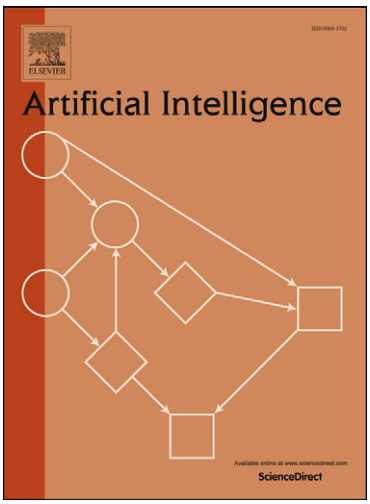
# Accepted Manuscript




A hybrid exact algorithm for complete set partitioning

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

PII: S0004-3702(15)00142-3
DOI: http://dx.doi.org/10.1016/j.artint.2015.09.006
Reference: ARTINT 2891

To appear in: *Artificial Intelligence*

Received date: 20 February 2015
Revised date: 14 August 2015
Accepted date: 18 September 2015

Please cite this article in press as: T. Michalak et al., A hybrid exact algorithm for complete set partitioning, *Artificial Intelligence* (2015), http://dx.doi.org/10.1016/j.artint.2015.09.006

# A Hybrid Exact Algorithm for Complete Set Partitioning

Tomasz Michalak[1,2,†], Talal Rahwan[3,†], Edith Elkind[1], Michael Wooldridge[1] and Nicholas R. Jennings[4]

[1]*Department of Computer Science, University of Oxford, UK.*
[2]*Institute of Informatics, University of Warsaw, Poland.*
[3]*Masdar Institute of Science and Technology, UAE.*
[4]*School of Electronics and Computer Science, University of Southampton, UK.*

**Abstract**

In the *Complete Set Partitioning* problem we are given a finite set of elements where every subset is associated with a value, and the goal is to partition this set into disjoint subsets so as to maximise the sum of subset values. This abstract problem captures the *Coalition Structure Generation* problem in cooperative games in characteristic function form, where each subset, or coalition, of agents can make a profit when working together, and the goal is to partition the set of agents into coalitions to maximise the total profit. It also captures the special case of the *Winner Determination* problem in combinatorial auctions, where bidders place bids on every possible bundle of goods, and the goal is to find an allocation of goods to bidders that maximises the profit of the auctioneer.

The main contribution of this article is an extensive theoretical analysis of the search space of the Complete Set Partitioning problem, which reveals that two fundamentally different exact algorithms can be significantly improved upon in terms of actual runtime. These are (1) a dynamic programming algorithm called "DP" [48, 36] and (2) a tree-search algorithm called "IP" [32]. We start by drawing a link between DP and a certain graph describing the structure of the search space. This link reveals that many of DP's operations are in fact redundant. Consequently, we develop ODP—an optimal version of DP that avoids *all* of its redundant operations. Since ODP and IP are based on different design paradigms, each has its own strengths and weaknesses compared to the other. Thus, one has to trade off the advantages of one algorithm for the advantages of the other. This raises the following question: Is this trade-off inevitable? To answer this question, we develop a new representation of the search space, which links both algorithms, and allows for contrasting the workings of the two. This reveals that ODP and IP can actually be combined, leading to the development of ODP-IP—a hybrid algorithm that avoids the limitations of its constituent parts, while retaining and significantly improving upon the advantages of each part.

We benchmark our algorithm against that of Björklund et al. [SIAM Journal of Computing, 2009], which runs in $O(2^n)$ time given $n$ agents. We observe that the algorithm of Björklund et al. relies on performing arithmetic operations with very large integers, and assumes that any such operation has unit cost. In practice, however, working with large integers on a modern PC is costly. Consequently, when implemented, our $O(3^n)$ algorithm outperforms that of Björklund et al. by several orders of magnitude on every problem instance, making ours *the fastest exact algorithm for complete set partitioning to date in practice*.

## 1. Introduction

The *Complete Set Partitioning* problem models the setting where every subset of a finite set is associated with a (positive or negative) real value, and the goal is to partition the set into pairwise disjoint subsets so as to maximise the sum of the subset values.[1] This problem captures a number of important applications. For instance, early papers on its algorithmic

[†]Tomasz Michalak and Talal Rahwan contributed equally to this article.

[1]One can also consider the minimisation variant of this problem, where the goal is to minimise the sum of subset values; however, since we allow both positive and negative subset values, these two variants of the problem are equivalent, so we focus of the maximisation problem.

complexity were motivated by the structuring of corporate tax in the United States [20, 21, 22] (see Section 7 for details). More recently, Complete Set Partitioning has been studied in the context of combinatorial auctions [36, 18], where there are multiple different items for sale, each bidder can place a bid on every subset of items, and the auctioneer's goal is to allocate the items to the bidders (and charge each bidder what she bid for the set of items she received) so as to maximise the total profit. It also plays an important role in the analysis of cooperation in multi-agent systems. Indeed, one of the most important aspects of such 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. The settings where the total worth of a coalition is determined solely by the identities of its members (and not by other co-existing coalitions) can be modeled by *characteristic function games*. In such games, we are given a set of agents, denoted by $A$, and the value of every subset, or *coalition*, of agents is specified by a *characteristic function* $v : 2^A \rightarrow \mathbb{R}$. To optimise the social welfare, we need to find a partition of agents into coalitions (a *coalition structure*) that maximises the sum of the values of the coalitions in the partition. A wide range of potential applications of coalition formation have been considered in the literature. For instance, by forming coalitions, autonomous sensors can improve their surveillance of certain areas [13], green-energy generators can reduce their uncertainty regarding their expected energy output [7], cognitive radio networks can increase their throughput [16], and buyers can obtain cheaper prices through bulk purchasing [19]; see the work of Sandholm et al. [38] and Rahwan et al. [35] for further examples.

In this article, our main goal is to improve our understanding of the complete set partitioning problem and to develop *exact* algorithms for this problem. Observe that the input to the complete set partitioning problem is a list of $2^n - 1$ real values (where $n$ is the size of the ground set), and every algorithm that is guaranteed to find an optimal solution on every instance of this problem has to inspect all of these values. Thus, the running time of every exact algorithm is at least exponential in $n$, and in some of the applications we have discussed (such as combinatorial auctions or multi-agent systems) the value of $n$ can be quite large. Therefore, an important goal is to design an algorithm that can find an optimal partition as efficiently as possible, both in the worst case and on average (for realistic value distributions).

The main techniques used in exact algorithms for solving computationally hard problems are: (1) dynamic programming, (2) tree search, (3) data preprocessing, and (4) local search [47]. With respect to our problem, we focus on two exact algorithms, which are based on techniques (1) and (2), respectively. Specifically:

- **DP**: This algorithm was originally proposed by Yeh [48] to solve the complete set partitioning problem, and was re-discovered by Rothkopf et al. [36], who used it to solve the winner determination problem in combinatorial auctions. This algorithm is based on *dynamic programming*: to find an optimal partition of the set of agents $A$, we start by computing an optimal partition of every subset $C \subseteq A$ with $|C| = 2$, then use those to compute an optimal partition of every $C \subseteq A$ with $|C| = 3$, and so on, until an optimal partition of $A$ is found.

- **IP**: This algorithm, which was proposed by Rahwan et al. [32] for the coalition structure generation problem, is based on a representation of the search space that groups partitions into disjoint subspaces based on the sizes of the subsets within each partition. 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 for different subspaces, it is possible to identify, and thus focus on, the promising subspaces. For every such subspace, the algorithm constructs multiple search trees, where every node represents a subset, and every path (from a root node to a leaf node) represents a partition. 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 avoid branches that have no potential of containing an optimal solution.

The above two algorithms are based on fundamentally different design paradigms, so it is not surprising that they exhibit quite different computational behaviour. More specifically, in what follows, we provide a comparison of these algorithms from three different perspectives:

- **Worst case performance:** The time required to exhaustively enumerate all partitions of an $n$-element set is $\Omega(n^{n/2})$ [38]. Such enumeration, however, involves repeating certain operations multiple times. As mentioned

earlier, DP avoids such repetition by storing partial solutions in memory, thereby lowering the required time to $O(3^n)$. On the other hand, the techniques used by IP to speed up the search cannot ensure that the worst-case time drops below $n^{n/2}$. 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 values of the subsets. It is possible to define a class of problem instances for which IP searches the entire space exhaustively (e.g., one in which every set $\{a_1, \ldots, a_s\}$ with $s \in \{1, \ldots, n\}$ has a value of 1, and every other set has a value of 0).

- **Average performance:** When tested on popular value distributions, IP has been shown to be faster than DP, by several orders of magnitude for some distributions [32]. 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 not capable of any such shortcuts.

- **Returning solutions anytime:** IP has the advantage of being an *anytime* algorithm: it returns a valid solution very quickly, and then improves on the quality of its solution over time, while establishing progressively-better guarantees on solution quality. DP, on the other hand, is not an anytime algorithm; it can only return a solution once it has successfully terminated. Being anytime is important since the size of the search space grows exponentially with the number of elements to be partitioned, and hence 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 one, and the quality of this solution improves over time.

The above comparison shows that each algorithm has its relative strengths and weaknesses compared to the other. In other words, there is a trade-off between the advantages of one algorithm and the advantages of the other. This raises the following question: *Is this trade-off inevitable?*

Against this background, the main contributions of this article are three-fold:

- **Analysing the search space**: We provide a theoretical analysis of the search space, which reveals how two fundamentally different exact algorithms can be combined and significantly improved upon in terms of actual runtime. This, in turn, contributes towards a better understanding of the set partitioning problem itself, and a better understanding of the complementarities that evidently exist between two algorithm-design paradigms, namely Dynamic Programming and Depth-First Search.

- **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. [38], where every node represents a partition. 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 items are in one set, DP makes a series of movements until it reaches an optimal node. This visualisation suggests that certain movements are not needed to reach an optimal node. We formalise this observation and use it to design our *Optimal Dynamic Programming (ODP)* algorithm, which performs only one third of DP's operations, without losing the guarantee of finding a best partition. ODP is optimal in that it avoids *all* redundant operations without losing the guarantee of finding an optimal solution.

- **Developing ODP-IP**: As we will show, ODP and IP approach the optimisation 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 algorithms, and reveals the potential of combining them into a single, superior algorithm. Building upon this analysis, we refine both algorithms, and use the refined versions as building blocks to construct a hybrid approach, called ODP-IP. The ODP-IP algorithm runs its two constituent algorithms in parallel, and uses the information provided by ODP to speed up IP's search. This approach results in an algorithm that has the best features of its components: it runs in $O(3^n)$ just like ODP, and returns solutions anytime, with

progressively-better guarantees, just like IP. Better still, when tested on a wide variety of value distributions, ODP-IP is empirically shown to significantly outperform both ODP and IP in all cases.[2]

Importantly, we benchmarked our algorithms against the primary alternative proposed in the literature, namely the *inclusion-exclusion* algorithm of Björklund et al. [8], which is an exact dynamic-programming algorithm for set partitioning that is built around the inclusion-exclusion principle. From a theoretical perspective, this is the state-of-the-art algorithm in terms of worst-case complexity; it runs in $O(2^n)$ time. However, when implementing it, we found that it requires multiplying numbers that consist of hundreds of digits, which tends to be costly in practice. As a consequence, in our tests the runtime of this algorithm grows at a rate of $O(6^n)$ rather than $O(2^n)$. For instance, given 15 agents, the algorithm requires more than five months to terminate, while our algorithm *for the worst-case problem instance* takes $0.01$ second (see Section 6.3 for more details). Thus, **ours is the fastest exact algorithm for complete set partitioning to date**.

The open-source Java implementation of all the algorithms discussed or developed in this article (namely, IP, DP, ODP, ODP-IP, as well as the inclusion-exclusion algorithm by Björklund et al. [8]) is publicly available at the following link: `https://github.com/trahwan/ODP-IP_and_InclusionExclusion`.

The remainder of this article is structured as follows. Section 2 formalises the complete set partitioning 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 notation used throughout the article. Appendix C and Appendix D contain the omitted proofs, while Appendix E discusses a certain aspect of ODP-IP in detail.

## 2. Preliminaries

In this section, we formally introduce the key definitions and notation used throughout the article. In what follows, we use the language of cooperative game theory, i.e., we talk about the coalition structure generation problem rather than the complete set partitioning problem. The reason for this is twofold. First, much of the recent work on this topic was done within the multi-agent research community, which views this problem from the coalition structure generation perspective, so adopting the language of cooperative games facilitates the comparison with prior work. The second reason is purely linguistic: when speaking about coalition structure generation, we refer to subsets of agents as *"coalitions"*, and thus avoid the overuse of the term *"set"*.

An instance of the *Coalition Structure Generation* problem is given by a finite set $A = \{a_1, a_2, \ldots, a_n\}$ and a function $v$ that assigns a real value to every non-empty subset of $A$. 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., $\mathcal{C}^A = \{C : C \subseteq A, C \neq \emptyset\}$; we have $\left|\mathcal{C}^A\right| = 2^n - 1$. 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 $\{a_4\} < \{a_4, a_5\}$.

An exhaustive partition of all the agents in a given set $C \subseteq A$ into disjoint coalitions is called a *coalition structure over* $C$. Formally, a coalition structure is defined as follows.

**Definition 1.** *Given a subset $C \subseteq A$, a* coalition structure over $C$ *is a collection of coalitions $CS = \{C_1, \ldots, C_{|CS|}\}$ that satisfies the following conditions: $\bigcup_{j=1}^{|CS|} C_j = C$, and for all $i, j \in \{1, \ldots, |CS|\}$ such that $i \neq j$ it holds that $C_i \cap C_j = \emptyset$.*

For each $C \subseteq A$, we will denote by $\Pi^C$ the set of all coalition structures over $C$. Furthermore, given a coalition structure $CS \in \Pi^C$, we will refer to the sum of the values of all coalitions in $CS$ as the *value of* $CS$, and denote it by $V(CS)$. Formally, $V(CS) = \sum_{C' \in CS} v(C')$. We are now ready to state our optimisation problem formally.

[2] Note that ODP and IP can only use a single processor each, while ODP-IP uses two processors running in parallel. While this alone can make ODP-IP twice as fast (assuming no overheads), our empirical evaluation shows that ODP-IP can be faster by one or two orders of magnitude, e.g., given 25 agents. This implies that the majority of the performance gain comes from the synergies between the two components, ODP and IP.

**Definition 2.** *The* coalition structure generation problem *is to find an optimal coalition structure* $CS^* \in \Pi^A$*, i.e., an (arbitrary) element of the set*

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

Given a coalition $C \subseteq A$, we denote by $f(C)$ the value of an optimal partition of $C$, i.e., $f(C) = V(CS)$, where $CS \in \arg\max_{CS \in \Pi^C} V(CS)$.

The coalition structure generation problem is computationally challenging, as the number of possible coalition structures over $n$ players, which is known as the $n$-th *Bell number* $B_n$ [6], satisfies $\alpha n^{n/2} \leq B_n \leq n^n$ for some positive constant $\alpha$ (see, e.g., the work of Sandholm et al. [38] for proofs of these bounds, and the book of de Bruijn [10] for an asymptotically tight bound). Moreover, it is NP-hard to find an optimal coalition structure given oracle access to the characteristic function [38].

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. [30], and (2) DP—the dynamic programming algorithm by Yeh [48].

