_2)$ as in the above six steps. Instead, we can now perform a single step, which is:

- Compute $v(C_1) + v(C_1) + f_{M^{**}}(A \setminus (C_1 \cup C_2))$.[6]

By repeating this step for every pair $C_1, C_2 : |C_1| = 2, |C_2| = 4$, we find a coalition structure:

$$\{C^*_1, C^*_2, C^*_3\} \in \underset{CS \in \Pi^A_{I_1}}{\arg\max}\ v(C_1) + v(C_2) + f_{M^{**}}(C_3).$$

What remains is to partition C^*_3 in the best way using the movements in M^{**} (so far we only know the value of that partition, which is $f_{M^{**}}(C^*_3)$; we don't yet know the partition itself). This

[6]This is because, in this example, $A \setminus (C_1 \cup C_2)$ is a coalition of 4 agents, which means that ODP has already computed $f_{M^{**}}(A \setminus (C_1 \cup C_2))$.

can be done by simply replacing C_3^* with: `getBestPartition`$(C_3^*, t(C_3^*))$, which partitions C_3^* by making the best out of all the movements that ODP has evaluated so far (see Algorithm 3). This process is illustrated in Figure 7(A), where IP searches $\Pi^A_{I_1}$, and the partitioning of C_3^* using `getBestPartition` is illustrated by the movements from $\Pi^A_{I_1}$ to the other subspaces, $\Pi^A_{I_2}$, $\Pi^A_{I_3}$, $\Pi^A_{I_4}$, and $\Pi^A_{I_5}$.

So far, we have shown how subspaces can be searched simultaneously, and that is by partitioning *exactly one* coalition using `getBestPartition`. However, one can partition *more than one* coalition. This way, more subspaces can be searched simultaneously. For example, while searching $\Pi^A_{I_1}$, if IP evaluates every $\{C_1, C_2, C_3\} \in \Pi^A_{I_1}$ as follows: $f_{M^{**}}(C_1)+f_{M^{**}}(C_2)+f_{M^{**}}(C_3)$, then the result of this search will be a coalition structure CS' that maximizes $f_{M^{**}}(C_1) + f_{M^{**}}(C_2) + f_{M^{**}}(C_3)$. By replacing every coalition $C \in CS'$ with `getBestPartition`$(C, t(C))$, we end up with the best coalition structure in all the subspaces that are reachable from $\Pi^A_{I_1}$. This is illustrated in Figure 7(B).

When searching multiple subspaces simultaneously, it is important to modify the branch-and-bound technique used by IP. To this end, recall that when searching a single subspace, Π^A_I, IP encounters a new combination of disjoint coalitions every time it takes one step deeper in the search tree. For every such combination, $\{C_1, \ldots, C_d\}$, IP computes an upper bound on the value of every coalition structure $CS \in \Pi^A_I : \{C_1, \ldots, C_d\} \subseteq CS$. Now if this upper bound happens to be smaller than $V(CS^{**})$—the value of the best solution found so far—then the combination is deemed unpromising. However, when searching multiple subspaces, e.g., $\Pi^A_{I_1}, \ldots, \Pi^A_{I_5}$, the upper bound must be computed taking into consideration all of those subspaces. In our example, the upper bound must be on the value of every $CS \in \left(\Pi^A_I \cup \cdots \cup \Pi^A_{I_5}\right) : \{C_1, \ldots, C_d\} \subseteq CS$. Appendix L provides more details on how to split multiple integers, and compares this with the case where only a single integer is partitioned.

Finally, it is important to note that IP generally ignores the order of the coalitions within a coalition structure. For instance, given 10 agents, $\{\{a_1, a_2\}, \{a_3, a_4, a_5, a_6\}, \{a_7, a_8, a_9, a_{10}\}\}$ and $\{\{a_1, a_2\}, \{a_7, a_8, a_9, a_{10}\}, \{a_3, a_4, a_5, a_6\}\}$ are considered the same, and so only one of the them is generated. However, in the case where multiple subspaces are being searching simultaneously, the order matters. For instance, let us consider the example from Figure 7(A). Here, since a coalition of size 4 will be replaced with its optimal partition, IP will have to evaluate every $\{C_1, C_2, C_3\} \in \Pi^A_{\{2,4,4\}}$ as follows: $v(C_1) + v(C_2) + f_{M^{**}}(C_3)$. As can be seen, $v(\{a_1, a_2\}) + v(\{a_3, a_4, a_5, a_6\}) + f_{M^{**}}(\{a_7, a_8, a_9, a_{10}\})$ is different than $v(\{a_1, a_2\})+v(\{a_7, a_8, a_9, a_{10}\})+f_{M^{**}}(\{a_3, a_4, a_5, a_6\})$, and so both must be calculated.

5.5. Summary and Complexity of ODP-IP

Below is a summary of the main modifications that we have made to ODP and IP to enable them to help each other when running in parallel as in ODP-IP:

- **Enable ODP to search subspaces of the integer partition graph**: To do this, first ODP must use $f_{M^{***}}$ instead of f_{M^*}. Second, for $s = 2, \ldots, \left\lfloor \frac{2 \times n}{3} \right\rfloor$, the algorithm must: (1) evaluate $m^{C',C''} \in M^{**} : |C'| + |C''| = s$, (2) evaluate $m^{C',C''} \in M^{**} : \{|C'|, |C''|\} = \{s, n - s\}$, (3) update $t(A)$, and (4) make the best movements from $\{A\}$ using the function `getBestPartition`$(A, t(A))$. The pseudo code can be found in Algorithm 4;

- **Speed up IP's depth-first search**: To do this, whenever a coalition, C_d, is added to a set of disjoint coalitions, $C_1, \ldots, C_{d-1}$, check whether $\{C_d\}$ and $\{C_1, \ldots, C_d\}$ are the best partitions of C_d and $C_1 \cup \cdots \cup C_d$, respectively, that have been encountered by IP and/or ODP so far. If not, skip every coalition structure containing $C_1, \ldots, C_d$.

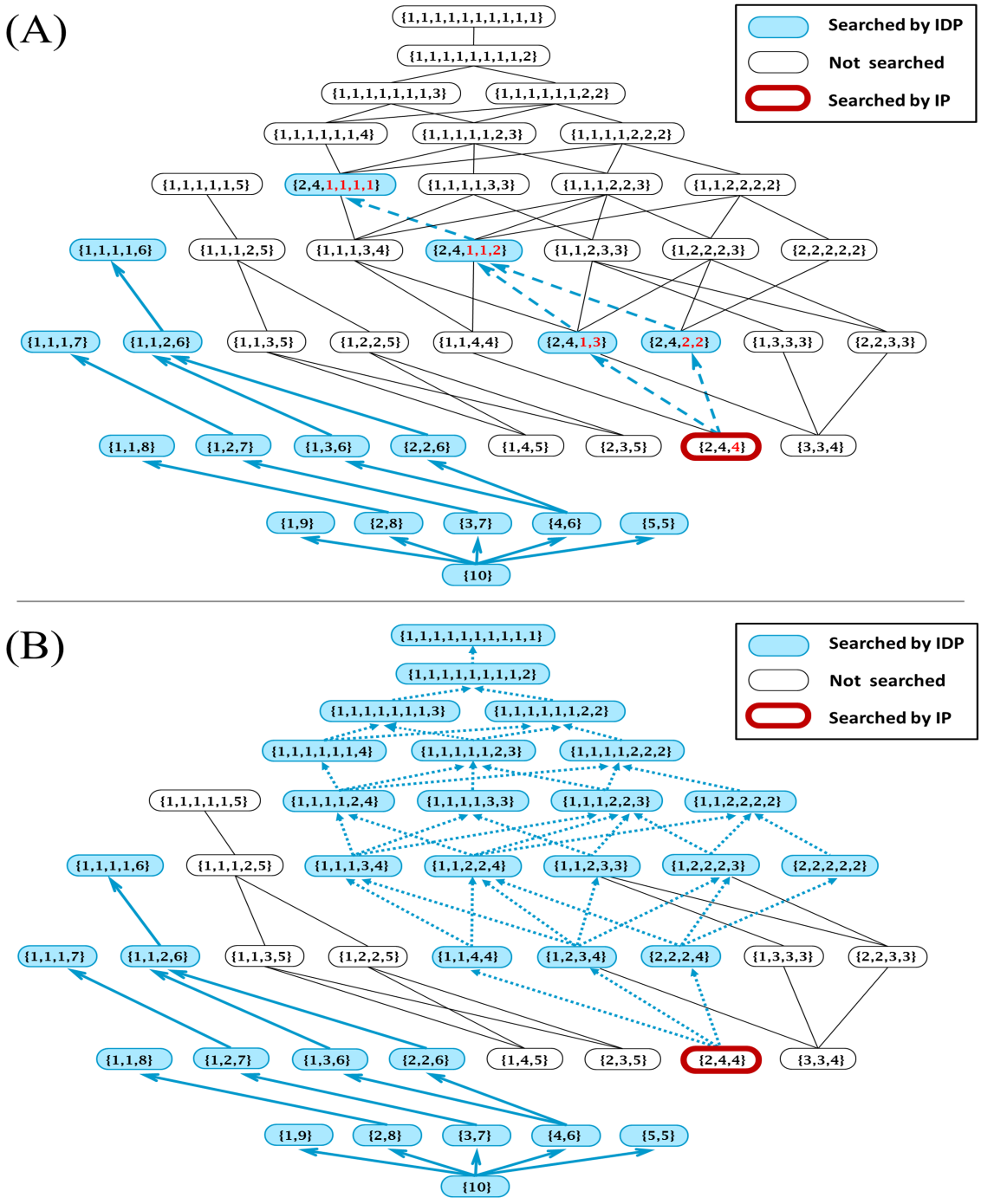

Figure 7: How IP can search multiple subspaces simultaneously given that ODP has already computed $f_{M^{**}}(C)$: $|C| \in \{2, 3, 4\}$. In Figure 7(A), several subspaces are searched simultaneously, and that is by splitting *exactly one* coalition. In Figure 7(B), more subspaces are searched simultaneously, and that is by splitting *multiple* coalitions.

- **Enable IP to search multiple subspaces simultaneously**: When searching a subspace, Π_I^A, identify the integer partitions that are reachable from I using the movements that have been evaluated by ODP thus far. Now, let $I^* \subseteq I$ be the integers in I that will be split to reach other integer partitions. Then, for every coalition structure, $CS \in \Pi_I^A$, evaluate every $C \in CS$ as $f_{M^{**}}(C)$ if the size of C corresponds to an integer in I^*, otherwise evaluate it as $v(C)$. Finally, modify IP's branch-and-bound technique such that the upper bounds reflect the subspaces whose integer partitions are reachable from I by splitting the integers in I^*. The details of this modification are in Appendix L.

We conclude this section with the following theorem.

Theorem 12. *Given n agents, ODP-IP runs in $O(3^n)$ time.*

Proof. See Appendix K.