### *3.1. The IP Algorithm*

The IP algorithm is based on the *integer partition-based representation* [29] 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 $n$ [1]. We denote the set of all integer partitions of $n$ by $\mathcal{I}^n$. 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* $\Pi^A_I \subseteq \Pi^A$ consisting of all coalition structures in which the sizes of the coalitions match the parts of $I$. For instance, $\Pi^{\{a_1,a_2,a_3,a_4\}}_{\{1,1,2\}}$ is the subspace consisting of all coalition structures over $\{a_1, a_2, a_3, a_4\}$ that contain two coalitions of size 1 and one coalition of size 2. A four-agent example is shown in Figure 1.

$$\Pi^A_{\{1,1,1,1\}} = \Big\{\{\{a_1\},\{a_2\},\{a_3\},\{a_4\}\}\Big\}$$

$$\Pi^A_{\{1,3\}} = \left\{\begin{array}{l}\{\{a_1,a_2,a_3\},\{a_4\}\},\\ \{\{a_1,a_2,a_4\},\{a_3\}\},\\ \{\{a_1,a_3,a_4\},\{a_2\}\},\\ \{\{a_2,a_3,a_4\}\ \{a_1\}\}\end{array}\right\} \qquad \Pi^A_{\{1,1,2\}} = \left\{\begin{array}{l}\{\{a_1,a_2\},\{a_3\},\{a_4\}\},\\ \{\{a_1,a_3\},\{a_2\},\{a_4\}\},\\ \{\{a_1,a_4\},\{a_2\},\{a_3\}\},\\ \{\{a_2,a_3\},\{a_1\},\{a_4\}\},\\ \{\{a_2,a_4\},\{a_1\},\{a_3\}\},\\ \{\{a_3,a_4\},\{a_1\},\{a_2\}\}\end{array}\right\}$$

$$\Pi^A_{\{4\}} = \Big\{\{\{a_1, a_2, a_3, a_4\}\}\Big\}$$

$$\Pi^A_{\{2,2\}} = \left\{\begin{array}{l}\{\{a_1,a_2\},\{a_3,a_4\}\},\\ \{\{a_1,a_3\},\{a_2,a_4\}\},\\ \{\{a_1,a_4\},\{a_2,a_3\}\}\end{array}\right\}$$

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

Using 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. To this end, for every coalition size $s \in \{1, 2, \ldots, n\}$, let $\mathcal{C}^A_s$ denote the set of all 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. Rahwan et al. [32] prove 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 $\Pi^A_I$, $I \in \mathcal{I}^n$, as follows.

**Theorem 3** (Rahwan et al. [32])**.** *For every* $I \in \mathcal{I}^n$*, let* $I(i)$ *be the multiplicity of* $i$ *in* $I$*. Then*

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

Since the value of the best coalition structure in $\Pi^A_I$ is at least the average value of the coalition structures in $\Pi^A_I$, we obtain the following *lower bound* on the value of the best coalition structure in $\Pi^A_I$: $LB_I = \sum_{i \in I} I(i) Avg_i$. By replacing $Avg_i$ with $Max_i$ in this expression, we obtain an *upper bound* $UB_I$ on the value of the best coalition structure in $\Pi^A_I$: $UB_I = \sum_{i \in I} I(i) Max_i$. 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 an optimal coalition structure $CS^*$. Knowing $UB^*$ enables us to bound the quality of $CS^{**}$—the best coalition structure found by the algorithm at a given point in time; we set $\beta = UB^*/V(CS^{**})$.[3] On the other hand, computing $LB^*$ is useful for identifying subspaces that have no potential of containing an optimal coalition structure: these are subspaces $\Pi^A_I$ with $UB_I < LB^*$. 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. If $LB^* < V(CS^{**})$, the algorithm also updates $LB^*$ by setting it to $V(CS^{**})$, and repeats the attempt of pruning unpromising subspaces, i.e., those whose upper bounds are smaller than the updated $LB^*$. The order in which the subspaces are searched is always based on the upper bounds: 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 $\Pi^A_I$, where $I = \{i_1, \ldots, i_k\}$, is carried out in a *depth-first* manner: the algorithm iterates over the coalitions in $\mathcal{C}^A_{i_1}$ and, for every coalition $C_1 \in \mathcal{C}^A_{i_1}$ 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 coalition $C_2 \in \mathcal{C}^A_{i_2}$ 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, $\mathcal{C}^A_{i_k}$, is reached. In this way, by the time the algorithm picks the last coalition $C_k \in \mathcal{C}^A_{i_k}$, it has selected a combination of $k-1$ coalitions that, together with $C_k$, form a coalition structure in $\Pi^A_I$. The algorithm repeats this process so that, eventually, every coalition structure in $\Pi^A_I$ is examined. Here, it should be noted that a straightforward 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 $|C_1| = |C_2|$. Rahwan et al. [32] explain how IP avoids such redundant operations.

To speed up the search, IP applies a *branch-and-bound* technique at every depth $d < k$. 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 $\mathcal{C}^A_{i_{d+1}}, \ldots, \mathcal{C}^A_{i_k}$, it checks whether

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

Now, if inequality (1) holds, 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 compute $Max_i$ and $Avg_i$ for all $i \in \{1, \ldots, n\}$. To do so, the algorithm iterates over the values of all coalitions, in order to compute the average and maximum values for coalitions of every 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 coalitions 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 *lexicographic* order, while *simultaneously* iterating over all coalitions of size $n - s$ in *anti-lexicographic* order.[4] With this order, the $i$-th coalition of size $s$ and 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 process, every subspace $\Pi^A_I$ with $|I| = 2$ is searched.

[3] This bound is meaningful only if the values of all coalitions are non-negative. However, it is only used to estimate the quality of the current solution when the algorithm is terminated prematurely, and the algorithm works correctly even if the characteristic function can take negative values.

[4] Such iteration can be carried out efficiently, e.g., using the techniques of Rahwan and Jennings [26].

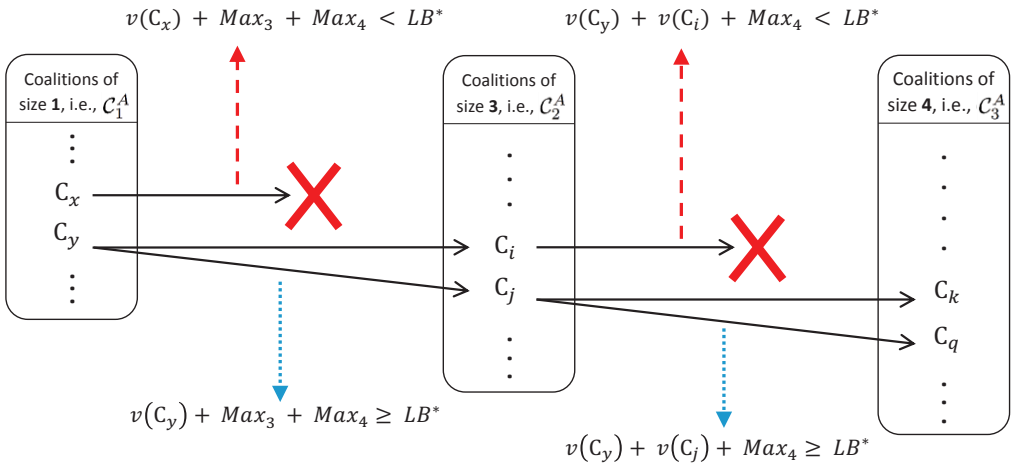



Figure 2: An illustration of IP's branch-and-bound technique when searching $\Pi^A_{\{1,3,4\}}$. Here, the algorithm recognises that the coalition structures containing $C_x$ or $C_y, C_i$ cannot be optimal, and so IP does not proceed deeper into the search tree.

The IP algorithm runs in $O(n^n)$ time, and in the worst case it can 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 $90\%$ when less than a $10^{-9}$ fraction of the search space for $25$ agents has been searched (given certain value distributions).

### *3.2. The DP Algorithm*

The DP algorithm is based on the following theorem.

**Theorem 4** (Yeh [48])**.** *Given a coalition $C \subseteq A$, the value of an optimal partition of $C$ 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 pseudocode of DP is given in Algorithm 1. 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 $C$. Once $f(C)$ and $t(C)$ are computed for every $C \subseteq A$, an optimal coalition structure $CS^*$ is computed recursively. A four-agent example is illustrated in Figure 3.

DP requires storing a total of $2^{n+1}$ values, namely $f(C)$ and $t(C)$ for every $C \subseteq A$. This memory requirement is not burdensome since we are dealing with the complete set partitioning problem, and the input to this problem contains $2^n$ values already. In other words, we implicitly assume there is $O(2^n)$ available space.

The running time of DP has been shown to be $O(3^n)$ [48]. 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 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 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 of DP, without losing the guarantee of finding an optimal solution.

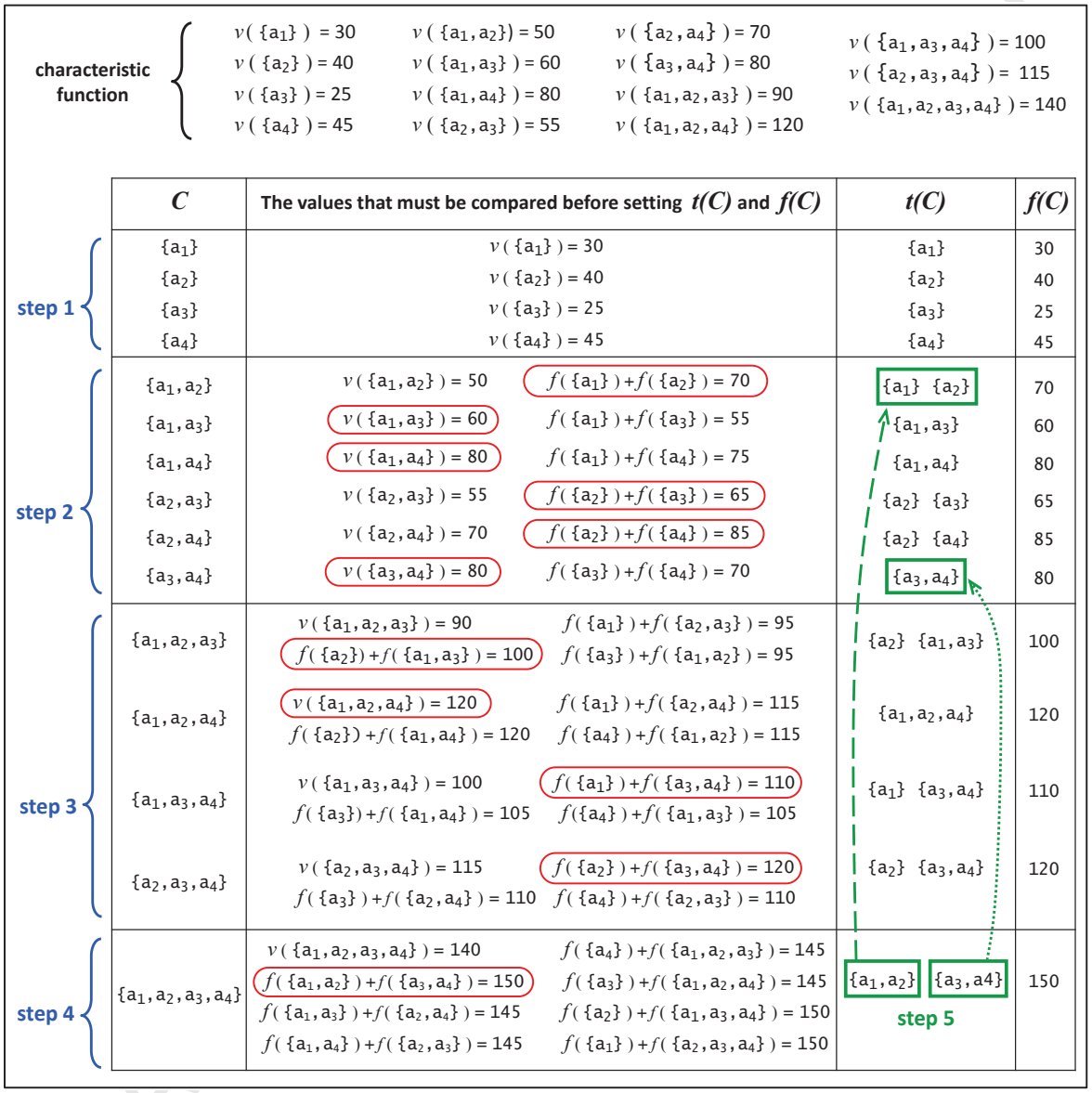



Figure 3: A four-agent example of how DP computes $t(C)$ and $f(C)$ for every $C \subseteq A$.

**ALGORITHM 1:** The DP algorithm.

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

1 **foreach** $C \subseteq A : |C| = 1$ **do** // for every coalition of size 1
2  $f(C) \leftarrow v(C)$
3  $t(C) \leftarrow \{C\}$
4 **foreach** $s = 2$ ***to*** $n$ **do**
5  **foreach** $C \subseteq A : |C| = s$ **do** // for every coalition of size $s$
6  $f(C) \leftarrow v(C)$
7  $t(C) \leftarrow \{C\}$ // start by considering the case where $C$ is not split
8  **foreach** $\{C', C''\} \in \Pi^C$ **do** // for every possible way of splitting $C$ in two
9  **if** $f(C) < f(C') + f(C'')$ **then**
10  $f(C) \leftarrow f(C') + f(C'')$ // to ensure that $f(C) = \max_{\{C',C''\} \in \Pi^C} \left(f(C') + f(C'')\right)$
11  $t(C) \leftarrow \{C', C''\}$ // to ensure that $t(C) \in \arg\max_{\{C',C''\} \in \Pi^C} \left(f(C') + f(C'')\right)$
// the algorithm has computed $t(C)$ and $f(C)$ for every $C \subseteq A$; the remaining lines compute $CS^*$
12 $CS^* \leftarrow \{A\}$
13 **foreach** $C \in CS^*$ **do**
14  **if** $t(C) \neq \{C\}$ **then** // i.e., if $\{C\}$ is not an optimal partition of $C$
15  $CS^* \leftarrow (CS^* \backslash \{C\}) \cup t(C)$ // replace $C$ with the two coalitions in $t(C)$
16  Go to line 13 and start with the new $CS^*$
17 **return** $CS^*$

### *4.1. The Link Between DP and the Coalition Structure Graph*

To obtain a deeper understanding of how DP works, we consider the *coalition structure graph* [38]. In this undirected graph, every node represents a coalition structure. These nodes are categorised into $n$ levels, namely $\Pi_1^A, \ldots, \Pi_n^A$, so that level $\Pi_i^A$ is composed of the nodes that represent coalition structures containing exactly $i$ coalitions each. An edge connects two coalition structures if and only if (1) they belong to two consecutive levels $\Pi_i^A$ and $\Pi_{i-1}^A$, and (2) the coalition structure in $\Pi_i^A$ can be obtained from the one in $\Pi_{i-1}^A$ by splitting one coalition into two. Figure 4 shows a four-agent example of the coalition structure graph. It also shows the values of all coalition structures based on the characteristic function from Figure 3.

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

1. **Task 1: evaluate all the movements in the graph**: For every coalition $C$ with $|C| \geq 2$, the algorithm *evaluates every partition* $\{C', C''\} \in \Pi^C$ by computing $f(C') + f(C'')$ (see line 9 of the pseudocode). 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**: In lines 8–11, the algorithm determines, 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 $t(C)$). In terms of the coalition structure graph, this step can be interpreted as follows. 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 $t(C) = \{C', C''\}$ means that, from any node representing $CS \ni C$, one of *the most beneficial movements* is to split $C$ into $C'$ and $C''$.

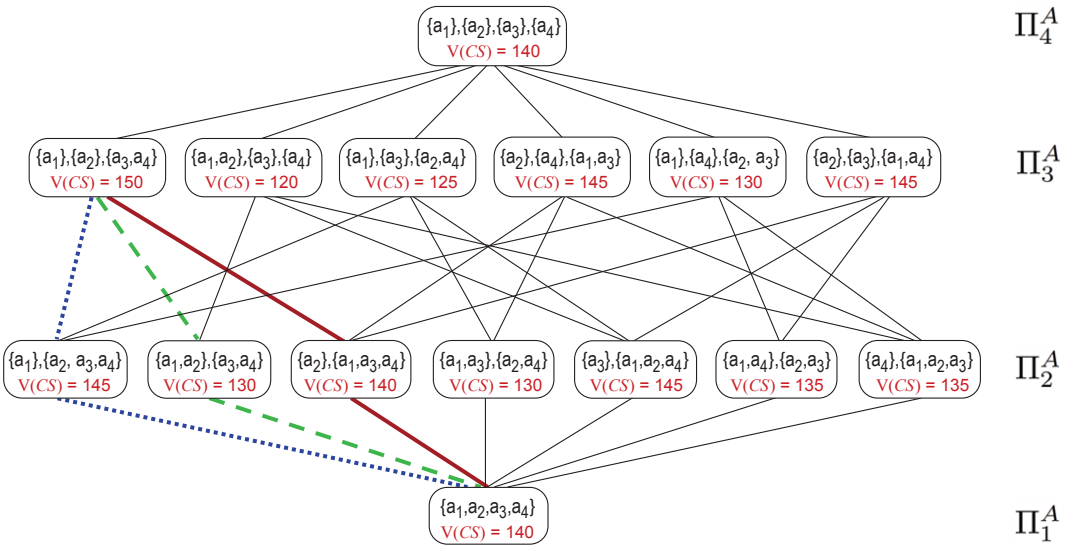



Figure 4: The coalition structure graph of four agents. The figure also shows the value of every coalition structure based on the characteristic function from Figure 3.

3. **Task 3: move upwards in the graph**: This occurs in lines 12 to 16. Here, DP first initialises $CS^*$ by setting $CS^* = \{A\}$. This means that DP starts at the node that represents $\{A\}$, i.e., the bottom node in the graph. After that, DP selects some coalition $C \in CS^*$ with $t(C) \neq \{C\}$ (if such a coalition exists), and replaces it with $t(C)$. By doing this, DP makes 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 visualised as a sequence of 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 $\{a_2\}$.

From this visualisation it is clear that, for every coalition structure $CS$ with $|CS| > 2$, there are multiple paths that start from the bottom node of the graph and end with the node that contains $CS$. For example, in Figure 4 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. Furthermore, if there are multiple paths that lead to the same optimal node, DP can reach this node through any of those paths. Indeed, we have not specified in which order the algorithm goes through the possible splits of $C$ (line 8), and for every choice of $\{C', C''\}$ from $\arg\max_{\{C',C''\} \in \Pi^C} \big(f(C') + f(C'')\big)$ there exists an order that results in $t(C)$ being set to $\{C', C''\}$. For example, in Figure 3, $t(\{a_1, a_2, a_3, a_4\})$ was set to $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ because this was *one of* the movements evaluated to 150. 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 this movement is also evaluated to 150. If that happened, DP would have found, based on $t(\{a_2, a_3, a_4\})$, that it is 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.

### *4.2. Analysing the Effect of Avoiding Certain Operations in DP*

We have shown 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 have also shown that DP is indifferent among the paths that lead to the same optimal node. All of these observations raise an important question: *"what happens if DP is modified so that it only evaluates some of the movements in the graph?"* Suppose that for a certain coalition $C$ the algorithm did not evaluate some movement that involves splitting $C$ into two coalitions, namely $C_1$ and $C_2$. That is, suppose that the term $\Pi^C$ in line 8 of the pseudocode was replaced with

$\Pi^C \setminus \{\{C_1, C_2\}\}$. 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, since DP always selects its movements from the table $t$, whenever a coalition structure $CS \ni C$ is reached, the movement to $CS' = (CS \setminus \{C\}) \cup \{C_1, C_2\}$ would no longer be an option. In other words, DP would ignore the existence of the edge that connects $CS$ to $CS'$, evaluate the movements through the remaining edges, and decide on its path accordingly. This can be visualised 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 formalise this observation in the remainder of this section.

Given 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 $C_2$. Observe that the 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 the coalition structure $CS' = (CS \setminus \{C\}) \cup \{C_1, C_2\}$. Further, 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 a coalition $C \subseteq A$, a subset of movements $M \subseteq \mathcal{M}$ and two partitions $\pi, \pi' \in \Pi^C$, we write $\pi \xrightarrow{M} \pi'$ if and only if $\pi'$ can be reached from $\pi$ via *a single movement* in $M$. That is, we set

$$\pi \xrightarrow{M} \pi' \text{ iff } \pi' = (\pi \setminus \{C_1 \cup C_2\}) \cup \{C_1, C_2\} \text{ for some } m^{C_1,C_2} \in M.$$

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

**Definition 5.** *Given a coalition $C \subseteq A$, a subset of movements $M \subseteq \mathcal{M}$ and two partitions $\pi, \pi' \in \Pi^C$, we say that $\pi'$* is reachable from $\pi$ via $M$*, and write $\pi \overset{M}{\rightsquigarrow} \pi'$, if and only if $\pi'$ is either equal to $\pi$, or can be reached from $\pi$ via one or more movements in $M$. More formally,*

$$\pi \overset{M}{\rightsquigarrow} \pi' \text{ iff } \pi = \pi' \text{ or } \pi \xrightarrow{M} \pi' \text{ or } \exists \{\pi_1, \dots, \pi_k\} \subseteq \Pi^C : \pi \xrightarrow{M} \pi_1 \xrightarrow{M} \dots \xrightarrow{M} \pi_k \xrightarrow{M} \pi'.$$

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

$$R^\pi_M = \{\pi\} \cup \bigcup_{\pi' \in \Pi^C : \pi \xrightarrow{M} \pi'} R^{\pi'}_M. \tag{3}$$

Now, let us define $f_M(C)$ as the value of an optimal partition in $R^{\{C\}}_M$. More formally, $f_M(C) = \max_{\pi \in R^{\{C\}}_M} V(\pi)$. With this definition, we are ready to generalise Theorem 4 (the main theorem behind DP) by replacing $f(C)$ and $\Pi^C$ with $f_M(C)$ and $R^{\{C\}}_M$, respectively.

**Theorem 6.** *For every coalition $C \subseteq A$ and for every subset of movements $M \subseteq \mathcal{M}$ it holds that*

$$f_M(C) = \begin{cases} v(C) & \text{if } |C| = 1 \\ \max\left\{ v(C),\ \max_{\{C', C''\} \in R^{\{C\}}_M} \left( f_M(C') + f_M(C'') \right) \right\} & \text{otherwise.} \end{cases} \tag{4}$$

For the proof of Theorem 6, see Appendix C.

Now, we can analyse the effect of replacing every $f(C)$ and $\Pi^C$ in DP with $f_M(C)$ and $R^{\{C\}}_M$, respectively. Let us call the resulting algorithm DP$_M$. Theorem 6 implies that DP$_M$ computes $f_M(C)$ recursively for every $C \subseteq A$. When DP$_M$ terminates, it has 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 as DP does. This leads to the following corollary.

**Corollary 7.** *For an arbitrary subset of movements $M \subseteq \mathcal{M}$, the algorithm DP$_M$ outputs a coalition structure in* $\arg\max_{CS \in R_M^{\{A\}}} V(CS)$.

Having analysed the effect of avoiding the evaluation of certain movements in the coalition structure graph, we will now use this analysis to design an optimal version of DP.

### *4.3. The ODP Algorithm*

In this section, we present our *optimal dynamic programming (ODP)* algorithm—a modified version of DP that avoids all the redundant operations of DP, while maintaining the guarantee of finding an optimal solution. Of course, it would be possible to avoid all redundant operations by simply considering all movements, and checking them one by one to identify (and avoid) any movements that lead to an already-examined split. This, however, would require storing *all* movements, rather than just the most promising ones, as is currently the case. Instead, ODP identifies the relevant movements *a priori*, without any need for extra memory requirements, and without having to search for these relevant movements.

According to Corollary 7, given a subset of movements $M$, DP$_M$ finds an optimal coalition structure if and only if $R_M^{\{A\}} = \Pi^A$. We will now identify a "small" set of movements $M$ for which this is the case. Recall that, given two coalitions $C', C'' \in \mathcal{C}^A$, we write $C' < C''$ if and only if $C'$ precedes $C''$ lexicographically.[5] Set

$$M^* = \left\{ m^{C',C''} \in \mathcal{M} : C' \cup C'' = A \text{ or } C' < C'' < A \setminus (C' \cup C'') \right\}. \tag{5}$$

It turns out that, to find an optimal partition, it suffices to consider the movements in $M^*$.

**Theorem 8.**

$$R_{M^*}^{\{A\}} = \Pi^A. \tag{6}$$

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

First, suppose that $k = 2$. In this case, we have $CS = \{C_1, C_2\}$, and so $C_1 \cup C_2 = A$. This means that $m^{C_1,C_2} \in M^*$.

Now, suppose that $k > 2$. In this case, since $C_1 < \cdots < C_k$, we obtain $C_1 < C_2 < (C_3 \cup \cdots \cup C_k)$, and hence $C_1 < C_2 < A \setminus (C_1 \cup C_2)$. Thus, $m^{C_1,C_2} \in M^*$ in this case as well. □

We are now ready to define our algorithm, which we call ODP.

**Definition 9.** *ODP is the version of DP that only evaluates the movements in $M^*$, i.e., uses $f_{M^*}$ instead of $f$. Formally, ODP = DP$_{M^*}$.*

Theorem 8 together with Corollary 7 imply that ODP finds an optimal partition of $A$. We will now analyse the running time of this algorithm. First, we prove the following two important lemmas. The first lemma will be used in our implementation of ODP (see lines 5 to 25 of Algorithm 3 in Appendix B), while the second lemma states that ODP does not evaluate any redundant movements.

**Lemma 10.** *For every coalition $C \in \mathcal{C}^A$ such that $\{a_1, a_2\} \not\subseteq C$, the ODP algorithm does not evaluate any of the possible ways of splitting $C$.*

[5] We chose this particular ordering as it helps us prove Theorem 8. However, we do not imply that this is the only ordering that serves this purpose.

*Proof.* Consider a coalition $C \in \mathcal{C}^A$ such that $\{a_1, a_2\} \not\subseteq C$. We will prove that for all $m^{C',C''} \in \mathcal{M}$ such that $C' \cup C'' = C$ it holds that $m^{C',C''} \notin M^*$.

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 $C' \cup C'' \neq A$ and $A \setminus (C' \cup C'') < C'$. Thus, $m^{C',C''} \notin M^*$ according to (5).
- **Case 2:** $a_1 \in C$ and $a_2 \notin C$. In this case, one of the two coalitions in $\{C', C''\}$ contains neither $a_1$ nor $a_2$. Let this coalition be $C''$. Now, since $a_2 \in A \setminus (C' \cup C'')$, we have $C' \cup C'' \neq A$ and $A \setminus (C' \cup C'') < C''$. This implies that $m^{C',C''} \notin M^*$ according to (5).

□

**Lemma 11.** *For every coalition structure $CS$ with $|CS| \geq 2$, the ODP algorithm evaluates exactly one movement that leads to $CS$.*

*Proof.* Consider a coalition structure $CS = \{C_1, \ldots, C_k\}$ with $k \geq 2$. Without loss of generality, we can 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 $C_1 \cup C_2 = A$, we have $m^{C_1,C_2} \in M^*$.
- **Case 2:** $k > 2$. In this case, we have $C_1 < C_2 < A \setminus (C_1 \cup C_2)$, so $m^{C_1,C_2} \in M^*$. It remains to show that no other movement in $M^*$ leads to $CS$. To this end, observe that a movement $m^{C_i,C_j} \in \mathcal{M}$ leads to $CS$ only if $C_i, C_j \in CS$. We will show that if $\{i, j\} \neq \{1, 2\}$, then $m^{C_i,C_j} \notin M^*$. This is a direct consequence of the following observations.
  - If $1 \notin \{i, j\}$, then $C_1 \subseteq A \setminus (C_i \cup C_j)$ and hence $A \setminus (C_i \cup C_j) < C_i$. Therefore, $m^{C_i,C_j} \notin M^*$.
  - If $1 \in \{i, j\}$ and $2 \notin \{i, j\}$, then $C_2 \subseteq A \setminus (C_i \cup C_j)$ and 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 will now establish a one-to-one correspondence between movements in $M^*$ and partitions of $A$ into two and three parts.

**Theorem 12.** *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,*

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

*Proof.* Fix a coalition structure $CS = \{C_1, \ldots, C_k\}$ with $k > 1$, and assume without loss of generality that $C_1 < \cdots < C_k$. To establish a one-to-one correspondence between $M^*$ and $\Pi_2^A \cup \Pi_3^A$, it is sufficient to make the following observations.

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

$$\left|\{m^{C',C''} \in M^* : C' \cup C'' = A\}\right| \;=\; \left|\Pi_2^A\right|. \tag{7}$$

- Every movement $m^{C',C''} \in M^*$ with $C' \cup C'' \subset A$ leads to exactly one coalition structure in $\Pi_3^A$, namely $\{C', C'', A \setminus (C' \cup C'')\}$, as this is the only coalition structure in $\Pi_3^A$ that contains both $C'$ and $C''$. Similarly, every coalition structure $\{C_1, C_2, C_3\} \in \Pi_3^A$ with $C_1 < C_2 < C_3$ is reachable via exactly one movement in $M^*$, namely, $m^{C_1,C_2}$. This means that there is a one-to-one correspondence between $\Pi_3^A$ and $\{m^{C',C''} \in M^* : C' \cup C'' \subset A\}$, and hence

$$\left|\{m^{C',C''} \in M^* : C' \cup C'' \subset A\}\right| \;=\; \left|\Pi_3^A\right|. \tag{8}$$

Combining equations (7) and (8), we obtain the desired result. □

We can use Theorem 12 to compute the size of $M^*$ (for the proof, see Appendix C).

**Corollary 13.** *The number of movements in $\mathcal{M}$ is $\frac{1}{2}(3^n + 1) - 2^n$, whereas the number of movements in $M^*$ is $\frac{1}{2}(3^{n-1} - 1)$.*

Corollary 13 shows that ODP evaluates roughly one third of the movements evaluated by DP.

More importantly, Theorem 12 can be used to show that it is not possible to evaluate fewer movements than those evaluated by ODP and still be guaranteed to find an optimal solution.

**Theorem 14.** *For every subset of movements $M \subseteq \mathcal{M}$ such that $|M| < |M^*|$ we have $R_M^{\{A\}} \neq \Pi^A$.*

*Proof.* Suppose that $R_M^{\{A\}} = \Pi^A$; we will argue that in this case $M$ has to contain at least $|\Pi_2^A| + |\Pi_3^A| = |M^*|$ movements.

Consider an arbitrary coalition structure of size 2, say, $CS^2 = \{C_1, C_2\}$. The only way to reach $CS^2$ is to make the movement $m^{C_1,C_2}$ from $\{A\}$. Thus, since $\text{DP}_M$ reaches all coalition structures, we have $m^{C_1,C_2} \in M$. Further, observe that if $\{C_1, C_2\}$ and $\{C_1', C_2'\}$ are two different coalition structures of size 2, then $m^{C_1,C_2} \neq m^{C_1',C_2'}$.

Similarly, consider an arbitrary coalition structure of size 3, say, $CS^3 = \{C_1, C_2, C_3\}$. If $M$ contains none of the movements $m^{C_1,C_2}$, $m^{C_1,C_3}$, $m^{C_2,C_3}$, then $\text{DP}_M$ cannot reach $CS^3$. Further, if $\{C_1, C_2, C_3\}$ and $\{C_1', C_2', C_3'\}$ are two different coalition structures of size 3, then the sets $\{m^{C_1,C_2}, m^{C_1,C_3}, m^{C_2,C_3}\}$ and $\{m^{C_1',C_2'}, m^{C_1',C_3'}, m^{C_2',C_3'}\}$ are disjoint. Indeed, the only coalition structure of size 3 that can be reached by a movement in the former set is $\{C_1, C_2, C_3\}$, whereas the only coalition structure of size 3 that can be reached by a movement in the latter set is $\{C_1', C_2', C_3'\}$. Moreover, none of the movements in the set $\{m^{C_1,C_2}, m^{C_1,C_3}, m^{C_2,C_3}\}$ can be used to reach a coalition structure of size 2, as we have $|C_1| + |C_2| < n$, $|C_1| + |C_3| < n$, $|C_2| + |C_3| < n$.

Now, if $\text{DP}_M$ can reach all coalition structures, then for each coalition structure of size 2 or 3 the set $M$ contains a movement that reaches this coalition structure, and we have argued that all these movements must be pairwise distinct. It follows that if $R_M^{\{A\}} = \Pi^A$, then $|M| \geq |\Pi_2^A| + |\Pi_3^A| = |M^*|$, which is what we wanted to prove. □

Appendix B provides the pseudocode of ODP, and shows how to avoid storing the table $t$, thus leading to a significant reduction in memory requirements at the expense of a negligible increase in computation time.