Having presented ODP-IP, the following section evaluates both of our algorithms, namely ODP and ODP-IP.

6. Performance Evaluation

This section is divided into two subsections: the first evaluates ODP, while the second evaluates ODP-IP.

6.1. Evaluating ODP

We know that DP evaluates $1/2(3^n - 1)$ movements (see Appendix K), while ODP evaluates $\frac{1}{2}\left(3^{n-1} - 1\right)$ movements (see Appendix H). In other words, ODP evalautes 33% of the movements compared to ODP. Furthermore, we know that the size-based version of ODP (i.e., the version that is compatible with IP) performs a number of movements that is bounded by the aforementioned two numbers. Figure 8 compares those numbers, with n running from 5 to 40. As can be seen, as the number of agents increases, the percentage of movements that are evaluated by the size-based version of ODP drops (compared to that of DP), and converges at around 37%. This is very close to the optimal reduction in movements, which is 33%.

6.2. Evaluating ODP-IP

ODP-IP was developed to try and obtain the best features of ODP and IP, namely: (1) being anytime, (2) running in $O(3^n)$ time, and (3) being on average as fast as (if not faster than) the fastest of the two algorithms, ODP and IP. While our analysis in Section 5.1 showed that ODP-IP indeed has features (1) and (2), the following experiments are meant to verify whether ODP-IP has feature (3). The algorithms were implemented in Java, and tested on a PC equipped with an Intel® Core™ i7 processor (3.40GHz) and 12GB of RAM.

Observe that the number of operations performed by ODP is not influenced by the characteristic function at hand (i.e., it depends solely on the number of agents involved). On the other hand, the number of operations performed by IP (and consequently by ODP-IP) depends on the effectiveness of IP's branch-and-bound technique, which in turn depends on the characteristic function at hand. With this in mind, we compare the termination times of all three algorithms (ODP, IP, and ODP-IP) given different value distributions, namely:

1. **Uniform**, as studied by Larson and Sandholm (2000): $\forall C \in \mathcal{C}^A, v(C) \sim U(a, b)$ where $a = 0$ and $b = |C|$.

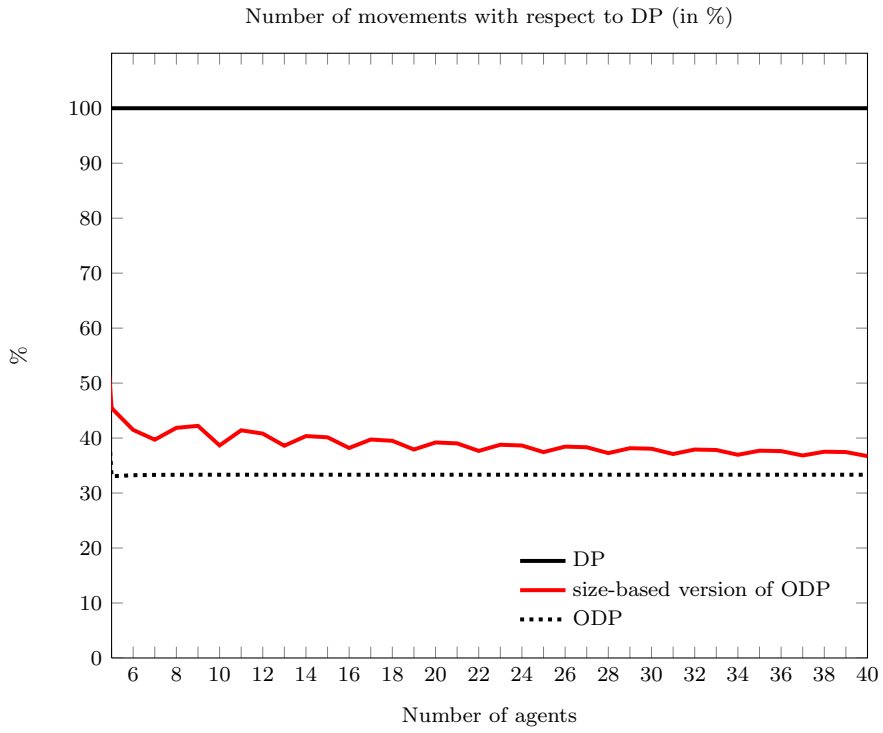

Figure 8: Percentage of movements made by the size-based version of ODP vs. DP converges ae around 37% which is very close to he optimal reduction of 33%.

2. **Normal**, as studied by Rahwan et al. (2007): $\forall C \in \mathcal{C}^A, v(C) \sim N(\mu, \sigma^2)$ where $\mu = 10 \times |C|$ and $\sigma = 0.1$.
3. **NDCS** (Normally Distributed Coalition Structures), proposed by Rahwan et al. (2009): $\forall C \in \mathcal{C}^A, v(C) \sim N(\mu, \sigma^2)$, where $\mu = |C|$ and $\sigma = \sqrt{|C|}$. The rationale behind developing NDCS came from the authors' observation that, with the above Uniform and Normal distributions, a coalition structure is *less* likely to be optimal if it contains *more* coalitions. In order to develop a test-bed that is free from such bias, the authors proposed NDCS and proved it to be the only *coalition-value* distribution that results in normally-distributed *coalition structure-values*. As a result, when using NDCS, all coalition structures are equally likely to be optimal.
4. **Modified Uniform**, as proposed by Service and Adams (2010): Every coalition's value is first drawn from $U(0, 10 \times |C|)$, and then increased by a random number $r \sim U(0, 50)$ with 20% probability.
5. **Modified Normal**, proposed by Rahwan et al. (2012) as a natural counterpart to the Modified Uniform distribution. In particular, each coalition's value if first drawn from $N(10 \times |C|, 0.01)$, and then increased by a random number $r \sim U(0, 50)$ with 20% probability.
6. **Exponential:** $\forall C \in \mathcal{C}^A, v(C) \sim |C| \times Exp(\lambda)$, where $\lambda = 1$.
7. **Beta:** $\forall C \in \mathcal{C}^A, v(C) \sim |C| \times Beta(\alpha, \beta)$, where $\alpha = \beta = 0.5$.
8. **Gamma:** $\forall C \in \mathcal{C}^A, v(C) \sim |C| \times Gamma(k, \theta)$, where $k = \theta = 2$.
9. **Agent-based Uniform**, as proposed by Rahwan et al. (2012): Each agent a_i is assigned a random *"power"*, $p_i \sim U(0, 10)$, reflecting its *average* performance over all coalitions. Furthermore, for any coalition $C \ni a_i$, the *actual* power of a_i in C is: $p_i^C \sim U(0, 2p_i)$.

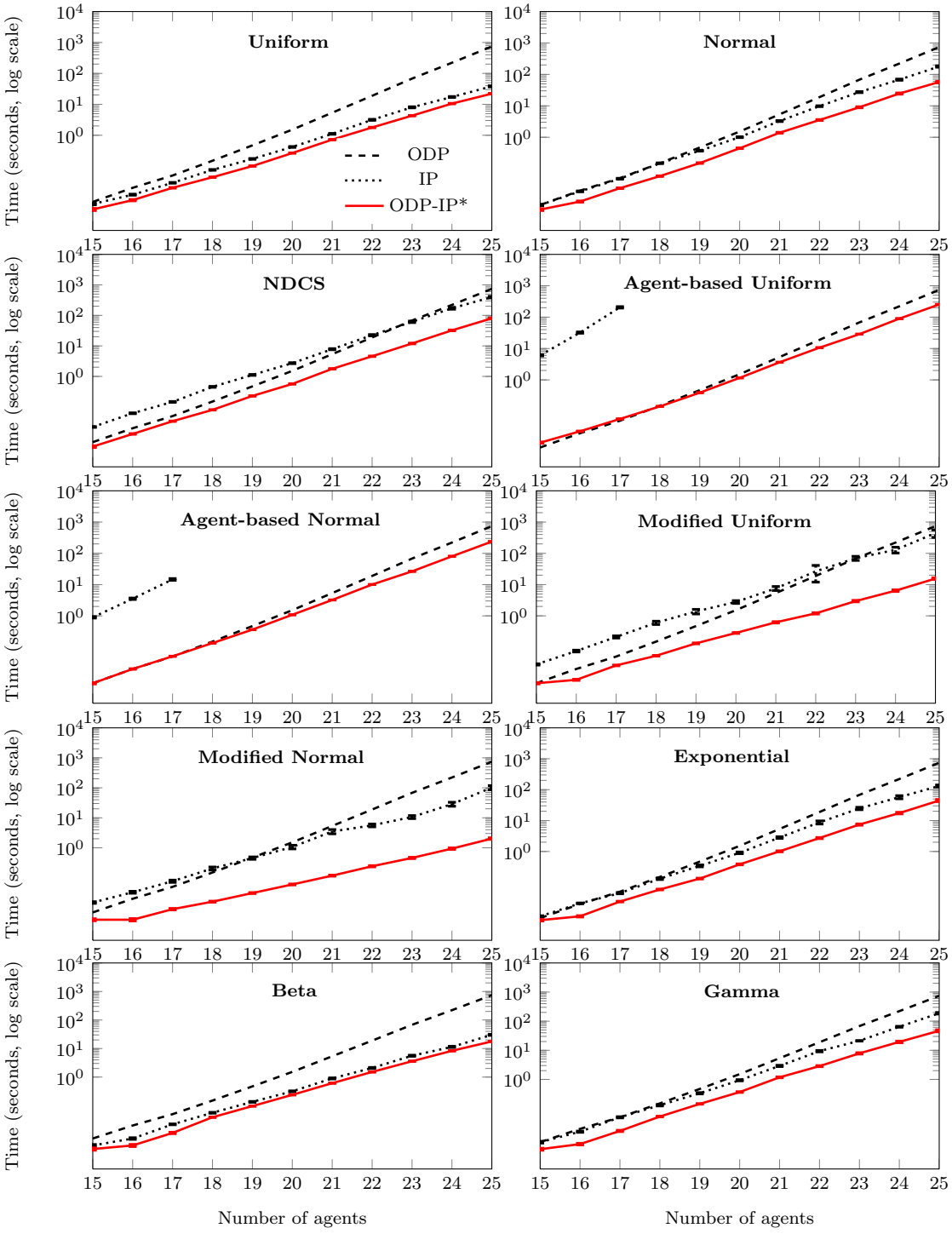

Figure 9: Time performance of ODP-IP vs. ODP and IP.