## 5. The ODP-IP Algorithm

Having detailed the ODP algorithm, we will now show how to combine it with the IP algorithm. Our starting point is the "vanilla" hybrid algorithm, which runs the two algorithms in parallel and terminates as soon as the faster of the two returns an answer. Our main contribution here is to modify ODP and IP so that they can assist one another during this process. Ideally, this should be done so that the two algorithms explore non-overlapping portions of the search space, while dividing the labour in an ad hoc manner (rather than *a priori*) to reflect the actual strengths of the two algorithms. For instance, if ODP happens to be twice as fast as IP on a given problem instance, then ODP must naturally end up putting twice as much effort as IP. Another desirable property would be to have some meaningful information flow between the two algorithms, rather than having each one working independently without acknowledging the presence of the other. Here, the goal would be to have some synergy between the two, making the overall outcome greater than the

sum of its parts. The main challenge here stems from the fact that ODP and IP are based on entirely different design paradigms.

Against this background, we present in Section 5.1 a new representation of the search space, which provides the corner stone upon which our hybrid algorithm is built. Based on this, we show in Section 5.2 how to modify ODP such that it searches subspaces of the integer partition graph—those same subspaces that IP was originally designed to explore. After that, in Section 5.3 we show how to use the information provided by ODP to speed up IP's depth-first search, while in Section 5.4 we show how to modify IP so that it searches multiple subspaces simultaneously, building upon ODP's partial outcome. When combining the modified versions of ODP and IP, we obtain our hybrid algorithm, ODP-IP, a summary of which is provided in Section 5.5.

### *5.1. The Link Between DP and IP*

Given the differences between ODP and IP, in terms of both the search-space representation and the search techniques that are being used, it is not trivial to determine how these two algorithms can be combined effectively, i.e., how to divide the search effort between the two algorithm in a meaningful way. As a starting step, we will draw a link between IP and DP (not ODP). In order to do so, we introduce yet another graph, which we call the *integer partition graph*. This is an undirected graph where every node represents an integer partition, and two nodes representing integer partitions $I$ and $I'$ are connected by an edge if and only if there exist 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).

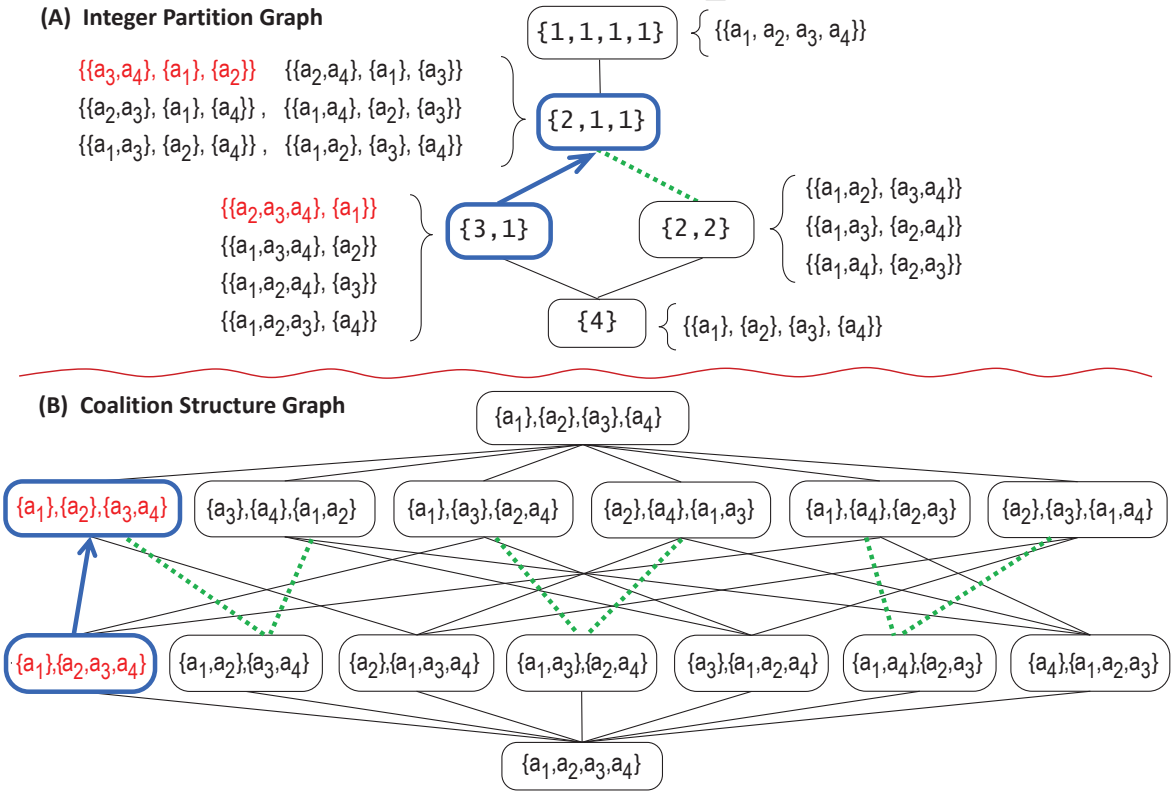



Figure 5: The integer partition graph (A) and the coalition structure graph (B) for four agents. A movement in (B) corresponds to a movement in (A), and the removal of the dotted edges in (B) corresponds to the removal of the dotted edge in (A).

By looking at this graph, we can visualise the way DP searches the subspaces that are represented by different integer partitions. To this end, recall that the operation of DP can be interpreted as the evaluation of movements in the coalition structure graph. Furthermore, avoiding the evaluation of some of these movements can be interpreted as removing the

edges through which these movements are made. Importantly, these same operations can also be visualised on the integer partition graph. Specifically, we make the following observations.

- By making a movement from a coalition structure $CS$ to a coalition structure $CS'$ in the coalition structure graph, DP makes a movement from the integer partition $I$ with $CS \in \Pi^A_I$ to the integer partition $I'$ with $CS' \in \Pi^A_{I'}$ in the integer partition graph. For example, 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).

- Removing all edges of the coalition structure graph that correspond to splitting a coalition of size $s$ into two coalitions of sizes $s'$ and $s''$ corresponds to removing every edge of the integer partition graph that connects an integer partition $I$ with $s \in I$ to the integer partition $I' = (I \setminus \{s\}) \uplus \{s', s''\}$. For instance, removing the dotted edges in Figure 5(B) corresponds to removing the dotted edge that connects $\Pi^A_{\{2,2\}}$ to $\Pi^A_{\{2,1,1\}}$ in Figure 5(A). This is because it is no longer possible to move from a coalition structure in $\Pi^A_{\{2,2\}}$ to a coalition structure in $\Pi^A_{\{2,1,1\}}$.

This visualisation provides a link between DP and IP, since the latter deals with subspaces that are represented by integer partitions. Building upon this, we show in the next section how to divide the search effort between ODP and IP.

### *5.2. Searching Subspaces Using ODP*

We have shown that, for a given triple of positive integers $s, s', s''$ with $s' + s'' = s$, avoiding the evaluation of *all* possible ways of splitting *all* coalitions of size $s$ into two coalitions of 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 $s = 2$, one can check that 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 $s, s', s'' \in \{1, \dots, n\}$ such that $s = s' + s''$, evaluates either *all* or *none* of the movements in which a coalition of size $s$ is split into coalitions of sizes $s'$ and $s''$. While the resulting version of DP still performs some redundant evaluations, we will later see that its performance is very close to that of ODP.

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

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

We will now show that, in order to find an optimal coalition structure, it suffices to evaluate all movements in $M^{**}$. The proof of the following theorem is similar to that of Theorem 8, and can be found in Appendix D.

**Theorem 15.**

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

Theorem 15 shows that $\text{DP}_{M^{**}}$ finds an optimal coalition structure. Furthermore, while it clearly performs some redundant evaluations, we will now argue that its running time is very close to that of ODP.

We will first show that $\text{DP}_{M^{**}}$ evaluates none of the movements from a coalition of size $s$, where $s \in \{\lfloor \frac{2n}{3} \rfloor + 1, \dots, n-1\}$. The proof of the following lemma is similar to that of Lemma 10, and can be found in Appendix D.

**Lemma 16.** *The $\text{DP}_{M^{**}}$ algorithm does not evaluate any of the possible ways of splitting a coalition of size $s$, where $s \in \{\lfloor \frac{2n}{3} \rfloor + 1, \dots, n-1\}$.*

We can now provide an upper bound of the number of movements evaluated by $\text{DP}_{M^{**}}$ (for the proof, see Appendix D).

**Theorem 17.** *The number of movements in $M^{**}$ is $\frac{1}{2}3^{n-1} + o(3^n)$.*

Theorem 17 means that $\text{DP}_{M^{**}}$ is essentially just as fast as ODP.

Next, we show how to further modify $\text{DP}_{M^{**}}$ so that it searches subspaces of the integer partition graph. To this end, observe that $\text{DP}_{M^{**}}$ works in three main steps:

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

We modify $\text{DP}_{M^{**}}$ by changing this sequence of operations as follows:

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

In what follows, we refer to the resulting algorithm as the *size-based version of ODP*, or *sb-ODP*; its pseudocode is given in Algorithm 2. To understand the intuition behind these modifications, let us consider a 10-agent example, where sb-ODP has just finished evaluating the movements $m^{C',C''} \in M^{**}$ such that $|C'| + |C''| \in \{2, 3\}$. At this moment, although some movements in $M^{**}$ have not yet been evaluated, sb-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 7, 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 sb-ODP thus far) until no such splitting is beneficial. Similarly, as soon as the movements $m^{C',C''} \in M^{**}$ with $|C'| + |C''| = 4$ are evaluated, more edges are added to the graph, and so more subspaces 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 $s$, sb-ODP gradually evaluates more and more subspaces, until it eventually searches the entire space.

We remark that, unlike $\text{DP}_{M^{**}}$, sb-ODP is an anytime algorithm: at any point in time $CS^{**}$ stores the best coalition structure identified so far, and the value of this coalition structure goes up as $s$ increases. However, this improvement comes at a price: while $\text{DP}_{M^{**}}$ evaluates each movement at most once, sb-ODP evaluates some of the movements twice. Specifically, a movement of the form $m^{C',C''}$ with $|C'| + |C''| = n$, $|C'| \leq \lfloor \frac{2n}{3} \rfloor$, $|C''| \leq \lfloor \frac{2n}{3} \rfloor$, $|C'| < |C''|$, is evaluated first for $s = |C'|$ and then for $s = |C''|$. Fortunately, the number of such movements is less than $2^{n-1} = o(3^n)$, and all other movements in $M^{**}$ are evaluated once. Thus, we obtain the following corollary.

**Corollary 18.** *The sb-ODP algorithm performs $\frac{1}{2}3^{n-1} + o(3^n)$ evaluations.*

So far in this section, we have developed a size-based version of ODP, and shown how to modify it so that it 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 focus on these subspaces, and avoid searching the ones that have been searched by ODP[6]. This division of work (between ODP and IP) gives ODP-IP the ability to calibrate itself automatically so that the amount of search

[6]From now on, whenever we talk about ODP-IP, we mean the combination of IP and sb-ODP. However, for readability, we will write "ODP" instead of "sb-ODP".

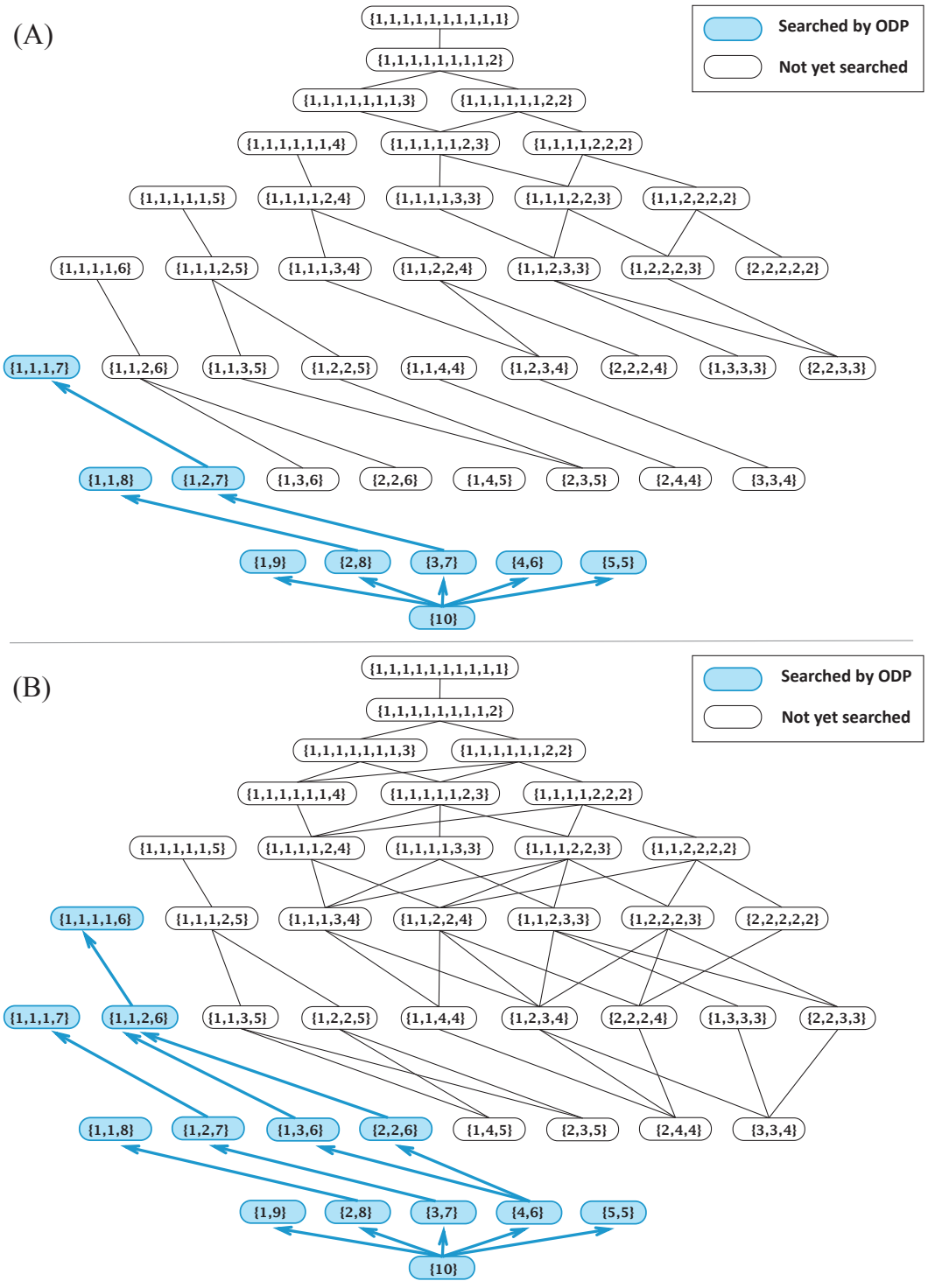



Figure 6: Illustration of how sb-ODP gradually covers all subspaces given ten agents. Figure 6(A) illustrates the subspaces that are reachable when sb-ODP finishes evaluating the movements $m^{C,C'} \in M^{**}$ with $|C|+|C'| \in \{2,3\}$. Figure 6(B) illustrates reachable subspaces with the evaluated movements being $m^{C,C'} \in M^{**}$ for $|C|+|C'| \in \{2,3,\mathbf{4}\}$.

**ALGORITHM 2:** The size-based version of ODP.

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

$t(A) \leftarrow \{A\}$; $CS^{**} \leftarrow \{A\}$ `// initialisation`
`// First, initialise` $f_{M^{**}}(C)$ `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` $s = 2, \ldots, \lfloor \frac{2n}{3} \rfloor$
**for** $s = 2$ ***to*** $\lfloor \frac{2n}{3} \rfloor$ **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 pseudocode in Algorithm 4`
**return** $CS^{**}$

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 share the workload without duplicating each other's efforts, in the following sections we show how to further enhance this combination. In particular, we will focus on how IP's performance can be improved by using the information that has been already computed by ODP.

### *5.3. Speeding up IP's Depth-First Search*

As mentioned in Section 3.1, every time IP reaches a certain depth $d$ in the search tree of a subspace $\Pi_I^A$, it adds a coalition $C_d$ to a set of disjoint coalitions $\{C_1, \ldots, C_{d-1}\}$. After that, it determines whether it is worthwhile to go deeper into the search tree. To do so, it checks whether inequality (1) holds. If not, then the set of all coalition structures in $\Pi_I^A$ that start with $\{C_1, \ldots, C_{d-1}\}$ is considered *promising*, i.e., one of the coalition structures in this set 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 into 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 goes one step deeper into the search tree, the information in $w$ is kept up-to-date throughout the search. Now, to use this information, IP is modified so that, at depth $d$, it checks whether one 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 an optimal coalition structure $CS^*$ such that $C_d \in CS^*$. In either case, every coalition structure containing $\{C_1, \ldots, C_d\}$ can be skipped during the search. Note that this pruning occurs *even if the set of all coalition structures that contain* $\{C_1, \ldots, C_d\}$ *is promising*. This is because the pruning here occurs whenever an optimal coalition structure cannot possibly appear among the coalition structures containing $\{C_1, \ldots, C_d\}$, *even if one of these coalition structures is 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 the table $w$, and ODP use another table, i.e., $f_{M^{**}}$, 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 make its decisions based not only on the best partitions that it has encountered, but also on 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 optimal partitions of all coalitions of size 9 or less in a very short time (e.g., less than 0.2 seconds on a standard desktop PC, see Section 6 for more details). Now suppose that, after this short time, IP started searching the subspace $\Pi^A_{\{2,2,2,2,1,1,3,3,3\}}$. As mentioned earlier, IP goes deeper into the search tree as long as it encounters promising coalition structures. For instance, suppose that the set of all coalition structures containing $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}, \{a_9\}$ happens to be promising. With the new branch-and-bound technique, and with the information now provided by ODP, this combination of coalitions would only be reached by IP if all of the following conditions 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 all these events happening simultaneously is extremely low for reasonable distributions of coalitional values, even if the set of all coalition structures containing $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}, \{a_9\}$ is promising. This example clearly demonstrates the great potential of this new branch-and-bound technique for speeding up the search.

### *5.4. Searching Multiple Subspaces Simultaneously*

In this section, we show how to modify IP so that it searches multiple subspaces simultaneously and thus avoids repeating certain operations. After that, we show how IP can use this technique more effectively using the partial outcome of ODP. For presentation clarity, we will postpone the formal description of this technique until after we have presented the basic idea through an example of five subspaces.

Recall that IP searches each subspace in a depth-first manner. The crucial idea behind the modification we are going to describe is that the first few levels of IP's search tree for a given subspace $\Pi_I^A$ can be *exactly the same* as those for several other subspaces.[7] 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 IP does) involves constructing pairs of disjoint coalitions $C_1, C_2$ with $|C_1| = \mathbf{2}, |C_2| = \mathbf{4}$ (for more details, see Section 3.1). Now, instead of repeating this process for every one of these five subspaces, we need only perform it once. More specifically, for every pair $C_1, C_2$ with $|C_1| = \mathbf{2}, |C_2| = \mathbf{4}$, we can perform the following steps:

1. Compute the value of $\{C_1, C_2, A \setminus (C_1 \cup C_2)\}$—the only coalition structure in $\Pi_{I_1}^A$ 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 and 3, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi_{I_2}^A$ 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 and 2, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi_{I_3}^A$ 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 and 2, and add those to $\{C_1, C_2\}$. This gives the best coalition structure in $\Pi_{I_4}^A$ 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 $\Pi_{I_5}^A$ that contains $C_1$ and $C_2$.
6. Select the best out of the coalition structures that were found in the above five steps.

[7] Note that, for any given subspace $\Pi_I^A$, the shape of the search tree depends on the ordering of the integers in $I$, so we assume that for each $I$ this ordering has been fixed in advance. Thus, effectively, in this section we treat $I$ as a list rather than a multi-set.

This procedure returns the best coalition structure containing $C_1$ and $C_2$ in the set $\cup_{i=1}^{5}\Pi^A_{I_i}$. By repeating this procedure for every pair $C_1, C_2$ with $|C_1| = \mathbf{2}, |C_2| = \mathbf{4}$, we can 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, suppose that IP started searching $\Pi^A_{I_1}, \ldots, \Pi^A_{I_5}$ after ODP has finished evaluating the movements $m^{C,C'} \in M^{**}$ with $|C| + |C'| \in \{2, 3, 4\}$. This means that, for all $C \subseteq A$ with $|C| \in \{2, 3, 4\}$, ODP has computed $f_{M^{**}}(C)$. In this case, for every pair $C_1, C_2$ with $|C_1| = 2, |C_2| = 4$, 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

1. Compute $v(C_1) + v(C_1) + f_{M^{**}}(A \setminus (C_1 \cup C_2))$.

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

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

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

It remains 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 do not yet know the partition itself). This 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 4). 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, i.e., $\Pi^A_{I_2}$, $\Pi^A_{I_3}$, $\Pi^A_{I_4}$, and $\Pi^A_{I_5}$.

In general, the modified version of IP proceeds as follows. As before, it picks the next subspace $\Pi^A_I$ to evaluate based on its upper bound $UB_I$. Next, it chooses an integer $s$ in $I$[8] such that ODP has already evaluated $f_{M^{**}}(C)$ for all coalitions $C$ with $|C| = s$. It then goes over all coalition structures in $\Pi^A_I$, and evaluates them: the coalitions that match integers in $I \setminus \{s\}$ are evaluated according to $v$, and the coalition that matches $s$ is evaluated according to $f_{M^{**}}$. This has the effect of simultaneously searching all subspaces that are reachable from $\Pi^A_I$ by splitting $s$. The resulting coalition structure can then be found using `getBestPartition`. To enhance readability, the details of this procedure are moved to Appendix E.

So far, we have shown how multiple subspaces can be searched simultaneously by partitioning *exactly one* coalition. However, one can partition *multiple* coalitions. 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 $f_{M^{**}}(C_1) + f_{M^{**}}(C_2) + f_{M^{**}}(C_3)$, then the result of this search will be a coalition structure $CS' = \{C'_1, C'_2, C'_3\}$ that maximises $f_{M^{**}}(C'_1) + f_{M^{**}}(C'_2) + f_{M^{**}}(C'_3)$. 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 $\Pi^A_I$, IP encounters a new combination of disjoint coalitions every time it takes one step deeper into the search tree. For every such combination $\{C_1, \ldots, C_d\}$, IP computes an upper bound on the value of every coalition structure $CS \in \Pi^A_I$ such that $\{C_1, \ldots, C_d\} \subseteq CS$. 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 computation of the upper bound must take into consideration all of those subspaces. In our example, the upper bound must be on the value of every $CS \in \Pi^A_I \cup \cdots \cup \Pi^A_{I_5}$ with $\{C_1, \ldots, C_d\} \subseteq CS$. This makes it more difficult to discard branches of the search tree. Appendix E provides more details on how to split multiple integers, and compares this approach with the one where only a single integer is partitioned. We remark that in our experimental evaluation (Section 6) we always split a single integer.

[8] Recall that in this section $I$ is treated as a list, so if $I$ contains multiple copies of $s$, only one of them is selected.

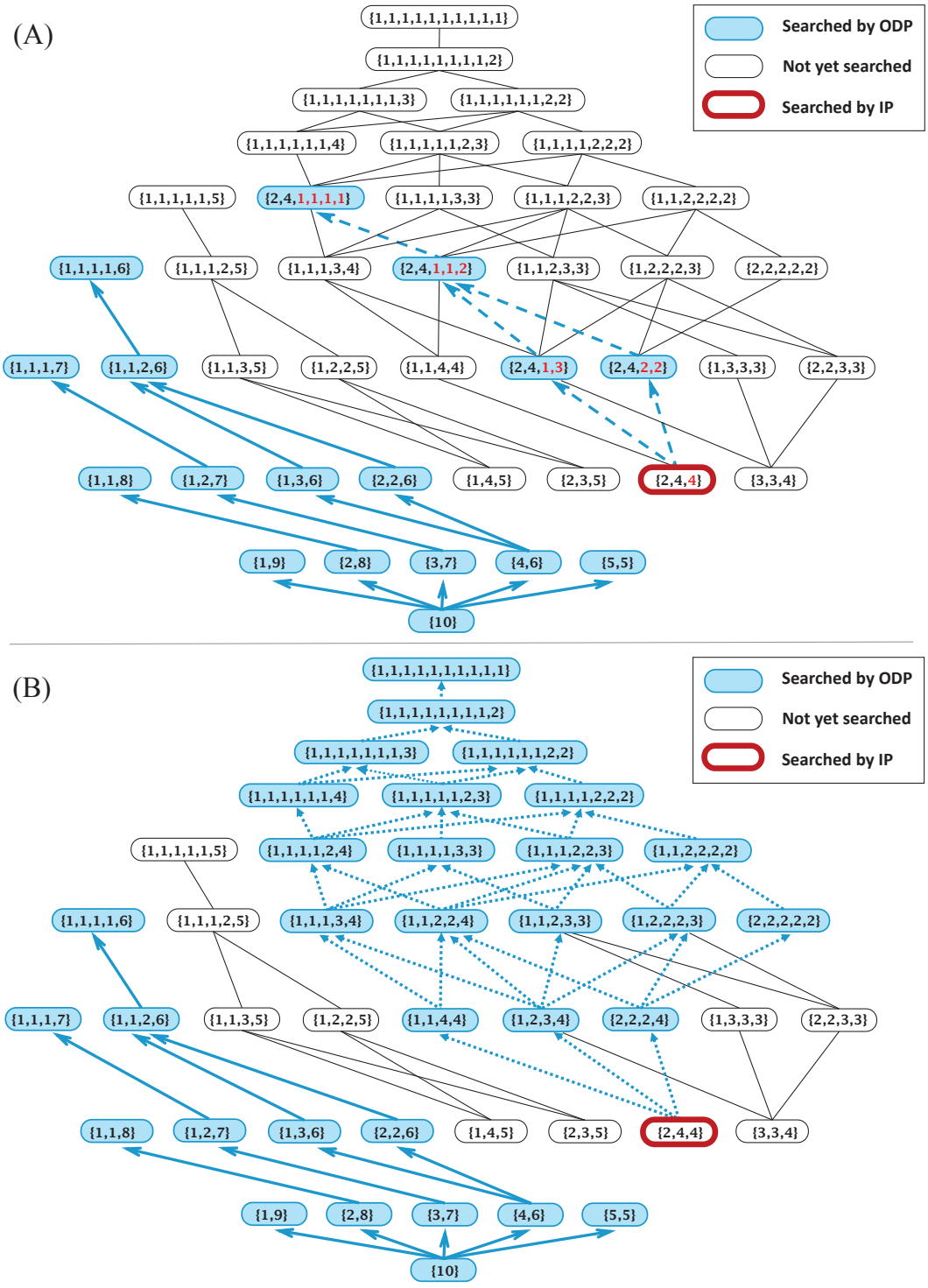



Figure 7: IP searching multiple subspaces simultaneously after ODP has computed $f_{M^{**}}(C)$ for $|C| \in \{2, 3, 4\}$. In Figure 7(A), several subspaces are searched simultaneously by splitting *exactly one* coalition. In Figure 7(B), more subspaces are searched simultaneously by splitting *multiple* coalitions.

Finally, note that the original IP algorithm ignores the order of the coalitions within a coalition structure. For instance, given a set of agents $A = \{a_1, \ldots, a_{10}\}$, the coalition structures $\{\{a_1, a_2\}, \{a_3, a_4, a_5, a_6\}, \{a_7, a_8, a_9, a_{10}\}\}$ and $\{\{a_1, a_2\}, \{a_7, a_8, a_9, a_{10}\}, \{a_3, a_4, a_5, a_6\}\}$ are considered to be the same, and so only one of the them is generated. However, when multiple subspaces are searched simultaneously, the order matters. For instance, 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 $v(C_1)+v(C_2)+f_{M^{**}}(C_3)$. 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 from $v(\{a_1, a_2\}) + v(\{a_7, a_8, a_9, a_{10}\}) + f_{M^{**}}(\{a_3, a_4, a_5, a_6\})$, and so both must be calculated.

### *5.5. Summary and Complexity Analysis 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:

- **Enable ODP to search subspaces of the integer partition graph**: To do this, ODP uses $f_{M^{**}}$ instead of $f_{M^*}$. Further, for $s = 2, \ldots, \lfloor \frac{2n}{3} \rfloor$, the algorithm: (1) evaluates all $m^{C',C''} \in M^{**}$ with $|C'| + |C''| = s$, (2) evaluates all $m^{C',C''} \in M^{**}$ with $\{|C'|, |C''|\} = \{s, n-s\}$, (3) updates $t(A)$, and (4) makes the best movements from $\{A\}$ using the function `getBestPartition`$(A, t(A))$. The pseudocode can be found in Algorithm 2.
- **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 $C_1, \ldots, C_d$.
- **Enable IP to search multiple subspaces simultaneously**: When searching a subspace $\Pi^A_I$, identify the integer partitions that are reachable from $I$ using the movements that have been evaluated by ODP thus far (e.g., see the dashed edges in Figures 7(A) and 7(B)). Now, let $I^* \subseteq I$ be the integers in $I$ that will be split to reach other integer partitions (e.g, $I^* = \{4\}$ in Figure 7(A) and $I^* = \{2, 4, 4\}$ in Figure7(B)). Then, for every coalition structure $CS \in \Pi^A_I$, evaluate every $C \in CS$ to $f_{M^{**}}(C)$ if the size of $C$ corresponds to an integer in $I^*$, otherwise evaluate it to $v(C)$. Finally, modify IP's branch-and-bound technique so 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 can be found in Appendix E.

We conclude this section with the following theorem, whose proof follows immediately from Corollary 18 and the observation that ODP-IP terminates as soon as one of ODP and IP does.

**Theorem 19.** *Given $n$ agents, ODP-IP runs in $O(3^n)$ time.*

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

## 6. Performance Evaluation

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

### *6.1. Evaluating ODP*

We know that DP evaluates $\frac{1}{2}(3^n + 1) - 2^n$ movements, while ODP evaluates $\frac{1}{2}\left(3^{n-1} - 1\right)$ movements (Corollary 13). In other words, ODP evaluates roughly 33% of the movements evaluated by ODP. Furthermore, we know that the size-based version of ODP (i.e., the version that is compatible with IP) evaluates $\frac{1}{2}3^{n-1} + o(3^n)$ movements (Corollary 18), i.e., in terms of performance it is more similar to ODP than to DP.

Figure 8 compares those numbers, with $n$ running from 5 to 40. It shows that, 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%. The reason behind the observed fluctuation is simply because the algorithm evaluates all splits of coalitions of size $\{1, 2, \ldots, \lfloor 2n/3 \rfloor\}$. Thus, the number of performed operations is influenced by the shape of the function $\lfloor 2n/3 \rfloor$.