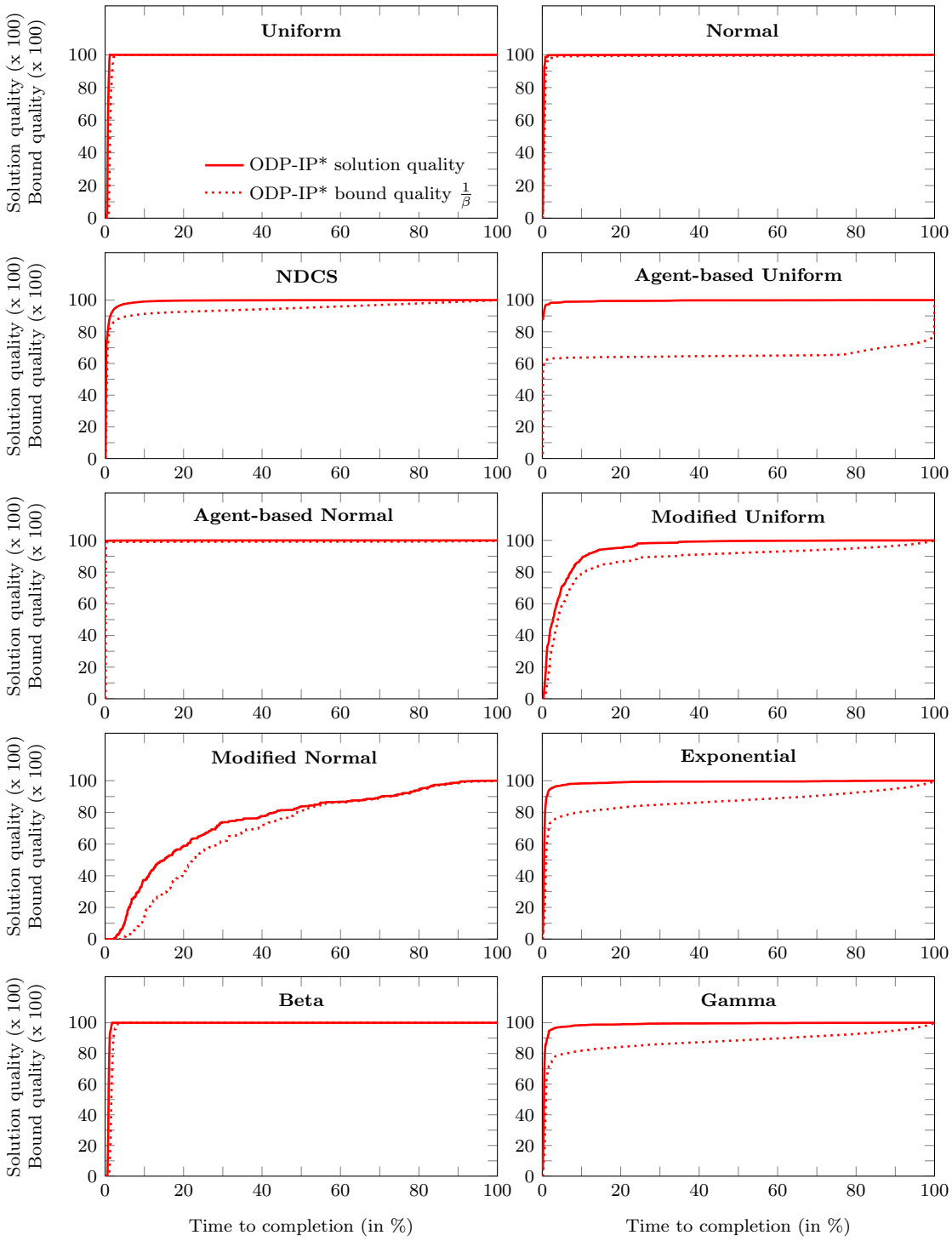

Figure 10: Solution quality and bound quality of ODP-IP.

Then, a coalition's value is computed as the sum of the powers of its member. That is, $\forall C \in \mathcal{C}^A, v(C) = \sum_{a_i \in C} p_i^C$.

10. **Agent-based Normal**, proposed in this article. As the name suggests, it is similar to the above distribution except that every agent's average and actual powers are drawn from normal, rather than uniform distributions. Formally, $\forall a_i \in A, p_i \sim N(10, 0.01)$ and $\forall a_i, C \subseteq A : a_i \in C, p_i^C \sim N(p_i, 0.01)$ and $\forall C \in \mathcal{C}^A, v(C) = \sum_{a_i \in C} p_i^C$.
11. **Size-independent Distributions**, proposed in this article. These are similar to distributions 1 to 8, excepted that every $|C|$ is replaced with 1. This removes the dependency relation between the coalition value and the coalition size. We will add "Size-independent" to the distribution name when referring to the version that is free from such dependency, e.g., the above **Uniform** distribution means: $v(C) \sim U(0, |C|)$, whereas the **Size-independent Uniform** distribution means: $v(C) \sim U(0, 1)$.

For each of the above distributions, we plotted the termination times of ODP, IP, and ODP-IP given different numbers of agents (see Figures 9). Here, time is measured in seconds, and plotted *on a log scale*. As can be seen, for all the aforementioned distributions, ODP-IP is faster than the fastest of the two other algorithms (by one or two orders of magnitude for some distributions). This is because the modifications introduced to IP and ODP (in sections 5.2, 5.3 and 5.4) allow the two algorithm to help one another, leading to a positive synergy when they join forces as in ODP-IP. Observe that those modifications (to IP and ODP) involve the use of branch-and-bound techniques, effectiveness of which depends heavily on the characteristic function at hand. As such, the resulting synergistic effect varies from one value distribution to another. This of course does not only influence the termination time (as we have seen in the aforementioned figures), but also influences how the solution quality, and established bounds, improve during the run time of ODP-IP (as we will see in the following figures).

Next, we evaluate the anytime property of ODP-IP. The results in Figures 10 are shown for 25 agents In particular, the x-axis in the figures corresponds to the percentage of time that has elapsed, with 0% being the time at which the algorithm starts, and 100% being the time at which it terminates. For every percentage of time, t%, we report the following:

- *Solution quality:* This is computed as the ratio between the value of the "current" best solution (found at t% of the runtime) and the value of the optimal solution (found at 100%). Formally, the solution-quality plot represents: $(\frac{V(CS^{**}) \times 100}{V(CS^*)})$%.
- *Bound quality:* This is computed as the ratio between the value of the "current" best solution and the maximum upper bound of all "remaining" subspaces (i.e., those that were not yet searched nor pruned).

With a few exceptions, the results show that if ODP-IP was interrupted before running to completion, it may still return a solution with relatively high quality and good guarantees (i.e., bound quality). Specifically, in terms of the guarantees that the algorithm places on its solution, we find that:

- With Agent-based Uniform and Modified Normal distributions, it takes a substantial percentage of the runtime until the guarantees reach 80%.
- With NDCS, Modified Uniform, Exponential, and Gamma distributions, the guarantees exceed 80% (or 90% in the NDCS case) after 10% of the runtime.
- With Normal, Agent-based Normal, Uniform, and Beta distributions, the guarantees exceed 99% after about 3% of the runtime.

In terms of solution quality, our results show that:

- With the Modified Normal distribution, it takes a substantial percentage of the runtime for solution quality to reach 80%.
- With the Modified Uniform distribution, solution quality reaches 90% after 10% of the runtime.
- With all other distributions, solution quality reaches 95% (if not 100%) after 3% of the runtime.

This concludes the empirical evaluation of both our algorithms, ODP and ODP-IP.

7. Related Work

The term *"complete set partitioning problem"* was introduced by Lin (1975) as a special class of set partitioning problems. The application that motivated this study was the structuring of corporate tax in the United States. In particular, several states, such as Ohio, allowed any corporation to file its annual unemployment compensation payment either on a subsidiary basis or by grouping subsidiaries into disjoint aggregations. The total unemployment compensation tax payment depended on particular aggregations chosen by the parent corporation. To provide an exact solution to this optimization problem, Lin and Salkin (1979; 1983) developed an integer programming algorithm with *branch search enumeration* (Garfinkel and Nemhauser, 1969) that runs in $O(2^{n^2/2})$. This algorithm was later on shown by Yeh (1986) to be substantially slower than DP. The same algorithm was later on re-discovered in the combinatorial-auctions literature, to solve the winner determination problem in cases where every possible bundle of goods has a (possibly zero-valued) bid placed on it (Rothkopf et al., 1995). Sandholm (2002) provided further analysis of the complexity of this algorithm, showing that its run time is polynomial in the size of the input (i.e., polynomial in the number of possible subsets of goods). However, this analysis did not expose the redundant operations in DP as we did in this article. As such, by developing ODP, we make a contribution to the literature on winner determination.

The advancements in multi-agent technologies in the late 90's renewed the interest in the complete set partitioning problem. In this literature, the problem was called the *"coalition structure generation problem"*, and was studied in the context of dividing agents into coalitions so as to maximize social welfare. In this context, a number of exact, *anytime* algorithms were proposed, with the focus being on establishing a bound on the quality of their *"interim"* solutions (i.e., the solutions that the algorithms return during execution, not after completion). These algorithms can be divided into two categories, based on the techniques they use:

- the first class of algorithms focuses on (1) proposing a criteria based on which the search space is divided into disjoint and exhaustive subspaces, and (2) identifying a sequence in which these subspaces should be searched, such that the worst case bound on solution quality is guaranteed to improve after each subspace. We will denote by $S_1, \ldots, S_k$ the chosen sequence of subspaces, and by β_i the bound established after searching $S_1 \cup \cdots \cup S_i$. This bound is based solely on comparing the coalition structures that have already been searched against those that are yet to be searched (i.e., those in $S_{i+1} \cup \cdots \cup S_k$), without paying attention to the actual coalition values at hand. This makes such algorithms applicable in settings where only *coalition structure values* can be observed, not *coalition values*. This also makes the bounds independent of the coalition-value distribution, meaning that such algorithms can guarantee their bounds, regardless of the distribution.

  Any algorithm in this class can be extended (possibly in different directions) by specifying the technique(s) used to search the subspaces. Such technique(s) can capitalize on the extra

information accrued during the actual search, e.g., to avoid examining all solutions in a subspace, or to establish bounds other than, and hopefully better than, $\beta_i : i = 1, \ldots, k$. The advantage of such an extension is that it can place guarantees on its bounds; they cannot be worst than $\beta_i : i = 1, \ldots, k$.

The first algorithm in this class was proposed in the seminal article by Sandholm et al. (1999), where the proposed sequence was: $S_1 = \Pi^A_1 \cup \Pi^A_2$ and $S_i = \Pi^A_{n-i+2} : i = 2, \ldots, n-1$. Two particularly interesting bounds were $\beta_1 = n$ and $\beta_2 = \lceil n/2 \rceil$; the authors proved that S_1 and S_2 are the *smallest subsets* of solutions that one can search to establish the *tight* bounds n and $\lceil n/2 \rceil$, respectively (unless, of course, one uses extra information obtained from the characteristic function at hand). An alternative algorithm was later on proposed by Dang and Jennings (2004), which proposed a different sequence, along with a different set of bounds, compared to Sandholm et al. This algorithm was able to establish certain bounds by going through a fewer number of solutions. Another algorithm was proposed by Rahwan et al. (2011), where every S_i is represented as the union of integer partition-based subspaces. As such, one can readily extend this algorithm by using IP (or ODP-IP) to search every S_i.