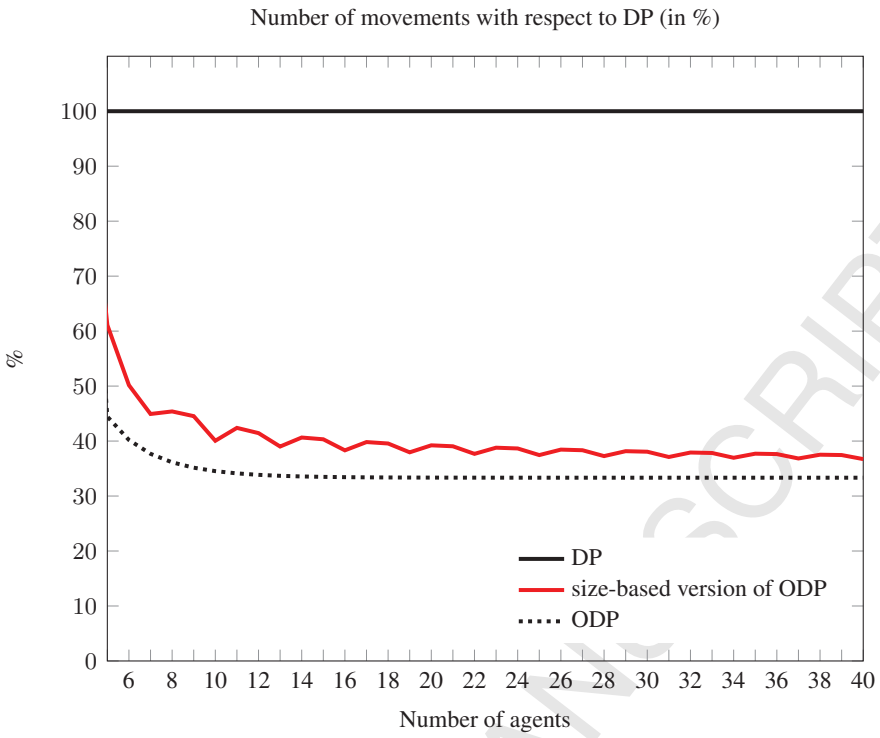



Figure 8: Percentage of movements evaluated by ODP and sb-ODP vs. DP

### 6.2. *Evaluating ODP-IP*

ODP-IP was developed in order to 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 (and hopefully faster than) the faster 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[9], 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. 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. Specifically, we consider the following distributions.

1. **Uniform**, as studied by Larson and Sandholm [17]: for all $C \in \mathcal{C}^A$, $v(C) \sim U(a, b)$, where $a = 0$ and $b = |C|$.
2. **Normal**, as studied by Rahwan et al. [30]: for all $C \in \mathcal{C}^A$, $v(C) \sim N(\mu, \sigma^2)$, where $\mu = 10 \times |C|$ and $\sigma = 0.1$.
3. **NDCS** (Normally Distributed Coalition Structures), as proposed by Rahwan et al. [32]: for all $C \in \mathcal{C}^A$, $v(C) \sim N(\mu, \sigma^2)$, where $\mu = |C|$ and $\sigma = \sqrt{|C|}$. The rationale behind developing NDCS came from the authors' observation that, with 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 this 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, under NDCS, all coalition structures are equally likely to be optimal.
4. **Modified Uniform**, as proposed by Service and Adams [40]: 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 probability 0.2.

[9]The open-source implementation is available at `https://github.com/trahwan/ODP-IP_and_InclusionExclusion`.

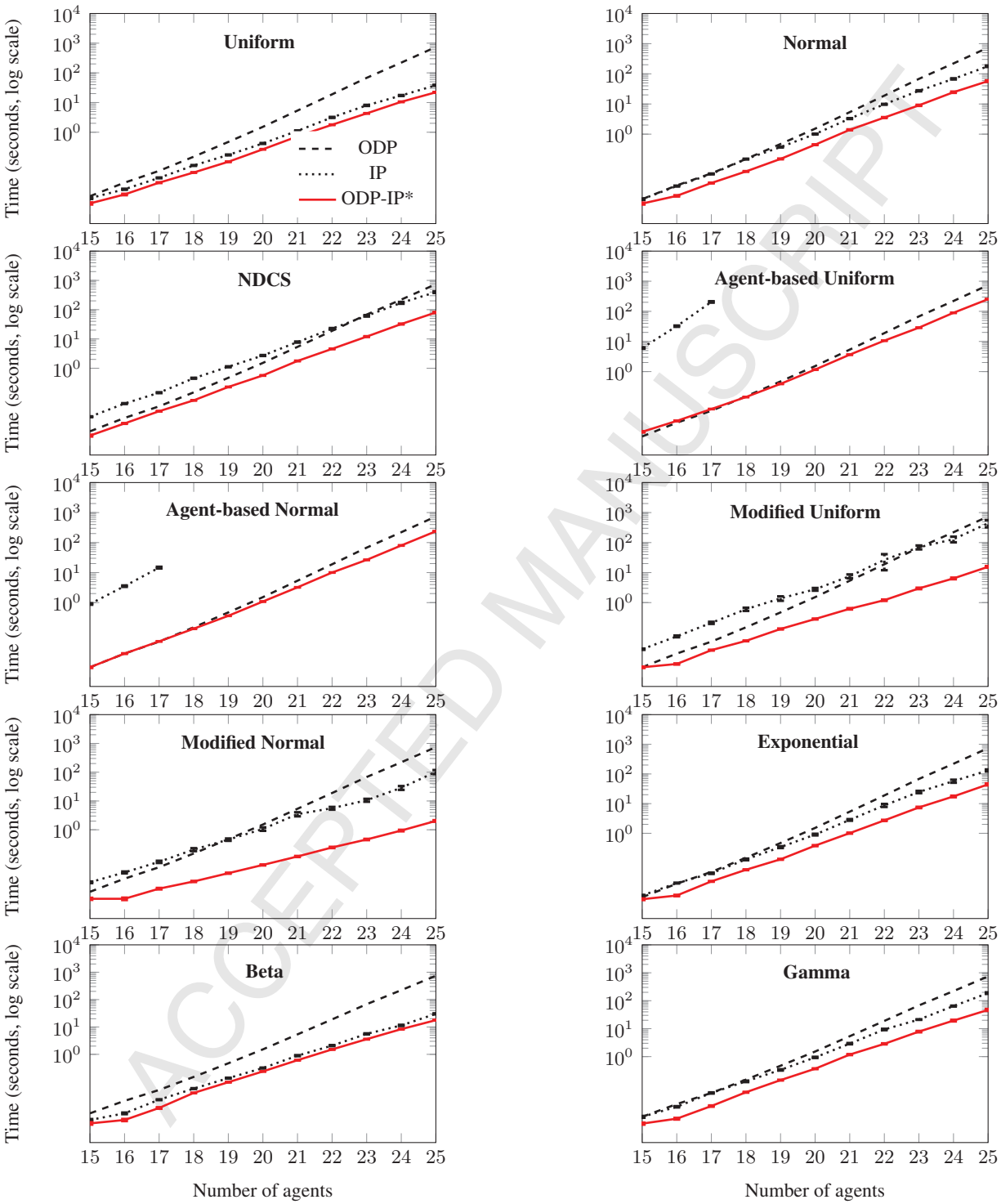



Figure 9: Time performance of ODP-IP vs. ODP and IP.

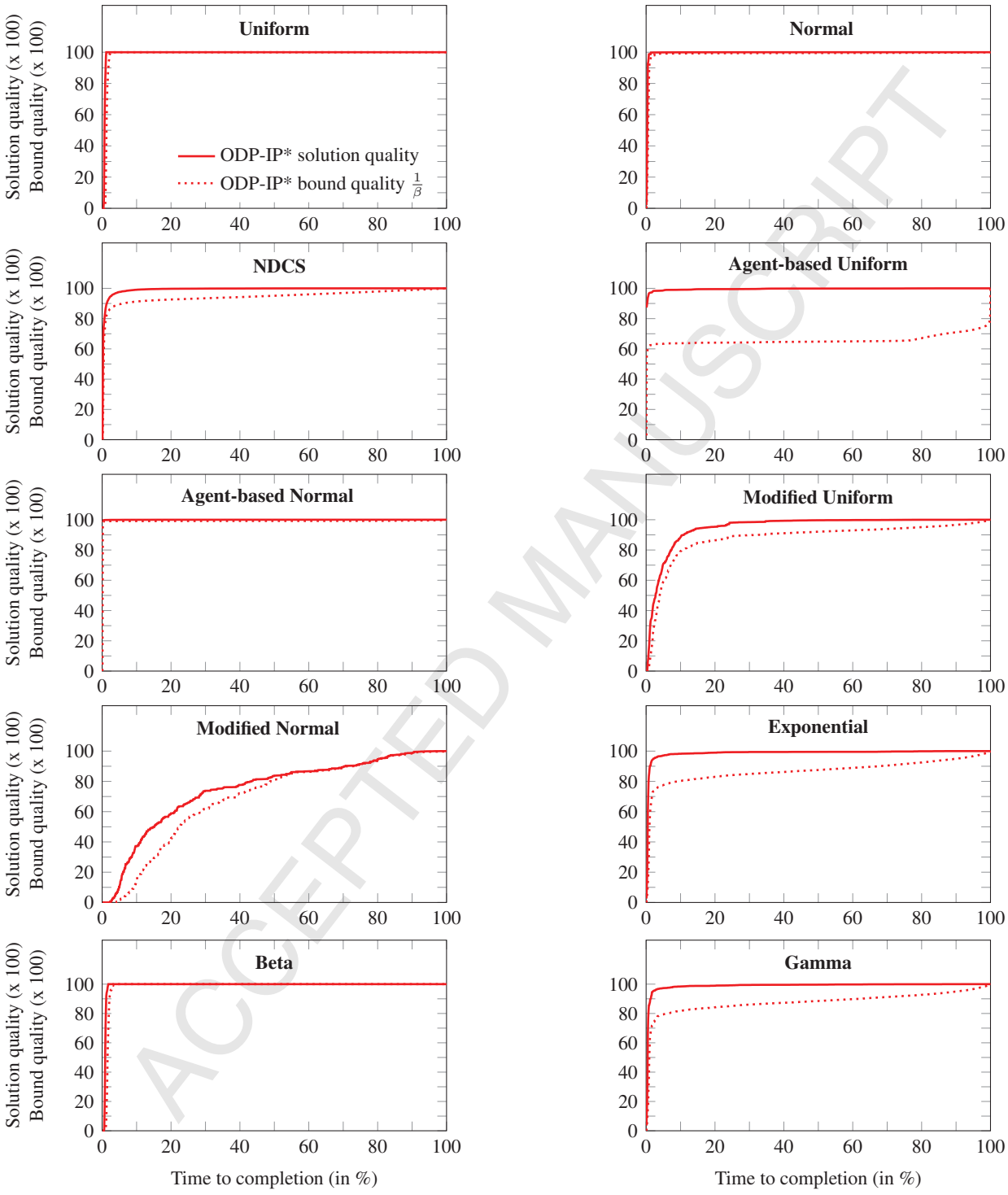



Figure 10: Solution quality and bound quality of ODP-IP.

5. **Modified Normal**, proposed by Rahwan et al. [34] as a natural counterpart to the Modified Uniform distribution. Under this distribution, each coalition's value is first drawn from $N(10 \times |C|, 0.01)$, and then increased by a random number $r \sim U(0, 50)$ with probability 0.2.
6. **Exponential:** for all $C \in \mathcal{C}^A$, $v(C) \sim |C| \times \text{Exp}(\lambda)$, where $\lambda = 1$.
7. **Beta:** for all $C \in \mathcal{C}^A$, $v(C) \sim |C| \times \text{Beta}(\alpha, \beta)$, where $\alpha = \beta = 0.5$.
8. **Gamma:** for all $C \in \mathcal{C}^A$, $v(C) \sim |C| \times \text{Gamma}(k, \theta)$, where $k = \theta = 2$.
9. **Agent-based Uniform**, as proposed by Rahwan et al. [34]: Under this distribution, each agent $a_i$ is assigned a random *"power"* $p_i \sim U(0, 10)$, reflecting its *average* performance over all coalitions. Then for every coalition $C \ni a_i$, the *actual* power of $a_i$ in $C$ is determined as $p_i^C \sim U(0, 2p_i)$, and a coalition's value is computed as the sum of the powers of its members. That is, for all $C \in \mathcal{C}^A$, $v(C) = \sum_{a_i \in C} p_i^C$.
10. **Agent-based Normal**, proposed in this article. As the name suggests, it is similar to the Agent-based Uniform distribution except that every agent's average and actual powers are drawn from normal, rather than uniform, distributions. Formally, for all $a_i \in A$, $p_i \sim N(10, 0.01)$ and for all $a_i$ and for all $C \subseteq A$ such that $a_i \in C$, $p_i^C \sim N(p_i, 0.01)$. Finally, for all $C \in \mathcal{C}^A$, $v(C) = \sum_{a_i \in C} p_i^C$.

For each of the above distributions, we plotted the termination times of ODP, IP, and ODP-IP given different numbers of agents (see Figure 9). Here, time is measured in seconds, and plotted *on a log scale*. For each distribution and each number of agents, we took an average over multiple runs; the number of runs was chosen to ensure tht the error bars are sufficiently small. 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 illustrates that the modifications introduced to IP and ODP (see Sections 5.2, 5.3, and 5.4) allow the two algorithms to help one another, leading to a positive synergy when they join forces as in ODP-IP. Observe that these modifications involve the use of branch-and-bound techniques, whose effectiveness depends heavily on the characteristic function at hand. Consequently, the resulting synergistic effect varies from one value distribution to another. This applies both to the termination time (as we have seen in Figure 9) and to the speed of improvement in the solution quality and established bounds during the runtime of ODP-IP (as we will see in the following figures).

Next, we evaluate the anytime property of ODP-IP. The results in Figure 10 are shown for 25 agents. The x-axis in the figures corresponds to the percentage of time that has elapsed, with 0% being the time when the algorithm starts, and 100% being the time when 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 is 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.

Next, we evaluate the effectiveness of the two main optimisation techniques in ODP-IP. In particular, Technique 1 improves the branch-and-bound approach used by IP (Section 5.3), whereas Technique 2 enables IP to search multiple subspaces simultaneously (Section 5.4). The results for 26 agents are shown in Table 1, where the shortest runtimes are highlighted in bold.[10] As can be seen, the effectiveness of each technique varies from one coalition-value distribution to another. Moreover, due to the overhead of those techniques, for some distributions the performance can actually be slower than simply running ODP and IP in parallel (see, e.g., the runtime for the agent-based distributions when Technique 1 is deactivated). However, when both techniques are activated, it is faster to run ODP-IP (sometimes by nearly two orders of magnitude, e.g., given the modified-normal distribution) than to run ODP or IP alone, or even run them both in parallel.

| | IP | DP | sb-ODP | ODP | ODP & IP in parallel | ODP-IP w/o Technique 1 | ODP-IP w/o Technique 2 | ODP-IP |
|---|---|---|---|---|---|---|---|---|
| NDCS | 1196 | 6127 | 3776 | 2206 | 1207 | 350 | 440 | **183** |
| Uniform | 76 | 6127 | 3776 | 2206 | 95 | 47 | 79 | **45** |
| Normal | 414 | 6127 | 3776 | 2206 | 442 | 142 | 322 | **127** |
| Agent-based Uniform | N/A | 6127 | 3776 | 2206 | 2786 | 3851 | 1061 | **880** |
| Agent-based Normal | N/A | 6127 | 3776 | 2206 | 2686 | 3737 | 1026 | **773** |
| Modified Uniform | 562 | 6127 | 3776 | 2206 | 419 | 304 | **17** | 31 |
| Modified Normal | 203 | 6127 | 3776 | 2206 | 263 | 56 | 4 | **3** |
| Beta | 54 | 6127 | 3776 | 2206 | 63 | 41 | 55 | **40** |
| Gamma | 437 | 6127 | 3776 | 2206 | 278 | **88** | 270 | 110 |
| Exponential | 346 | 6127 | 3776 | 2206 | 324 | 107 | 208 | **89** |
| Sum over all distributions | N/A | 61270 | 37760 | 22060 | 8563 | 8723 | 3482 | **2283** |

Table 1: Evaluating the effectiveness of two techniques of ODP-IP: Technique 1 improves the brand-and-bound of IP (Section 5.3), while Technique 2 enables IP to search multiple subspaces simultaneously (Section 5.4). The table shows runtime (in seconds) for 26 agents, taken for each coalition-value distribution as an average over 100 runs (error bars were relatively small, and were omitted to enhance readability).

Finally, observe that we do not benchmark our algorithms against integer-programming solvers. This is because Rahwan et al. [30] showed that even an industrial-strength solver such as ILOG's CPLEX is not suited for complete set partitioning, where matrices tend to be very dense. In particular, Rahwan et al. showed that CPLEX is much slower than IP (let alone ODP-IP), and that it runs out of memory with about 18 agents.

### *6.3. Benchmarking Against the Inclusion-Exclusion Algorithm*

In this section, we benchmark ODP-IP against the Inclusion-Exclusion algorithm of Björklund et al. [8]. As mentioned in the introduction, this is theoretically the state-of-the-art set partitioning algorithm in terms of worst-case runtime; it runs in time $O(2^n)$. In contrast, the running time of ODP-IP is $O(3^n)$. Thus, theoretically speaking, our algorithm should be significantly slower when solving a worst-case problem instance. Our goal in this section is to verify whether this happens in practice. Importantly, we test both algorithms *on a worst-case problem instance*, not on average instances.

[10] The run time of IP is not available for the agent-based distributions. This is because IP so ineffective with such distributions that its average run-time will actually take months to be computed; see Figure 9.

Thus, when we say "in practice", we do not mean "on average". Instead, we mean measuring the runtime on a PC, to account for any potential delays that were disregarded in the theoretical analysis.

We provide the pseudocode of the inclusion-exclusion algorithm in Appendix F, and provide the open-source Java implementation at `https://github.com/trahwan/ODP-IP_and_InclusionExclusion`. Figure 11 depicts the runtime of the algorithm on a log scale, given different numbers of agents. It also depicts the functions $y = 2^x$ and $y = 6^x$. As can be seen, the runtime growth rate resembles $6^n$, not $2^n$ (we had to extrapolate the results beyond 11 agents, as the runtime became extremely slow). The reason behind this delay is that the algorithm encodes information in extremely large numbers, which may contain hundreds, or even thousands of digits. To be more precise, for every coalition value, $v(C)$, the algorithm needs to use the following number: $(n^n)^{v(C)}$. Furthermore, $v(C)$ must be integer. Therefore, even if we use 64-bit integers, we can only handle cases of up to 6 agents with coalition values restricted to the set $\{0, 1, \ldots, 6\}$. We have considered using Matlab, which is slower, but at least allows for much larger numbers, compared to Java. However, the maximum number that can be represented in Matlab is $1.8 \cdot 10^{308}$, which means that we could handle at most 16 agents with coalition values restricted to the set $\{0, 1, \ldots, 16\}$. Thus, in our implementation, we used BigInteger—a Java class that allows for integers that are unbounded in length. However, the algorithm of Bjorklund et al. needs to perform many operations with these large numbers, resulting in huge delays that are hidden by the asymptotic analysis. As a result, even for 15 agents, the algorithm needs about $1.5 \cdot 10^{10}$ milliseconds (more than 5 months) to terminate. This is despite the fact that, in order to speed up the algorithm, we restrict the experiment to coalition values taken from the set $\{0, 1, \ldots, 9\}$, which is obviously very restrictive (a larger range would increase the number of required digits dramatically, resulting in a very significant slowdown). On the other hand, the runtime of ODP-IP cannot possibly be slower than that of ODP, even on worst-case instances. The runtime of ODP (which depends solely on the number of agents, and is not affected by any variations in coalition values, as long as these can be represented by floating-point numbers) is only 0.01 seconds given 15 agents.

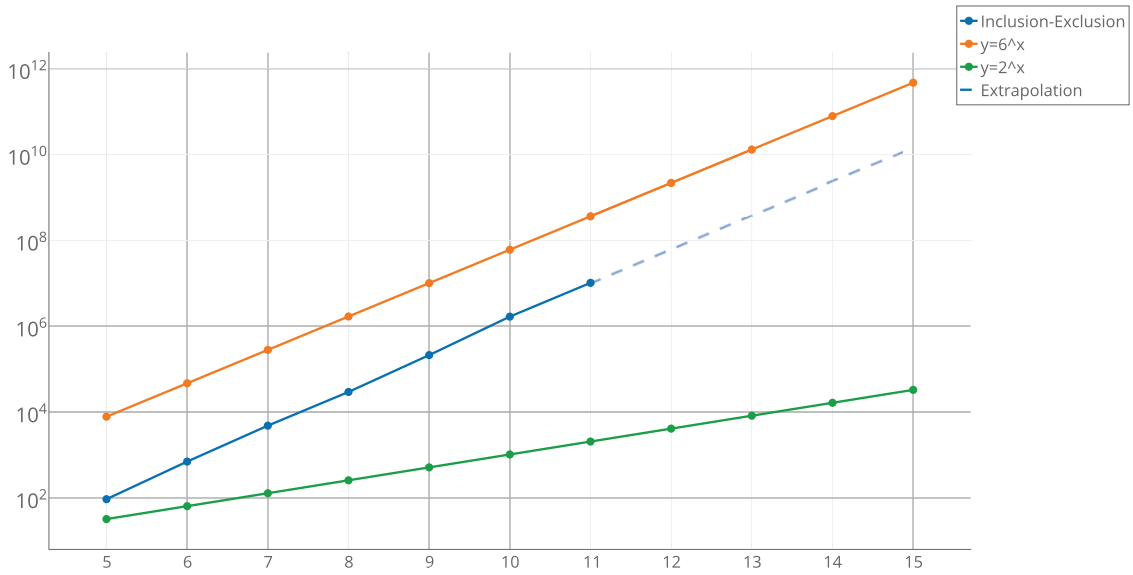



Figure 11: Runtime for the Inclusion-Exclusion algorithm of Björklund et al. [8] in milliseconds, on a log-scale, given different numbers of agents.

## 7. Related Work

The term *"complete set partitioning problem"* was introduced by Lin [20] for 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 optimisation problem, Lin and Salkin [21, 22] developed an integer programming algorithm with *branch search enumeration* [12] that runs in time $O(2^{n^2/2})$. Yeh [48] later showed that this algorithm is substantially slower than DP. The DP 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 [36]. Sandholm [37] provided further analysis of the complexity of this algorithm; in particular, he observed that its running time is polynomial in the size of the input (i.e., the number of possible subsets of goods). However, in contrast with our work, this analysis did not expose the redundant operations in DP.

The "complete" set partitioning problem differs from the "incomplete" version in terms of the input: in the complete version, the input consists of the values of all possible subsets, whereas in the incomplete version some values are listed explicitly, while others are assumed to be 0. Thus, the complete version usually involves *tens* of agents, with *billions* and billions of possible partitions. On the other hand, the incomplete version could involve, say, *thousands* of agents, and *thousands* of subsets. Thus, the complete version has a much larger, and much more structured, input. This is a fundamental difference, rendering some techniques effective for one version, and ineffective for the other. Consider CPLEX, for example. It is very effective on the incomplete version, but very quickly runs out of memory for the complete version [32].

The rapid growth of the multi-agent systems research community in the 1990s led to renewed 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 partitioning agents into coalitions so as to maximise the 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 criterion for dividing the search space into disjoint and exhaustive subspaces, and (2) identifying a sequence in which these subspaces should be searched, so that the worst-case bound on solution quality is guaranteed to improve after each subspace. We will denote the chosen sequence of subspaces by $S_1, \ldots, S_k$, and the bound established after searching $S_1 \cup \cdots \cup S_i$ by $\beta_i$. This bound is based solely on comparing the coalition structures that have already been considered against those that are yet to be considered (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 capitalise on the extra information accrued during the actual search, which can be used, 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 worse than $\beta_i$, $i = 1, \ldots, k$.

  The first algorithm in this class was put forward in the seminal article by Sandholm et al. [38], where the proposed sequence was $S_1 = \Pi^A_1 \cup \Pi^A_2$ and $S_i = \Pi^A_{n-i+2}$ for $i = 2, \ldots, n-1$. 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 suggested by Dang and Jennings [9], who 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 smaller number of solutions. Another algorithm was proposed by Rahwan et al. [33]; it represents every $S_i$ as a union of integer partition-based subspaces. Consequently, one can readily extend this algorithm by using IP (or ODP-IP) to search every $S_i$.

All the algorithms in this class discussed so far are proposed for characteristic function games, where there are no influences among co-existing coalitions. Rahwan et al. [31] proposed the first algorithm for the more general class of *partition function games* (PFGs), i.e., games with *externalities*. In such games, the value of a coalition depends on the coalition structure it appears in. Rahwan et al. focused on two sub-classes of partition function games: (1) PFG$^+$, where externalities are non-negative, and (2) PFG$^-$, where externalities are non-positive. Each of these two sub-classes is a generalisation of characteristic function games and, arguably, many realistic partition function games are either PFG$^+$ games or PFG$^-$ games. This algorithm was later on extended by Banerjee and Kraemer [5] 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 this merge 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 by PFG$^{type}$. Banerjee and Kraemer [5] argue that this class is intuitive, and maps to a number of applications.

- The second class of anytime, exact algorithms focuses on finding, and recognising, 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. [30, 32], which uses branch-and-bound techniques as described in Section 3.1. A distributed version of IP was later on proposed by Michalak et al. [23] as the first distributed, exact algorithm for coalition structure generation.

  Since the initial publication of ODP [28], an anytime version of the size-based version of ODP was proposed by Service and Adams [41]. 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. The authors showed how, using this extra information, every time the algorithm finishes evaluating the splits of all coalitions of a certain size $s$, it can construct a coalition structure whose value is guaranteed to be within a bound $r$ from optimal, 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 in Section 6.2 (see the difference in termination time 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 exceed 80% (or even 99%) after 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, as it has 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. Recall that in the presence of externalities, a 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 the solution quality without examining every single coalition structure. However, for two common classes of partition function games, namely PFG$^+$ and PFG$^-$, Rahwan et al. [32, 35] 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 [5] proposed an extension of IP to handle externalities in PFG$^{type}$ settings.

Another extension of ODP, which however is not anytime, is due to Voice et al. [46], who focus on the restricted coalition formation model proposed by Myerson [24]. In this model, 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 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 6 that, for any 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 significant 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 every coalition $C \subseteq A$ is returned by a characteristic function $v : 2^A \rightarrow \mathbb{R}$. However, one can also study the optimal coalition structure generation problem for alternative representations, which are designed to efficiently capture situations where the characteristic function has some structure. For instance, Ueda et al. [44] studied coalition structure generation under the DCOP (Distributed Constraint Optimisation Problem) representation (where every agent has a set of actions to choose from), while Bachrach et al. [4] and Bachrach et al. [3] studied it under the skill-game representation (where every agent has a set of skills required to perform tasks). Ohta et al. [25] studied coalition structure generation under the Marginal Contribution Nets representation of Ieong and Shoham [14], where synergies between agents are described by a (possibly small) collection of weighted logical formulas. Furthermore, Ueda et al. [45] and Aziz and de Keijzer [2] considereddd this problem under the agent-type representation (where agents are grouped into categories, or "types"). A common denominator of all these works is that the proposed algorithms for the coalition structure generation problem capitalise heavily on features of the underlying representation. As ODP-IP is a general-purpose algorithm, it is unlikely to outperform these algorithms on 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 using specific 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 are usually fast, and can therefore be applied when the number of agents is large. These include a greedy algorithm by Shehory and Kraus [42], a genetic algorithm by Sen and Dutta [39], a simulated-annealing algorithm by Keinänen [15], and an algorithm by Di Mauro et al. [11] that combines a greedy technique with another local-search technique.

## 8. Conclusions and Future Work

Our goal in this article was to provide extensive theoretical analysis of the search space of the Complete Set Partitioning problem and to improve upon two fundamentally-different exact algorithms, namely DP and IP. We drew a link between the workings of DP and the coalition structure graph, which revealed that many of DP's operations are redundant. Building upon this observation, we developed ODP—an optimal version of DP that avoids *all* redundant operations.

Although ODP 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). The community can benefit from the open-source implementation, which is made publicly available.[11]

While the focus in this article was on settings where there are no influences (or externalities) among co-existing coalitions, it would be interesting to see whether the underlying techniques of ODP-IP can be extended to settings with externalities, and to identify conditions under which such an extension can be efficient (in the spirit of Rahwan et al. [35]).

[11] The implementations used as part of this research (namely that of IP, DP, ODP, ODP-IP, and the inclusion-exclusion algorithm) are available at the following link: `https://github.com/trahwan/ODP-IP_and_InclusionExclusion`.

## 9. Acknowledgments

Part of the research presented 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. Tomasz Michalak and Michael Wooldridge were supported by the European Research Council under Advanced Grant 291528 ("RACE"). Edith Elkind was partially supported by the National Research Foundation (Singapore) under grant NRF-RF2009-08 and by the European Research Council under Starting Grant 639945 ("ACCORD"). This article is a significantly revised and extended version of the following papers: [28], [27], and [34]. More specifically:

- While the basic idea of ODP was presented in the short paper [28], it did not include Theorems 8, 12, and 14, which are essential for proving the correctness of ODP. Furthermore, paper [28] did not include the optimal version of DP, where *all* redundant operations are removed. The aforementioned theorems, as well as the optimal version of DP, are presented for the first time in the current article.

- While the basic idea of combining ODP and IP was presented in paper [27], 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 choose the value of $m$. Furthermore, the worst-case complexity was greater than $O(3^n)$. Finally, there were cases where the hybrid performance was slower than that of 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 time $O(3^n)$. 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 E). As a result, our hybrid is always faster than its constituent parts.

- While the idea of running ODP and IP in parallel has appeared in paper [34], it did not include the technique for searching several subspaces simultaneously. Furthermore, the evaluation section was limited in that it did not show how the bound and solution quality improves over the runtime of the algorithm.

## References

[1] G. E. Andrews and K. Eriksson. *Integer Partitions*. Cambridge University Press, Cambridge, UK, 2004. ISBN 0521600901.

[2] H. Aziz and B. de Keijzer. Complexity of coalition structure generation. In *Proceedings of the 10th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2011)*, pages 191–198, 2011.

[3] Y. Bachrach, R. Meir, K. Jung, and P. Kohli. Coalitional structure generation in skill games. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-2010)*, pages 703–708, 2010.

[4] Y. Bachrach, D. Parkes, and J. S. Rosenschein. Computing cooperative solution concepts in coalitional skill games. *Artificial Intelligence*, 204:1–21, 2013.

[5] B. Banerjee and L. Kraemer. Coalition structure generation in multi-agent systems with mixed externalities. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2010)*, pages 175–182, 2010.

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

[7] E. Y. Bitar, E. Baeyens, P. P. Khargonekar, K. Poolla, and P. Varaiya. Optimal sharing of quantity risk for a coalition of wind power producers facing nodal prices. In *Proceedings of the 31st IEEE American Control Conference (ACC-2012)*, pages 4438–4445, 2012.

[8] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. ISSN 0097-5397.

[9] V. D. Dang and N. R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004)*, pages 564–571, 2004.

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

[11] N. Di Mauro, T. M. A. Basile, S. Ferilli, and F. Esposito. Coalition structure generation with GRASP. In *Proceedings of the 14th International Conference on Artificial Intelligence: Methodology, Systems, and Applications (AIMSA-2010)*, pages 111–120, 2010.

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

[13] Z. Han and H. V. Poor. 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, 2009.

[14] S. Ieong and Y. Shoham. Marginal contribution nets: a compact representation scheme for coalitional games. In *Proceedings of the 6th ACM Conference on Electronic Commerce (ACM EC-2005)*, pages 193–202, 2005.

[15] H. Keinänen. Simulated annealing for multi-agent coalition formation. In *Proceedings of the 3rd KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA-2009)*, pages 30–39, 2009.

[16] Z. Khan, J. Lehtomaki, M. Latva-Aho, and L. A. DaSilva. On selfish and altruistic coalition formation in cognitive radio networks. In *Proceedings of the 5th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM-2010)*, pages 1–5, 2010.

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

[18] D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*, pages 297–317. MIT Press, 2006.

[19] C. Li, K. Sycara, and A. Scheller-Wolf. Combinatorial coalition formation for multi-item group-buying with heterogeneous customers. *Decision Support Systems*, 49(1):1–13, April 2010. ISSN 0167-9236. doi: 10.1016/j.dss.2009.12.002.

[20] C. H. Lin. *Corporate tax structures and a special class of set partitioning problems*. PhD thesis, Department of Operations Research, Case Western Reserve University, 1975.

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

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

[23] T. Michalak, J. Sroka, T. Rahwan, M. Wooldridge, P. McBurney, and N. R. Jennings. A distributed algorithm for anytime coalition structure generation. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2010)*, pages 1007–1014, 2010.