All the algorithms discussed so far in this class are proposed for *characteristic function games*, where there are no *"externalities"* (i.e., influences between co-existing coalitions). Rahwan et al. (2009) proposed the first algorithm for *partition function games* (PFGs), i.e., games with externalities. In particular, they focused on two sub-classes: (1) PFG$^+$, where externalities are non-negative, and (2) PFG$^-$, where externalities are non-positive. Arguably, many realistic partition function games fall under one of those two sub-classes.[7] The algorithm was later on extended by Banerjee and Kraemer (2010) to settings where agents are grouped into categories, or *"types"*. Here, the authors assume that if two coalitions C_1 and C_2 merge, then the externality imposed by such merging on a third coalition C_3 is non-negative if the types of the agents in $C_1 \cup C_2$ do not overlap with those of the agents in C_3. Otherwise, the externality is non-positive. Let us denote this class of games as PFG^{type}. The authors argued that this class is intuitive, and maps to a number of applications.

- The second class of anytime, exact algorithms focuses on finding, and recognizing, an optimal coalition structure as quickly as possible. The main techniques used here are (1) branch-and-bound, where the aim is to identify, and thus avoid evaluating, unpromising combinations of coalitions, and (2) dynamic-programming, where the aim is to avoid evaluating any combination of coalitions more than once.

  Arguably, the first algorithm in this class is IP, due to Rahwan et al. (2007, 2009), which uses branch-and-bound techniques as described earlier in Section 3.1. A distributed version of IP was later on proposed by Michalak et al. (2010) as the first distributed, exact algorithm for coalition structure generation. The algorithm incorporates a pre-processing stage whereby "filter rules" are applied to identify coalitions that have no potential of belonging to an optimal solution. This stage is implemented in a distributed fashion; the coalition space is divided into n disjoint subsets that are each assigned to a different agent. Each agent applies the filter rules to its share, and sends the results to the others. The algorithm then proceeds to search one integer partition-based subspace at a time, just like IP. This search, however, is carried out in a distributed fashion; each subspace is divided into n disjoint pieces that are each searched by a different agent. Although the agents' shares contain virtually the same number of solutions, the search effort may differ from one agent to another, depending on the effectiveness of the branch-and-bound technique. Therefore, a load-balancing technique

[7] Observe that each of these two sub-classes is a generalization of characteristic function games.

is introduced to handle any such potential differences.

Since the initial publication of ODP in (Rahwan and Jennings, 2008b), an anytime version of the size-based version of ODP was proposed by Service and Adams (2011). In this version, an initial stage is added whereby, for each coalition, C, the algorithm identifies and stores the subset of C that has the highest value. Using this extra information, the authors showed how, every time the algorithm finishes evaluating the splits of all coalitions of a certain size, s, a coalition structure can be constructed of which the value is guaranteed to be within a bound r from the optimal one, where $r = \max\{i : i \in \mathbb{Z}, s \leq \lfloor \frac{n}{i} \rfloor\}$. The termination time of this modified ODP is almost identical to that of the original ODP (except for the time required to run the added initial stage). This implies that the modified ODP algorithm is significantly slower than ODP-IP for all coalition-value distributions mentioned earlier in Section 6.2 (see the difference in termination between ODP-IP and ODP in Figure 9). Moreover, the guarantees provided by Service and Adams's modified ODP do not exceed 50% until termination, while the guarantees provided by ODP-IP often exceeds 80% (or even 99%) within only 10% (or even 3%) of the termination time (see Figure 10). Finally, Service and Adams's modified ODP requires twice as much memory compared to ODP-IP (to store the best subset of every coalition).

So far in this class, we focused on algorithms for characteristic function games. Next, we shift our attention to partition function games. In such settings, due to externalities, any coalition may have different values depending on the coalition structure it is embedded in. It is not difficult to show that in the most general case, where externalities are arbitrary, it is impossible to place any bound on solution quality without examining every single coalition structure. However, for two common classes of partition function games, namely PFG^+ and PFG^-, Rahwan et al. (2009, 2012) proved that it is possible to compute upper and lower bounds on the values of any set of disjoint coalitions in linear time. These bounds can then be used to identify unpromising search directions using techniques similar to those used in IP. Similarly, Banerjee and Kraemer (2010) proposed an extension of IP to handle externalities in PFG^{type} settings.

Another extension of ODP, though not anytime as Service and Adams's, is due to Voice et al. (2012), which focuses on a special class of coalitional games inspired by Myerson (1976). In these games, the space of feasible coalitions is restricted by a graph, G, where nodes represent agents, and edges represent possibilities of collaboration; a coalition C is only feasible if all the agents in C induce a connected subgraph of G. A "feasible" coalition structure is then simply one where all coalition are feasible. Recall that we have shown in Theorem 4 that, for any arbitrary set of movements between coalition structures, if DP only evaluates those movements it will find the best coalition structure reachable using those movements. Voice et al. focused on the set of movements between feasible coalition structures (i.e., restricted by G). This provides significantly speedups in computation when the graph is sparse.

In this article we focused on the classical representation of characteristic function games, where the value of any coalition, $C \subseteq A$, is returned by a characteristic function, $v : 2^A \rightarrow R$. However, there are other works that study alternative representations designed to efficiently capture situations where the characteristic function has some structure. For instance, Ueda et al. (2010) studied coalition structure generation under the DCOP (Distributed Constraint Optimization Problem) representation (where every agent has a set of actions to choose from), while Bachrach et al. (2008, 2010) studied it under the skill-game representation (where every agent has a set of skills required to perform tasks). Ohta et al. (2009) studied coalition structure generation under the Marginal Contribution-net representation of Ieong and Shoham (2005), where synergies between agents are described by a (possibly small) collection of weighted logical formulas. Furthermore, Ueda et al.

(2011) and Aziz and de Keijzer (2011) studied the problem under the agent-type representation (where agents are grouped into categories, or"types"). A common denominator of all those works is that the proposed algorithms to the coalition structure generation problem capitalize heavily on features of the alternative representation under consideration. Thus, our general algorithm—ODP-IP—is unlikely to outperform those algorithms in problem instances where the alternative representation happens to compactly and efficiently represent the game. In such settings, ODP-IP can serve as a common benchmark to evaluate the potential speedups achieved by focusing on certain representations.

While this article focuses on *exact* coalition structure generation algorithms, we mention a number of *metaheuristic* algorithms, which do not guarantee that an optimal solution is ever found, nor do they provide any guarantees on the quality of their solutions. However, such algorithms can usually be applied when the number of agents is large. These include a greedy algorithm by Shehory and Kraus (1998), and genetic algorithm by Sen and Dutta (2000), a simulated-annealing algorithm by Keinänen (2009), and an algorithm by Mauro et al. (2010) that combines a greedy technique with another local-search technique.

8. Conclusions and Future Work

In recent years, the coalition structure generation problem has attracted increasing attention within the multi-agent systems community. It involves identifying an optimal way of partitioning the set of agents into exhaustive and disjoint subsets (or coalitions). In this article, we studied this combinatorial optimization problem in characteristic function games. In such settings, the problem becomes identical to the complete set partitioning problem, and to a special case of the winner determination problem.

The two state-of-the-art exact algorithms for this problem were DP by Yeh (1986) and IP by Rahwan et al. 2009, which are based on two distinct techniques used in combinatorial optimization algorithms, namely dynamic programming, and tree search, respectively. Each of these two algorithms has its advantages and drawbacks compared to the other. In particular, DP's worst-case performance is better than that of IP, since the former algorithm runs in $O(3^n)$, while the latter in $O(n^n)$ time (given n agents). On the other hand, testing the two algorithms against a variety of value-distributions shows that IP is most often significantly faster than DP. It also has the advantage of being anytime, unlike DP.

In an attempt to exploit the strengths of both approaches and, at the same time, avoid their main limitations, we modified, and improved upon, both DP and IP; this modification enabled us to combine the two into a single, hybrid algorithm called ODP-IP. Although the constituent parts of ODP-IP are based on different design paradigms, the integer-partition graph representation proposed in this article exposes the possibility of merging the two.

Our goal in this article was to try and develop an algorithm that exploits the strengths of both approaches and, at the same time, avoids their main limitations. Although DP and IP are based on different design paradigms, we developed a new search-space representation (namely the integer-partition graph) that exposes the possibility of having them combined into a single, hybrid algorithm. Building upon this, we modified, and improved upon, both DP and IP, and combined the modified versions into a new algorithm called ODP-IP. Our analysis and empirical evaluation showed that ODP-IP possesses the strengths, and avoids the weaknesses, of both DP and IP: it is anytime, runs in $O(3^n)$ time, and is faster than both algorithms for a wide variety (10 in total) of value-distributions considered in this article (with speedups reaching one or two orders of magnitude, given 25 agents).

While the focus in this article was on games where there are no influences (or externalities) between co-existing coalitions, we aim in the future to extend the underlying techniques of ODP-IP

to games with externalities, and identifying subclasses where such an extension can be efficient (in the spirit of Rahwan et al. 2012).

9. Acknowledgements

The research in this article was undertaken as part of the ORCHID Project, which is funded by EPSRC (Engineering and Physical Sciences Research Council), grant: EP/I011587/1. This article is a significantly revised and extended version of the following papers: (Rahwan and Jennings, 2008b), (Rahwan and Jennings, 2008a), and (Rahwan et al., 2012). More specifically:

- While the basic idea of ODP was presented in the short paper: (Rahwan and Jennings, 2008b), it did not include Theorems 4.1 and 4.1, which are essential for proving the correctness of ODP. Furthermore, (Rahwan and Jennings, 2008b) did not include the optimal version of DP, where *all* redundant operations are removed. Theorems 4.1 and 4.1, as well as the optimal version of DP, are presented for the first time in this current article.

- While the basic idea of combining ODP and IP was presented in: (Rahwan and Jennings, 2008a), the combination therein involved running ODP and IP in a *sequential* fashion. In more detail, ODP first computes the best partitions of all coalitions up to a certain size, $m \leq \lfloor 2n/3 \rfloor$. After that, ODP stops running, and IP starts running to build on ODP's results. The problem was that the optimal value of m was very different from one coalition-value distribution to another, and there was no way to know *a priori* how to optimally set the parameter, m. Furthermore, the worst-case complexity was greater than $O(3^n)$. Finally, there were cases where the hybrid performance was slower IP and/or ODP.

  In the current article, ODP and IP run *in parallel*, thus eliminating the need for any parameters. Furthermore, our new combination of algorithms runs in $O(3^n)$. Finally, we propose a new method to speed up IP's depth-first search (see Section 5.3), and carefully select the subspaces that must be simultaneously searched (see Appendix L). As a result, our hybrid performance is always faster than its constituent parts.

- While the idea of running ODP and IP in parallel has appeared in: (Rahwan et al., 2012), it did not include the technique in which several subspaces are searched simultaneously. Furthermore, the evaluation section was limited in that it did not show how the bound and solution quality improve over the run time of the algorithm.

References

Andrews, G. E. and Eriksson, K. 2004. *Integer Partitions*. Cambridge University Press, Cambridge, UK.

Aziz, H. and de Keijzer, B. 2011. Complexity of coalition structure generation. In *AAMAS '11: Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 191–198.

Bachrach, Y., Meir, R., Jung, K., and Kohli, P. 2010. Coalitional structure generation in skill games. In *Twenty Fourth AAAI Conference on Artificial Intelligence (AAAI)*. 703–708.

Bachrach, Y. and Rosenschein, J. S. 2008. Coalitional skill games. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*. 1023–1030.

Banerjee, B. and Kraemer, L. 2010. Coalition structure generation in multi-agent systems with mixed externalities. In *AAMAS '10: Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 175–182.

Bell, E. T. 1934. Exponential numbers. *American Mathematical Monthly 41*, 411–419.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Pr, New Jersey, USA.

Bitar, E., Baeyens, E., Khargonekar, P., Poolla, K., and Varaiya, P. 2012. Optimal sharing of quantity risk for a coalition of wind power producers facing nodal prices. In *Proceedings 31st IEEE American Control Conference*.

Chalkiadakis, G., Elkind, E., Markakis, E., Polukarov, M., and Jennings, N. R. 2010. Cooperative games with overlapping coalitions. *Journal of Artificial Intelligence Research (JAIR) 39*, 179–216.

Dang, V. D. and Jennings, N. R. 2004. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 564–571.

de Bruijn, N. G. 1981. *Asymptotic Methods in Analysis*. Dover.

Garfinkel, R. and Nemhauser, G. 1969. The set partitioning problem: Set covering problem with equality constraints. *Operations Research 17,* 5, 848–856.

Han, Z. and Poor, H. V. 2009. Coalition games with cooperative transmission: a cure for the curse of boundary nodes in selfish packet-forwarding wireless networks. *IEEE Transactions on Communications 57,* 1, 203–213.

Ieong, S. and Shoham, Y. 2005. Marginal Contribution Nets: a Compact Representation Scheme for Coalitional Games. In *ACM EC '05: 6th ACM Conference on Electronic Commerce*. 193–202.

Keinänen, H. 2009. Simulated annealing for multi-agent coalition formation. In *Proceedings of the Third KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. KES-AMSTA '09. Springer-Verlag, Berlin, Heidelberg, 30–39.

Khan, Z., Lehtomaki, J., and DaSilva, M. L.-A. L. A. 2010. On selfish and altruistic coalition formation in cognitive radio networks. In *Fifth International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM-10)*. Cannes, France.

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

Lehmann, D., Müller, R., and Sandholm, T. 2006. The winner determination problem. In *Combinatorial Auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. 297–317.

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

Lin, C. 1975. Corporate tax structures and a special class of set partitioning problems. Ph.D. thesis, Department of Operations Research, Case Western Reserve University.

Lin, C. and Salkin, H. 1979. Aggregation of subsidiary firms for minimal unemployment compensation payments via integer programming. *Management Science 25,* 4, 405–408.

Lin, C. and Salkin, H. 1983. An efficient algorithm for the complete set partitioning problem. *Discrete Applied Mathematics 6*, 149–156.

Lucas, W. and Thrall, R. 1963. n-person games in partition function form. *Naval Research Logistic Quarterly*, 281–298.

Mauro, N. D., Basile, T. M. A., Ferilli, S., and Esposito, F. 2010. Coalition structure generation with grasp. In *Proceedings of the 14th international conference on Artificial intelligence: methodology, systems, and applications*. AIMSA'10. Springer-Verlag, Berlin, Heidelberg, 111–120.

Michalak, T., Sroka, J., Rahwan, T., Wooldridge, M., McBurney, P., and Jennings, N. R. 2010. A Distributed Algorithm for Anytime Coalition Structure Generation. In *AAMAS '10: Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 1007–1014.

Myerson, R. B. 1976. Graphs and cooperation in games. Discussion Papers 246, Northwestern University, Center for Mathematical Studies in Economics and Management Science. Sept.

Ohta, N., Conitzer, V., Ichimura, R., Sakurai, Y., Iwasaki, A., and Yokoo, M. 2009. Coalition structure generation utilizing compact characteristic function representations. In *CP'09: 15th International Conference on Principles and Practice of Constraint Programming*. 623–638.

Rahwan, T. and Jennings, N. R. 2007. An algorithm for distributing coalitional value calculations among cooperative agents. *Artificial Intelligence 171,* 8–9, 535–567.

Rahwan, T. and Jennings, N. R. 2008a. Coalition structure generation: Dynamic programming meets anytime optimisation. In *AAAI'08: Twenty Third AAAI Conference on Artificial Intelligence*. 156–161.

Rahwan, T. and Jennings, N. R. 2008b. An improved dynamic programming algorithm for coalition structure generation. In *AAMAS'08: Seventh International Conference on Autonomous Agents and Multi-Agent Systems*. 1417–1420.

Rahwan, T., Michalak, T., and Jennings, N. R. 2012. A hybrid algorithm for coalition structure generation. In *Twenty Sixth Conference on Artificial Intelligence (AAAI-12)*. Toronto, Canada.

Rahwan, T., Michalak, T., Jennings, N. R., Wooldridge, M., and McBurney, P. 2009. Coalition structure generation in multi-agent systems with positive and negative externalities. In *IJCAI'09: Twenty First International Joint Conference on Artificial Intelligence*. 257–263.

Rahwan, T., Michalak, T., Wooldridge, M., and Jennings, N. R. 2012. Anytime coaliton structure generation in multi-agent systems with positive or negative externalities. *Artificial Intelligence 186*, 95–122.

Rahwan, T., Michalak, T. P., and Jennings, N. R. 2011. Minimum search to establish worst-case guarantees in coalition structure generation. In *IJCAI'11: Twenty Second International Joint Conference on Artificial Intelligence*. 338–343.

Rahwan, T., Ramchurn, S. D., Dang, V. D., and Jennings, N. R. 2007. Near-optimal anytime coalition structure generation. In *IJCAI'07: Twentieth International Joint Conference on Artificial Intelligence*. 2365–2371.

Rahwan, T., Ramchurn, S. D., Giovannucci, A., Dang, V. D., and Jennings, N. R. 2007. Anytime optimal coalition structure generation. In *AAAI'07: Twenty Second Conference on Artificial Intelligence*. 1184–1190.

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

Roman, S. 1984. *The Umbral Calculus*. Academic Press.

Rothkopf, M. H., Pekec, A., and Harstad, R. M. 1995. Computationally manageable combinatorial auctions. *Management Science 44,* 8, 1131–1147.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence 135,* 132, 1 – 54.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. 1999. Coalition structure generation with worst case guarantees. *Artificial Intelligence 111,* 1–2, 209–238.

Sen, S. and Dutta, P. 2000. Searching for optimal coalition structures. In *ICMAS'00: Sixth International Conference on Multi-Agent Systems*. 286–292.

Service, T. C. and Adams, J. A. 2010. Approximate coalition structure generation. In *AAAI*.

Service, T. C. and Adams, J. A. 2011. Constant factor approximation algorithms for coalition structure generation. *Autonomous Agents and Multi-Agent Systems 23,* 1, 1–17.

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

Ueda, S., Iwasaki, A., Yokoo, M., Silaghi, M. C., Hirayama, K., and Matsui, T. 2010. Coalition structure generation based on distributed constraint optimization. In *Twenty Fourth AAAI Conference on Artificial Intelligence (AAAI)*. 197–203.

Ueda, S., Kitaki, M., Iwasaki, A., and Yokoo, M. 2011. Concise characteristic function representations in coalitional games based on agent types. In *IJCAI'11: Twenty Second International Joint Conference on Artificial Intelligence*. 393–399.

Voice, T., Ramchurn, S. D., and Jennings, N. R. 2012. On coalition formation with sparse synergies. In *AAMAS*.

Woeginger, G. J. 2003. Combinatorial optimization - eureka, you shrink! Springer-Verlag New York, Inc., New York, NY, USA, Chapter Exact algorithms for NP-hard problems: a survey, 185–207.

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

Appendix A. Summary of Notation

A	the set of agents
a_i	an agent in A
n	the number of agents in A
$\mathcal{C}^A$	set of all coalitions over A
$\mathcal{C}^A_s$	set of all coalitions over A that are of size s each
C	a coalition
CS	a coalition structure
CS^*	an optimal coalition structure
CS^{**}	the best coalition structure found at any point in time (i.e., the current best solution found so far)
β	the established bound on the quality of CS^{**}, where: $\frac{V(CS^*)}{V(CS^{**})} \leq \beta$
$\mathcal{I}^n$	the set of all integer partitions of n
I	an integer partition, i.e., a multiset of integers
Π^A	the set of all coalition structures over A
Π^A_s	the set of all coalition structures over A that are of size s
Π^A_I	the set of all coalition structures over A in which the coalition sizes match the parts in integer partition I
Π^C	the set of all partitions of C
Π^C_s	the set of all partitions of C that are of size s
Π^C_I	the set of all partitions of C in which the coalition sizes match the parts in integer partition I
Π	the set of all possible partitions, i.e., $\Pi = \cup_{C \subseteq A} \Pi^C$
π	a partition, i.e., a disjoint set of coalitions where every agent appears *at most* once
$V(CS)$	the value of the coalition structure CS
$V(\pi)$	the value of the partition π
$v(C)$	the value of the coalition C
Max_s	the maximum value of all coalitions of size s
Avg_s	the average value of all coalitions of size s
UB^*	upper bound on the value of CS^*
UB_I	upper bound on the value of the best CS in Π^A_I
LB^*	lower bound on the value of CS^*
LB_I	lower bound on the value of the best CS in Π^A_I
$\mathcal{M}$	the set of all possible movements in the coalition structure graph
M	a subset of $\mathcal{M}$
M^*	the subset of $\mathcal{M}$ that is evaluated by ODP
M^{**}	the subset of $\mathcal{M}$ that is evaluated by the size-based version of ODP
$M^{s',s''}$	the subset of $\mathcal{M}$ in which every movement corresponds to splitting a coalition of size $(s' + s'')$ into two coalitions of sizes s' and s''
$m^{C',C''}$	the movement that corresponds to splitting $C = C_1 \cup C_2$ into C_1 and C_2
R^π_M	the set of all partitions that are reachable from π via M
$f(C)$	the value of an optimal partition of C
$f_M(C)$	the value of the partition with the highest value in $R^{\{C\}}_M$

Appendix B. Proof of Theorem 3

Theorem 3 *Given an arbitrary partition, $\{C_1, \ldots, C_k\} \in \Pi$, and an arbitrary subset of movements, $M \subseteq \mathcal{M}$, the following holds:*

$$R_M^{\{C_1,\ldots,C_k\}} \;=\; R_M^{\{C_1\}} \;\times\; \ldots \;\times\; R_M^{\{C_k\}}$$

Proof. From Definition 4, it is easy to check that the following holds for any three disjoint partitions, $\pi_a, \pi_b, \pi_c \in \Pi$:

$$\pi_a \overset{M}{\rightsquigarrow} \pi_b \quad \Rightarrow \quad \pi_c \cup \pi_a \overset{M}{\rightsquigarrow} \pi_b \cup \pi_c \tag{B.1}$$

$$\left(\; \pi_a \overset{M}{\rightsquigarrow} \pi_b \;\right) \;\wedge\; \left(\; \pi_b \overset{M}{\rightsquigarrow} \pi_c \;\right) \quad \Rightarrow \quad \pi_a \overset{M}{\rightsquigarrow} \pi_c \tag{B.2}$$

Now, to prove Theorem 3, it is sufficient to prove that the following holds for any arbitrary partition $\pi \in \Pi$:

$$\pi \in R_M^{\{C_1\}} \;\times\; \ldots \;\times\; R_M^{\{C_k\}} \quad \Rightarrow \quad \pi \in R_M^{\{C_1,\ldots,C_k\}} \tag{B.3}$$

$$\pi \in R_M^{\{C_1,\ldots,C_k\}} \quad \Rightarrow \quad \pi \in R_M^{\{C_1\}} \;\times\; \ldots \;\times\; R_M^{\{C_k\}} \tag{B.4}$$

We will start by proving that (B.3) holds. To this end, if $\pi \in R_M^{\{C_1\}} \;\times\; \ldots \;\times\; R_M^{\{C_k\}}$, then surely there exists k partitions, $\pi_1 \in R_M^{\{C_1\}}, \ldots, \pi_k \in R_M^{\{C_k\}}$, such that:

$$\pi \;=\; \pi_1 \cup \cdots \cup \pi_k \tag{B.5}$$

Moreover, since $\pi_i \in R_M^{\{C_i\}}$ for all $i \in \{1, \ldots, k\}$, then by definition we have:

$$\begin{array}{lcl}
\{C_1\} & \overset{M}{\rightsquigarrow} & \pi_1 \\
\{C_2\} & \overset{M}{\rightsquigarrow} & \pi_2 \\
\{C_3\} & \overset{M}{\rightsquigarrow} & \pi_3 \\
 & \vdots & \\
\{C_k\} & \overset{M}{\rightsquigarrow} & \pi_k
\end{array}$$

Based on this, as well as (B.1), we find that:

$$\begin{array}{rclcl}
\big(\{C_2\} \cup \cdots \cup \{C_k\}\big) & \cup & \{C_1\} & \overset{M}{\rightsquigarrow} & \pi_1 \;\cup\; \big(\{C_2\} \cup \cdots \cup \{C_k\}\big) \\
\big(\pi_1 \cup \{C_3\} \cup \cdots \cup \{C_k\}\big) & \cup & \{C_2\} & \overset{M}{\rightsquigarrow} & \pi_2 \;\cup\; \big(\pi_1 \cup \{C_3\} \cup \cdots \cup \{C_k\}\big) \\
\big(\pi_1 \cup \pi_2 \cup \{C_4\} \cup \cdots \cup \{C_k\}\big) & \cup & \{C_3\} & \overset{M}{\rightsquigarrow} & \pi_3 \;\cup\; \big(\pi_1 \cup \pi_2 \cup \{C_4\} \cup \cdots \cup \{C_k\}\big) \\
 & & \vdots & & \\
\big(\pi_1 \cup \cdots \cup \pi_{k-1}\big) & \cup & \{C_k\} & \overset{M}{\rightsquigarrow} & \pi_k \;\cup\; \big(\pi_1 \cup \cdots \cup \pi_{k-1}\big)
\end{array}$$

This can be written differently as follows:

$$\begin{aligned}
\{C_1\} \cup \cdots \cup \{C_k\} &\overset{M}{\rightsquigarrow} \pi_1 \cup \{C_2\} \cup \cdots \cup \{C_k\} \\
\pi_1 \cup \{C_2\} \cup \cdots \cup \{C_k\} &\overset{M}{\rightsquigarrow} \pi_1 \cup \pi_2 \cup \{C_3\} \cup \cdots \cup \{C_k\} \\
\pi_1 \cup \pi_2 \cup \{C_3\} \cup \cdots \cup \{C_k\} &\overset{M}{\rightsquigarrow} \pi_1 \cup \pi_2 \cup \pi_3 \cup \{C_4\} \cup \cdots \cup \{C_k\} \\
&\vdots \\
\pi_1 \cup \cdots \cup \pi_{k-1} \cup \{C_k\} &\overset{M}{\rightsquigarrow} \pi_1 \cup \cdots \cup \pi_k
\end{aligned}$$

From this, as well as (B.2), we find that:

$$\{C_1\} \cup \cdots \cup \{C_k\} \overset{M}{\rightsquigarrow} \pi_1 \cup \cdots \cup \pi_k$$

This can be written differently based on (B.5) as follows:

$$\{C_1, \ldots, C_k\} \overset{M}{\rightsquigarrow} \pi$$

In other words, $\pi \in R_M^{\{C_1,\ldots,C_k\}}$. This concludes our proof of (B.3). It remains to prove that (B.4) holds. To this end, observe that if $\pi \in R_M^{\{C_1,\ldots,C_k\}}$, then this means that π can be broken into k disjoint partitions, $\pi_1, \ldots, \pi_k$, such that $\pi_1 \cup \cdots \cup \pi_k = \pi$, and $\cup \pi_i = C_i$ for all $i \in \{1, \ldots, k\}$. This is simply because π is, by definition, the result of splitting (some of) the coalitions in $\{C_1, \ldots, C_k\}$ into smaller coalitions. Now, to complete the proof, it is sufficient to show that:

$$\forall i \in \{1, \ldots, n\}, \pi_i \in R_M^{\{C_i\}}$$

The proof follows directly from the fact that $\pi_1 \cup \cdots \cup \pi_k$ is reachable from $\{C_1, \ldots, C_k\}$ via M, which means that for every $i \in \{1, \ldots, n\}$ we must have one of two possibilities: either $\pi_i = \{C_i\}$, or there exist movements in M by which the agents in C_i can be (repeatedly) split into smaller coalitions until they become partitioned as in π_i. □

Appendix C. Proof of Theorem 4

Theorem 4 *For any coalition $C \subseteq A$, and for any subset of movements, $M \subseteq \mathcal{M}$, the following holds:*

$$f_M(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\left\{v(C)\ ,\ \max_{\{C',C''\} \in R_M^{\{C\}}} \left(f_M(C') + f_M(C'')\right)\right\} & \text{otherwise.} \end{cases} \tag{6}$$

Proof. If $|C| = 1$, then no movement can be made from $\{C\}$ (because C cannot be split into two coalitions). This means that $R_M^{\{C\}} = \{\{C\}\}$ which, in turn, means that $f_M(C) = v(C)$. It remains to prove that equation (6) holds for the case where $|C| > 1$.

Since $\{C\}$ contains exactly one coalition (which is C), then every partition reachable from $\{C\}$ via a *single movement* in M contains *exactly two* coalition and vice versa; every partition in $R_M^{\{C\}}$ that contains *exactly two* coalitions must be reachable from $\{C\}$ via a *single movement* in M. This implies that:

$$\left\{\pi' \in \Pi : \{C\} \xrightarrow{M} \pi'\right\} \quad = \quad \left\{\{C', C''\} \in R_M^{\{C\}}\right\} \tag{C.1}$$

Thus, we have:

$$\begin{aligned} R_M^{\{C\}} &= \{\{C\}\} \cup \bigcup_{\pi' \in \Pi : \{C\} \xrightarrow{M} \pi'} R_M^{\pi'} \qquad \text{(based on (5))} \\ &= \{\{C\}\} \cup \bigcup_{\{C', C''\} \in R_M^{\{C\}}} R_M^{\{C', C''\}} \qquad \text{(based on (C.1))} \\ &= \{\{C\}\} \cup \bigcup_{\{C', C''\} \in R_M^{\{C\}}} \left(R_M^{\{C'\}} \times R_M^{\{C''\}} \right) \qquad \text{(based on Theorem 3)} \end{aligned}$$

Based on this, we have:

$$\begin{aligned} f_M(C) &= \max\nolimits_{\pi \in R_M^{\{C\}}} V(\pi) \\ &= \max \left\{ v(C) \ , \ \max\nolimits_{\{C', C''\} \in R_M^{\{C\}}} \left(\ \max\nolimits_{\pi \in \left(R_M^{\{C'\}} \times R_M^{\{C''\}} \right)} V(\pi) \ \right) \right\} \\ &= \max \left\{ v(C) \ , \ \max\nolimits_{\{C', C''\} \in R_M^{\{C\}}} \left(\ \max\nolimits_{\pi \in R_M^{\{C'\}}} V(\pi) + \max\nolimits_{\pi \in R_M^{\{C'\}}} V(\pi) \ \right) \right\} \\ &= \max \left\{ v(C) \ , \ \max\nolimits_{\{C', C''\} \in R_M^{\{C\}}} \left(f_M(C') + f_M(C'') \right) \right\} \end{aligned}$$

□

Appendix D. Proof of Theorem 5

Theorem 5 *For any two coalitions $C', C'' \in \mathcal{C}^A$, let us write $C' < C''$ if and only if C' precedes C'' lexicographically. Now, if we define $M^* \subseteq \mathcal{M}$ as:*

$$M^* = \left\{ m^{C', C''} \in \mathcal{M} : \left(C' \cup C'' = A \right) \wedge \left(C' < C'' < A \setminus (C' \cup C'') \right) \right\} \tag{7}$$

Then, the following holds:

$$R_{M^*}^{\{A\}} = \Pi^A \tag{8}$$

Proof. To prove (8), it suffices to prove that every coalition structure $CS = \{C_1, \ldots, C_k\} : k \geq 2$ is reachable from some other coalition structure $CS' : |CS'| = k - 1$ via some movement in M^*. To this end, without loss of generality, assume that $C_1 < \cdots < C_k$. Then, to prove (8), it suffices to show that CS is reachable from the coalition structure $(CS \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$ via M^*, i.e., it suffices to show that $m^{C_1, C_2} \in M^*$.

First, let us consider the case where $k = 2$. In this case, we have $CS = \{C_1, C_2\}$, and so $C_1 \cup C_2 = A$. This means that $m^{C_1, C_2} \in M^*$ (see equation (7)).

Now, let us consider the case where $k > 2$. Here, since $C_1 < \cdots < C_k$, then the assumed lexicographic order implies that: $C_1 < C_2 < (C_3 \cup \cdots \cup C_k)$, which can be written as $C_1 < C_2 < A \setminus (C \cup C')$. This means that $m^{C_1, C_2} \in M^*$ (again see equation (7)). □

Appendix E. Proof of Theorem 6

Theorem 6 *For any coalition $C \in \mathcal{C}^A$, if $\{a_1, a_2\} \not\subseteq C$, then ODP does not evaluate any of the possible ways of splitting C.*

Proof. For any coalition $C \in \mathcal{C}^A : \{a_1, a_2\} \not\subseteq C$, we will prove that:

$$\forall m^{C',C''} \in \mathcal{M} : C' \cup C'' = C, m^{C',C''} \notin M^*$$

We will deal with each of the following complementary cases separately:

- **Case 1:** $a_1 \notin C$. This means that $a_1 \in A \setminus (C' \cup C'')$. Therefore, we have: $A \setminus (C' \cup C'') < C'$ and $C' \cup C'' \neq A$. As such, $m^{C',C''} \notin M^*$ according to (7).
- **Case 2:** $a_1 \in C$ and $a_2 \notin C$. Here, surely one of the two coalitions in $\{C', C''\}$ does not contain a_1 nor a_2. Let this coalition be C''. Now since $a_2 \in A \setminus (C' \cup C'')$, then we have: $A \setminus (C' \cup C'') < C''$ and $C' \cup C'' \neq A$. This implies that $m^{C',C''} \notin M^*$ according to (7).

□

Appendix F. Proof of Theorem 7

Theorem 7 *It is not possible to evaluate fewer splits compared to ODP, and still be guaranteed to find an optimal coalition structure.*

Proof. We will prove that, for every $CS = \{C_1, \ldots, C_k\} : k \geq 2$, the ODP algorithm evaluates *exactly* one movement that leads to CS (from another coalition structure containing $|CS| - 1$ coalitions). Without loss of generality, we will assume that $C_1 < \cdots < C_k$. In our proof, we will distinguish between the following two cases:

- **Case 1:** $k = 2$. In this case, there is exactly one possible movement that leads to CS, which is m^{C_1,C_2}. Since $C_1 \cup C_2 = A$, then $m^{C_1,C_2} \in M^*$ according to (7).
- **Case 2:** $k > 2$. Here, since $C_1 < C_2 < A \setminus (C_1 \cup C_2)$, then $m^{C_1,C_2} \in M^*$ according to (7). It remains to show that no other movement in M^* leads to CS. To this end, a movement $m^{C_i,C_j} \in \mathcal{M}$ leads to CS if and only if $C_i, C_j \in CS$. Next, for any such movement, we will show that if $\{i, j\} \neq \{1, 2\}$, then $m^{C_i,C_j} \notin M^*$. Here,
  - if $1 \notin \{i, j\}$, then $C_1 \subseteq A \setminus (C_i \cup C_j)$ and so: $A \setminus (C_i \cup C_j) < C_i$. Therefore, $m^{C_i,C_j} \notin M^*$.
  - if $1 \in \{i, j\}$ and $2 \notin \{i, j\}$, then either $C_2 < C_i$ (in which case $A \setminus (C_i \cup C_j) < C_i$) or $C_2 < C_j$ (in which case $A \setminus (C_i \cup C_j) < C_j$). In either case, $m^{C_i,C_j} \notin M^*$.

We have shown that, for each $CS = \{C_1, \ldots, C_k\} : k \geq 2$, ODP evaluates exactly one of the movements that lead to CS, namely m^{C_1,C_2}. Furthermore, we know that ODP does not evaluate a movement more than once. Therefore, avoiding the evaluation of even a single movement in M^* leaves at least one coalition structure with no movements leading to it. □

Appendix G. Proof of Theorem 8

Theorem 8 *The number of movements in M^* is equal to the number of coalition structures in $\Pi_2^A \cup \Pi_3^A$—levels 2 and 3 of the coalition structure graph. That is:*

$$\left|M^*\right| \quad = \quad \left|\Pi_2^A\right| + \left|\Pi_3^A\right|$$

Proof. For any coalition structure $CS = \{C_1, \ldots, C_k\} : k > 1$, we will assume without loss

of generality that $C_1 < \cdots < C_k$. Earlier in our proof of Theorem 7, we showed that there is exactly one movement in M^* that leads to CS; this movement is m^{C_1,C_2}. Now, to prove Theorem 8, it is sufficient to observe the following:

- Every $m^{C,C'} \in M^* : C \cup C' = A$ leads to exactly one coalition structure, namely $\{C, C'\}$, which is in Π_2^A. Similarly, every $\{C_1, C_2\} \in \Pi_2^A$ is reachable via exactly one movement in M^*, namely m^{C_1,C_2}. As such, there is a one-to-one correspondence between Π_2^A and $\{m^{C,C'} \in M^* : C \cup C' = A\}$. Therefore,

$$\left|\{m^{C,C'} \in M^* : C \cup C' = A\}\right| \quad = \quad \left|\Pi_2^A\right|$$

- Every $m^{C,C'} \in M^* : (C \cup C') \subset A$ leads to exactly one coalition structure in Π_3^A, namely $\{C, C', A \setminus (C \cup C')\}$ (this is because no other coalition structure in Π_3^A contains both C and C'). Similarly, every $\{C_1, C_2, C_3\} \in \Pi_3^A$ is reachable via exactly one movement in M^*, namely m^{C_1,C_2}. This means there is a one-to-one correspondence between Π_3^A and $\{m^{C,C'} \in M^* : (C \cup C') \subset A\}$, and so:

$$\left|\{m^{C,C'} \in M^* : (C \cup C') \subset A\}\right| \quad = \quad \left|\Pi_3^A\right| \tag{G.1}$$

□

Appendix H. Proof of Theorem 9

Theorem 9 *Given n agents, ODP runs in $O(3^n)$ time.*

Proof. Clearly, the main operations in ODP are those that involve the evaluation of different movements. Now based on Theorem 8, the number of such operations is equal to $|\Pi_2^A| + |\Pi_3^A|$. To compute this number, recall that the number of ways to partition n elements into k parts—widely known as *the Stirling number of the Second Kind* (Roman, 1984), and denoted as $S(n, k)$—is computed as follows:

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i\binom{k}{i}(k-i)^n$$

Thus, the number of movements that are evaluated by ODP is: $S(n, 2) + S(n, 3)$, which equals:

$$\frac{1}{2}\left(3^{n-1} - 1\right)$$

□

Appendix I. Proof of Theorem 10

Theorem 10 *For any two positive integers, $s', s'' \in \mathbb{Z}^+$, let us define $M^{s',s''} \subseteq \mathcal{M}$ as the set that consists of every movement in which a coalition of size $(s' + s'')$ is split into two coalitions of sizes s' and s''. More formally, $M^{s',s''} = \{m^{C',C''} \in \mathcal{M} : |C'| = s', |C''| = s''\}$. Furthermore, let us define $M^{**} \subseteq \mathcal{M}$ as follows:*

$$M^{**} = \left(\bigcup_{s',s'' \in \mathbb{Z}^+ : \max\{s',s''\} \leq n-s'-s''} M^{s',s''}\right) \cup \left(\bigcup_{s',s'' \in \mathbb{Z}^+ : s'+s''=n} M^{s',s''}\right) \tag{9}$$

Then, the following holds:

$$R^{\{A\}}_{M^{**}} = \Pi^A \tag{10}$$

Proof. To prove (10), it is sufficient to prove that every coalition structure $CS : |CS| \geq 2$ is reachable via M^{**} from some other coalition structure $CS' : |CS'| = |CS| - 1$. To this end, let us assume, without loss of generality, that $CS = \{C_1, \ldots, C_k\}$, where $k \geq 2$ and $|C_1| \leq \cdots \leq |C_k|$. Then, to prove (10), it is sufficient to show that CS is reachable from the coalition structure $(CS \backslash \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$ via M^{**}. We will do this by showing that $m^{C_1,C_2} \in M^{**}$.

First, let us consider the case where $k = 2$. In this case, we have $CS = \{C_1, C_2\}$, and so $|C_1| + |C_2| = n$. This means that m^{C_1,C_2} is added to M^{**} by the term $\bigcup_{s',s'' \in \mathbb{Z}^+ : s'+s''=n} M^{s',s''}$ (see equation (9)).

Now, let us consider the case where $k > 2$. Here, since $|C_1| \leq |C_2|$, then $\max\{|C_1|, |C_2|\} = |C_2|$. Furthermore, since C_1 and C_2 are the smallest two coalitions in CS, the number of agents in C_2 must be smaller than, or equal to, the number of agents in $C_3 \cup \cdots \cup C_k$. In other words, $|C_2| \leq n - |C_1| - |C_2|$. This means m^{C_1,C_2} is added to M^{**} by the term $\bigcup_{s',s'' \in \mathbb{Z}^+ : \max\{s',s''\} \leq n-s'-s''} M^{s',s''}$ (again see equation (9)). □

Appendix J. Proof of Theorem 11

Theorem 11 *The size-based version of ODP (i.e., the one that evaluates M^{**}) does not evaluate any of the possible ways of splitting a coalition of size $s \in \{\lfloor \frac{2\times n}{3} \rfloor + 1, \ldots, n-1\}$.*

Proof. For every $s \in \{\lfloor \frac{2\times n}{3} \rfloor + 1, \ldots, n-1\}$, we need to prove that the following holds:

$$\forall s', s'' \in \mathbb{Z}^+ : (s' + s'') = s, (M^{s',s''} \cap M^{**}) = \emptyset \tag{J.1}$$

Since $s' + s'' \neq n$, then based on (9)—the equation that defines M^{**}—we can prove (J.1) by proving that:

$$\max\{s', s''\} > n - s' - s'' \tag{J.2}$$

Since $s' + s'' = s$ and $\max\{s', s''\} \geq \lceil s/2 \rceil$, then to prove (J.2) it is sufficient to prove that:

$$\lceil s/2 \rceil > n - s$$

To prove that this inequality holds for all possible values of s, it is sufficient to prove that it holds for the smallest possible value of s, which is $\lfloor \frac{2\times n}{3} \rfloor + 1$. Thus, all we need is to prove that:

$$\left\lceil \frac{\left(\lfloor \frac{2\times n}{3} \rfloor + 1\right)}{2} \right\rceil > n - \left(\left\lfloor \frac{2 \times n}{3} \right\rfloor + 1 \right) \tag{J.3}$$

This can be proved by mathematical induction. First we will prove that, if (J.3) holds for n, then it is also holds for $(n+3)$. In other words, assuming that (J.3) holds for n, we will prove that:

$$\left\lceil \frac{\left(\left\lfloor \frac{2\times(n+3)}{3} \right\rfloor + 1 \right)}{2} \right\rceil > (n+3) - \left(\left\lfloor \frac{2 \times (n+3)}{3} \right\rfloor + 1 \right)$$

This can be done as follows:

$$
\begin{aligned}
\left\lceil \frac{\left(\left\lfloor \frac{2\times(n+3)}{3} \right\rfloor + 1\right)}{2} \right\rceil &= \left\lceil \frac{\left(\left\lfloor \frac{2\times n}{3} + 2 \right\rfloor + 1\right)}{2} \right\rceil \\
&= \left\lceil \frac{\left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 2 + 1\right)}{2} \right\rceil \\
&= \left\lceil \frac{\left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right)}{2} \right\rceil + 1 \\
&> n - \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right) + 1 \quad \text{(because we assume that (J.3) holds for } n\text{)} \\
&= n - \left\lfloor \frac{2\times n}{3} \right\rfloor \\
&= (n+3) - \left(\left\lfloor \frac{2\times n}{3} + 2 \right\rfloor + 1\right) \\
&= (n+3) - \left(\left\lfloor \frac{2\times(n+3)}{3} \right\rfloor + 1\right)
\end{aligned}
$$

We have shown that if (J.3) holds for n, then it also holds for $n+3$. It remains to show is that (J.3) holds for $n = 3$, $n = 4$, and $n = 5$:

- For $n = 3$, $\left\lceil \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right)/2 \right\rceil = 2$ and $n - \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right) = 0$
- For $n = 4$, $\left\lceil \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right)/2 \right\rceil = 2$ and $n - \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right) = 1$
- For $n = 5$, $\left\lceil \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right)/2 \right\rceil = 2$ and $n - \left(\left\lfloor \frac{2\times n}{3} \right\rfloor + 1\right) = 1$

□

Appendix K. Proof of Theorem 12

Theorem 12 *Given n agents, ODP-IP runs in $O(3^n)$ time.*

Proof. First, let us compute the number of movements that are evaluated by DP. Since DP evaluates all the possible ways of splitting every $C \subseteq A$ in two, the total number of evaluations is:

$$\sum_{s=1}^{n} \binom{n}{s} 2^{s-1}$$

Furthermore, it is known that:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} \tag{K.1}$$

By setting $a = 2$ and $b = 1$ in equation (K.1), we find that:

$$\frac{1}{2}\left(3^n - 1\right) = \sum_{s=1}^{n} \binom{n}{s} 2^{s-1}$$

Thus, the number of movements that are evaluated by DP is $1/2(3^n - 1)$. Furthermore, we know that the number of movements evaluated by the optimal version of ODP is $\frac{1}{2}\left(3^{n-1} - 1\right)$ (see Appendix

H). Finally, since the number of movements that are evaluated by the size-based version of ODP is more than that of ODP, and less than that of DP, this number is in:

$$\left[\frac{1}{2}\left(3^{n-1}-1\right)\ ,\ \ \frac{1}{2}\left(3^{n}-1\right)\right]$$

Thus, the size-based version of ODP runs in $O(3^n)$. This algorithm runs in parallel with IP in ODP-IP. Based on this, ODP-IP is guaranteed to terminate in $O(3^n)$ time. □

Appendix L. Analyzing the Different Methods of Searching Multiple Subspaces Simultaneously

This appendix provides further details on how IP can simultaneously search multiple subspaces using the information provided by ODP. To this end, assume that ODP has already finished evaluating $m^{C,C'} \in M^{**} : (|C| + |C'|) \in \{2, \ldots, s^*\}$. Then, for any given subspace, $\Pi^A_I : I = \{i_1, \ldots, i_{|I|}\}$, we modify IP such that, instead of searching for a coalition structure in $\arg\max_{CS \in \Pi^A_I} V(CS)$, it performs the following steps:

1. **Identify $\mathcal{X}^*$—the set of integer partitions whose subspaces have not yet been searched, and are reachable from Π^A_I** using only the movements that have been evaluated by ODP so far. For instance, given $I = \{2, 4, 4\}$ and $s^* = 4$, the set $\mathcal{X}^*$ consists of all the integer partitions that are reachable through the dotted edges in Figure 7(B).

2. **Identify I^*—the set of integer(s) in I that will be split** in order to reach (some of) the subspaces in $\mathcal{X}^*$. As mentioned earlier in Section 5.4, one can choose to either split a single integer in I, or split as many integers as possible. We will consider both cases. Now if exactly one integer will be split, then put in I^* the integer of which the splitting allows for reaching *the largest number of integer partitions* in $\mathcal{X}^*$. On the other hand, if multiple integers will be split, then put in I^* the integers of which the splitting allows for reaching *all the integer partitions* in $\mathcal{X}^*$. **The subset of $\mathcal{X}^*$ that is reachable by splitting the integer(s) in I^* will be denoted by $\mathcal{Y}^*$.** For instance, given $I = \{2, 4, 4\}$ and $s^* = 4$, if exactly one integer will be split, then we have $I^* = \{4\}$, and $\mathcal{Y}^*$ consists of the integer partitions that are reachable through the dashed edges in Figure 7(A). Otherwise, if multiple integers will be split, then we have $I^* = \{2, 4, 4\}$, and $\mathcal{Y}^*$ consists of the integer partitions that are reachable through the dotted edges in Figure 7(B).

3. **Change the order of the integers in I and in every $I' \in \mathcal{Y}^*$.** To this end, let i^*_j denote the j^{th} element in I^*. Furthermore, for every $i^*_j \in I^*$, and every $I' \in \mathcal{Y}^*$, let $S(I', i^*_j)$ be the subset of I' that results from splitting i^*_j. Now, order the integers in I by putting the ones in $I \setminus I^*$ first, then putting i^*_1, then i^*_2, and so on until $i^*_{|I^*|}$. Similarly, for every $I' \in \mathcal{Y}^*$, change the order in I' by putting the ones in $I' \setminus I^*$ first, then those in $S(I', i^*_1)$, then those in $S(I', i^*_2)$, and so on until $S(I', i^*_{|I^*|})$.

4. **Search Π^A_I, where every $\{C_1, \ldots, C_{|I|}\} \in \Pi^A_I$ is evaluated as follows:**

$$\sum_{j=1}^{|I \setminus I^*|} v(C_j) \ + \sum_{j=|I \setminus I^*|+1}^{|I|} f_{M^{**}}(C_j) \tag{L.1}$$

During this search, at every depth, d, **use the following, modified, branch-and-bound inequality**:

$$\sum_{j=1}^{\min(d,|I\setminus I^*|)} v(C_j) \;+\; \sum_{j=\min(d,|I\setminus I^*|)+1}^{d} f_{M^{**}}(C_j) \;+\; UB_I^d \;<\; V(CS^{**}) \qquad \text{(L.2)}$$

where UB_I^d is an upper bound computed as follows:

$$UB_I^d = \max\left(\sum_{j=d+1}^{|I|} Max_{i_j^*}\;,\;\; \max_{I'\in\mathcal{Y}^*}\sum_{j=d+1}^{|I|}\sum_{s\in S(I',i_j^*)} Max_s\right)$$

The result of this search is a coalition structure $\{C_1^*,\ldots,C_{|I|}^*\}\in\Pi_I^A$ that maximizes (L.1).

5. **Replace every** $C_j^* : j > |I\setminus I^*|$ **with** `getBestPartition`$(C_j^*, t(C_j^*))$. The result is a coalition structure in: $\arg\max_{CS\in\left(\{\Pi_I^A\}\cup\mathcal{Y}^*\right)} V(CS)$.

At first glance, it may seem that partitioning multiple integers is better than partitioning only one, because more subspaces can be searched simultaneously. Surprisingly, however, we will show why it can actually be faster to partition only one integer. As mentioned earlier, when IP searches a subspace, Π_I^A, several other subspaces can be searched simultaneously as long as: (1) those subspaces have not yet been searched, and (2) the integer partitions that represent those subspaces are reachable from I using the movements that ODP has evaluated thus far. Recall that $I^*\subseteq I$ denotes the integers that will be split in order to reach other subspaces, and $\mathcal{Y}^*$ denotes the set of integer partitions representing those subspaces. Based on this, the more integers we put in I^*, the more subspaces we have in $\mathcal{Y}^*$ (an example is illustrated in Figure 7). However, we will show why it is actually faster to always put only one integer in I^*, even if it meant fewer subspaces will be in $\mathcal{Y}^*$ (i.e., few subspaces will be searched simultaneously with Π_I^A). To this end, recall that the branch-and-bound technique usually involves checking whether the inequality in (1) holds. For convenience, we re-write the inequality below:

$$\sum_{j=1}^{d} v(C_j) + \sum_{j=d+1}^{|I|} Max_{i_j} < V(CS^{**}) \qquad (1)$$

However, when searching multiple subspaces simultaneously, we use the inequality in (L.2) instead of the one in (1). Note that (L.2) will hold fewer times compared to (1), because the left-hand side in the former inequality is greater than that in the latter one. This increase (in the left-hand side) is in fact the price that must be paid in order to avoid searching every $\Pi_{I'}^A : I'\in\mathcal{Y}^*$ separately later on. The problem, however, is that this price is often greater than what is necessary. To see why this is the case, let us analyse the two modifications that are behind this increase:

- The first modification is when $|I\setminus I^*| < d$. In this case, every $C_j : j\in\{|I\setminus I^*|,\ldots,d\}$ is evaluated as $f_{M^*}(C_j)$ instead of $v(C_j)$.
- The second modification is in the upper bound on the values of the coalitions that will be added to $C_1,\ldots,C_d$. In particular, since every $\Pi_{I'}^A : I'\in\mathcal{Y}^*$ is being searched simultaneously with Π_I^A, the upper bound in (L.2) becomes UB_I^d, instead of the one used in (1), which is $\sum_{j=d+1}^{|I|} Max_{i_j}$.

A key point here is that $\mathcal{Y}^*$ does not contain all the integer partitions that are reachable from I; it only contains those representing subspaces *that have not yet been searched*. This important point is reflected in the second modification, but not in the first one. More specifically, in the second modification, a new upper bound is used that *only takes into account* Π_I^A *as well as* $\Pi_{I'}^A : I' \in \mathcal{Y}^*$. However, in the first modification, every $C_j : j \in \{|I \setminus I^*|, \ldots, d\}$ is evaluated as $f_{M^*}(C_j)$—the value of the best partition of C_j in all the subspaces that are reachable from Π_I^A, *including those that have already been searched*. In other words, this modification ignores the fact that certain subspaces have already been searched.

Now, let us analyse the case where I^* contains exactly one integer. To this end, observe that the branch-and-bound technique is generally used only when $d < |I| - 1$. This is because, if $d = |I| - 1$, then there is no need to determine whether $\{C_1, \ldots, C_d\}$ is promising. Instead, one can straight away construct the only coalition structure contain $C_1, \ldots, C_d$, and that is by putting all remaining agents in a coalition of their own. Based on this observation, whenever the branch-and-bound technique is used, we always have $d < |I| - 1$. This implies that, when I^* contains exactly one integer, we always have: $\min(d, |I \setminus I^*|) = d$. Consequently, the inequality in (L.2) can be written as follows:

$$\sum_{j=1}^{d} v(C_j) \;+\; UB_I^d \;\;<\;\; V(CS^{**})$$

This way, we get rid of the first modification, and only keep the second one, which takes into consideration only the subspaces *that have not yet been searched*, and are reachable from Π_I^A.