[24] R. Myerson. Graphs and cooperation in games. *Mathematics of Operations Research*, 2(3):225–229, 1977.

[25] N. Ohta, V. Conitzer, R. Ichimura, Y. Sakurai, A. Iwasaki, and M. Yokoo. Coalition structure generation utilizing compact characteristic function representations. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP-2009)*, pages 623–638, 2009.

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

[27] T. Rahwan and N. R. Jennings. Coalition structure generation: Dynamic programming meets anytime optimisation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-2008)*, pages 156–161, 2008.

[28] T. Rahwan and N. R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2008)*, pages 1417–1420, 2008.

[29] T. Rahwan, S. D. Ramchurn, V. D. Dang, and N. R. Jennings. Near-optimal anytime coalition structure generation. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, pages 2365–2371, 2007.

[30] T. Rahwan, S. D. Ramchurn, A. Giovannucci, V. D. Dang, and N. R. Jennings. Anytime optimal coalition structure generation. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-2007)*, pages 1184–1190, 2007.

[31] T. Rahwan, T. Michalak, N. R. Jennings, M. Wooldridge, and P. McBurney. Coalition structure generation in multi-agent systems with positive and negative externalities. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI-2009)*, pages 257–263, 2009.

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

[33] T. Rahwan, T. Michalak, and N. R. Jennings. Minimum search to establish worst-case guarantees in coalition structure generation. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pages 338–343, 2011.

[34] T. Rahwan, T. Michalak, and N. R. Jennings. A hybrid algorithm for coalition structure generation. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-2012)*, 2012.

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

[36] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44 (8):1131–1147, 1995.

[37] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1–2):1–54, 2002.

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

[39] S. Sen and P. Dutta. Searching for optimal coalition structures. In *Proceedings of the 6th International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 286–292, 2000.

[40] T. C. Service and J. A. Adams. Approximate coalition structure generation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 854–859, 2010.

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

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

[43] R. P. Stanley. *Enumerative Combinatorics, Vol. 1*. Cambridge University Press, Cambridge, UK, 1997.

[44] S. Ueda, A. Iwasaki, M. Yokoo, M. C. Silaghi, K. Hirayama, and T. Matsui. Coalition structure generation based on distributed constraint optimization. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-2010)*, pages 197–203, 2010.

[45] S. Ueda, M. Kitaki, A. Iwasaki, and M. Yokoo. Concise characteristic function representations in coalitional games based on agent types. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-2011)*, pages 393–399, 2011.

[46] T. Voice, S. D. Ramchurn, and N. R. Jennings. On coalition formation with sparse synergies. In *Proceedings of the 11th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2012)*, pages 223–230, 2012.

[47] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial optimization - Eureka, you shrink!*, pages 185–207. Springer-Verlag, New York, NY, USA, 2003.

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

## Appendix A. Summary of Notation

| | |
|---|---|
| $A$ | the set of agents |
| $a_i$ | an agent in $A$ |
| $n$ | the number of agents in $A$ |
| $\mathcal{C}^A$ | the set of all coalitions over $A$ |
| $\mathcal{C}_s^A$ | the set of all size-$s$ coalitions over $A$ |
| $C$ | a coalition |
| $CS$ | a coalition structure |
| $CS^*$ | an optimal coalition structure |
| $CS^{**}$ | the best coalition structure found at a given point in time (i.e., the current best solution) |
| $\beta$ | the established bound on the quality of $CS^{**}$, i.e., an upper bound on $\frac{V(CS^*)}{V(CS^{**})}$ |
| $\mathcal{I}^n$ | the set of all integer partitions of $n$ |
| $I$ | an integer partition, i.e., a multiset of integers |
| $\Pi^A$ | the set of all coalition structures over $A$ |
| $\Pi_s^A$ | the set of all coalition structures over $A$ that consist of exactly $s$ coalitions |
| $\Pi_I^A$ | the set of all coalition structures over $A$ in which coalition sizes match the parts in the integer partition $I$ |
| $\Pi^C$ | the set of all partitions of $C$ |
| $\Pi_s^C$ | the set of all partitions of $C$ that have exactly $s$ parts |
| $\Pi_I^C$ | the set of all partitions of $C$ in which set sizes match the parts in the integer partition $I$ |
| $\Pi$ | the set of all possible partitions, i.e., $\Pi = \cup_{C \subseteq A} \Pi^C$ |
| $\pi$ | a partition, i.e., a set of pairwise disjoint coalitions |
| $V(CS)$ | the value of the coalition structure $CS$ |
| $V(\pi)$ | the value of the partition $\pi$ |
| $v(C)$ | the value of the coalition $C$ |
| $Max_s$ | the maximum value among all coalitions of size $s$ |
| $Avg_s$ | the average value of all coalitions of size $s$ |
| $UB^*$ | an upper bound on the value of $CS^*$ |
| $UB_I$ | an upper bound on the value of the best coalition structure in $\Pi_I^A$ |
| $LB^*$ | a lower bound on the value of $CS^*$ |
| $LB_I$ | a lower bound on the value of the best coalition structure in $\Pi_I^A$ |
| $\mathcal{M}$ | 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 set of movements in $\mathcal{M}$ that correspond to splitting a coalition of size $s'+s''$ into two coalitions of sizes $s'$ and $s''$, respectively |
| $m^{C_1,C_2}$ | the movement that corresponds to splitting $C = C_1 \cup C_2$ into $C_1$ and $C_2$ |
| $R_M^\pi$ | the set of all partitions that are reachable from $\pi$ via $M$ |
| $f(C)$ | the value of an optimal partition of $C$ |
| $f_M(C)$ | the value of a partition with the highest value in $R_M^{\{C\}}$ |

## Appendix B. Pseudocode of the ODP-IP Algorithm

Algorithm 3 provides the pseudocode of ODP, while Algorithm 4 provides the pseudocode of `getBestPartition`—the function used in line 32 of Algorithm 3. Here, we use a hash table to access the characteristic function, where a coalition $C$ acts as the key to the entry containing $v(C)$. While this is clearly the most efficient data structure, for large problem instances the available memory might not be sufficient, in which case other data structures must be explored. However, we did not consider any such alternatives in this article.

We remark that, to save memory, we avoid storing the table $t$, i.e., we only store $f_{M^*}(C)$ for all $C \subseteq A$, and then recompute $t$ on-the-fly (see Algorithm 4). Observe that we need to compute the value of $t$ for at most $2n-1$ coalitions during each execution of the algorithm, and, given the values of $f(C')$ for all $C' \subseteq C$, we can compute $t(C)$ in time $2^{|C|}$. Thus, this modification requires less than $(2n-1) \times 2^n$ additional operations, which is negligible compared to $O(3^n)$—the runtime of ODP. In return, the algorithm only needs to store the characteristic function, $v$, instead of storing both $v$ and $t$, resulting in 50% reduction in memory requirements compared to DP.

## Appendix C. Proofs for Section 4

**Theorem 6.** *For every coalition $C \subseteq A$ and for every subset of movements $M \subseteq \mathcal{M}$ it holds that*

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

*Proof.* If $|C| = 1$, then no movement can be made from $\{C\}$. This means that $R_M^{\{C\}} = \{\{C\}\}$ and hence $f_M(C) = v(C)$.

Now, suppose that $|C| > 1$. Consider a partition $\{C', C''\} \in \Pi^C$ and a set of movements $M \subseteq \mathcal{M}$. Abusing notation, set

$$R_M^{\{C'\}} \times R_M^{\{C''\}} = \left\{\pi_1 \cup \pi_2 : \pi_1 \in R_M^{\{C'\}}, \pi_2 \in R_M^{\{C''\}}\right\}.$$

Observe that

$$R_M^{\{C',C''\}} = R_M^{\{C'\}} \times R_M^{\{C''\}}. \tag{C.1}$$

Since $\{C\}$ contains exactly one coalition, namely, $C$, every partition reachable from $\{C\}$ via a single movement in $M$ contains exactly two coalitions. Conversely, every partition in $R_M^{\{C\}}$ that contains exactly two coalitions is reachable from $\{C\}$ via at most one movement in $M$. Thus,

$$\left\{\pi' \in \Pi : \{C\} \xrightarrow{M} \pi'\right\} = \left\{\{C', C''\} \in R_M^{\{C\}}\right\}. \tag{C.2}$$

Hence, we have

$$\begin{aligned} R_M^{\{C\}} &= \{\{C\}\} \cup \bigcup_{\pi'\in\Pi:\{C\}\xrightarrow{M}\pi'} R_M^{\pi'} \\ &= \{\{C\}\} \cup \bigcup_{\{C',C''\}\in R_M^{\{C\}}} R_M^{\{C',C''\}} \\ &= \{\{C\}\} \cup \bigcup_{\{C',C''\}\in R_M^{\{C\}}} \left(R_M^{\{C'\}} \times R_M^{\{C''\}}\right), \end{aligned}$$

where the first equality is based on (3), the second equality is based on (C.2), and the last equality is based on (C.1).

**ALGORITHM 3:** The Optimal Dynamic Programming algorithm (ODP).

**Input**: $v(C)$ for all $C \subseteq A$.
**Output**: an 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*}({a_1, a_2}) < v({a_1}) + v({a_2}) then
    f_{M*}({a_1, a_2}) ← v({a_1}) + v({a_2})
// Third, compute f_{M*}(C) for every coalition of size s = 3, ..., n − 1
for s = 3 to n − 1 do
    foreach S ⊆ A \ {a_1, a_2} : |S| = s − 2 do // This is an efficient way of evaluating the splits of C into C', C'' such that: C' < C'' < A \ (C' ∪ C''). It only evaluates (some of) the splits of a coalition C that contains both a_1 and a_2. Here, S denotes the C after removing a_1 and a_2 from it.
        C ← S ∪ {a_1, a_2} // 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 ← {a_3, ..., a_{j−1}}
            if f_{M*}(C) < v(S' ∪ {a_1}) + v(S'' ∪ {a_2}) then
                f_{M*}(C) ← v(S' ∪ {a_1}) + v(S'' ∪ {a_2})
            if f_{M*}(C) < v(S' ∪ {a_2}) + v(S'' ∪ {a_1}) then
                f_{M*}(C) ← v(S' ∪ {a_2}) + v(S'' ∪ {a_1})
            if S' ∩ A^j ≠ ∅ then // to ensure that S'' ∪ {a_1, a_2} < S' < A \ C
                if f_{M*}(C) < v(S') + v(S'' ∪ {a_1, a_2}) then
                    f_{M*}(C) ← v(S') + v(S'' ∪ {a_1, a_2})
            if S'' ∩ A^j ≠ ∅ then // to ensure that S' ∪ {a_1, a_2} < S'' < A \ C
                if f_{M*}(C) < v(S' ∪ {a_1, a_2}) + v(S'') then
                    f_{M*}(C) ← v(S' ∪ {a_1, a_2}) + v(S'')
    foreach C ⊆ A : |C| = s, {a_1, a_2} ⊈ 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 pseudocode in Algorithm 4
return CS*
```

**ALGORITHM 4:** `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**: a best partition of $C$ that is reachable via $M^*$, i.e., a best partition in $R_{M^*}^{\{C\}}$.

```
// 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 ← ∅ // initialisation
    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} 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
```

Consequently, we obtain

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

□

**Corollary 13.** *The number of movements in $\mathcal{M}$ is $\frac{1}{2}\left(3^n+1\right) - 2^n$, whereas the number of movements in $M^*$ is $\frac{1}{2}\left(3^{n-1}-1\right)$.*

*Proof.* For $\mathcal{M}$, the argument is essentially provided by Yeh [48]; we reproduce it here for completeness. For each coalition $C$ of size $k$, $2 \leq k \leq n$, there are $2^{k-1}-1$ ways of splitting $C$ into two non-empty coalitions, so DP evaluates $2^{k-1}-1$ movements from $C$. Thus, the total number of movements evaluated by DP can be written as

$$\sum_{k=2}^{n} \binom{n}{k} \left(2^{k-1}-1\right) = \sum_{k=0}^{n} \binom{n}{k} 2^{k-1} - \frac{1}{2} - n - \sum_{k=0}^{n} \binom{n}{k} + 1 + n.$$

Using the fact that

$$(1+x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k$$

with $x = 2$ and $x = 1$, we conclude that DP evaluates $\frac{1}{2}(3^n + 1) - 2^n$ movements.

We will now consider $M^*$. By Theorem 12, we have

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

Now, recall that the number of ways to partition $n$ elements into $k$ parts—known as *the Stirling number of the second kind* [43], and denoted by $S(n,k)$—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 D. Proofs for Section 5

**Theorem 15.**

$$R_{M^{**}}^{\{A\}} = \Pi^A. \tag{10}$$

*Proof.* We will show that every coalition structure $CS$ with $|CS| \geq 2$ is reachable via $M^{**}$ from some other coalition structure $CS'$ with $|CS'| = |CS| - 1$. To this end, assume without loss of generality that $CS = \{C_1, \ldots, C_k\}$, where $k \geq 2$ and $|C_1| \leq \cdots \leq |C_k|$. We will argue that $m^{C_1,C_2} \in M^{**}$, and hence $CS$ can be reached from the coalition structure $(CS \backslash \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$ via $M^{**}$. Let $s_1 = |C_1|$, $s_2 = |C_2|$; by construction, we have $m^{C_1,C_2} \in M^{s_1,s_2}$.

First, suppose that $k = 2$. In this case, we have $CS = \{C_1, C_2\}$, and so $s_1 + s_2 = n$. This means that $M^{s_1,s_2}$ is a subset of $M^{**}$ (see equation (9)).

Now, suppose that $k > 2$. We have $|C_1| \leq |C_2| \leq |C_3| \leq n - |C_1| - |C_2|$, so $\max\{s_1, s_2\} = s_2 \leq n - s_1 - s_2$. This means that $M^{s_1,s_2}$ is a subset of $M^{**}$ in this case as well (again, see equation (9)). □

**Lemma 16.** *The $DP_{M^{**}}$ algorithm does not evaluate any of the possible ways of splitting a coalition of size $s$, where $s \in \{\lfloor \frac{2n}{3} \rfloor + 1, \ldots, n-1\}$.*

*Proof.* We will prove that for every $s \in \{\lfloor \frac{2n}{3} \rfloor + 1, \ldots, n-1\}$ and for all $s', s'' \in \mathbb{Z}^+$ such that $s' + s'' = s$ it holds that

$$\lceil s/2 \rceil > n - s. \tag{D.1}$$

This immediately implies our claim: since $s' + s'' = s$, we have $\max\{s', s''\} \geq \lceil s/2 \rceil > n - s$ and hence $M^{s',s''} \cap M^{**} = \emptyset$. Since the expression $\lceil s/2 \rceil + s$ is monotone in $s$, it suffices to prove equation (D.1) for the smallest value of $s$ in $\{\lfloor \frac{2n}{3} \rfloor + 1, \ldots, n-1\}$, i.e., for $s_0 = \lfloor \frac{2n}{3} \rfloor + 1$. We have $s_0 > \frac{2n}{3}$, so $\lceil \frac{s_0}{2} \rceil \geq \frac{s_0}{2} > \frac{n}{3}$, and thus $\lceil \frac{s_0}{2} \rceil + s_0 > \frac{n}{3} + \frac{2n}{3} = n$. Rearranging, we obtain $\lceil \frac{s_0}{2} \rceil > n - s_0$, which is what we wanted to prove. □

**Theorem 17.** *The number of movements in $M^{**}$ is $\frac{1}{2}3^{n-1} + o(3^n)$.*

*Proof.* Let $\widehat{M} = \{m^{C',C''} : C' \cup C'' \subset A\} \cap M^{**}$, and consider the mapping $\alpha : \widehat{M} \to \Pi_3^A$ given by

$$\alpha(m^{C',C''}) = \{C', C'', A \setminus (C' \cup C'')\}.$$

Note that by construction of $\widehat{M}$, for each $m^{C',C''} \in \widehat{M}$ the set $A \setminus (C' \cup C'')$ is not empty, so $\alpha(m^{C',C''})$ is indeed an element of $\Pi_3^A$, i.e., a partition of $A$ into three non-empty parts.

To prove the theorem, we will show that (1) $|M^{**} \setminus \widehat{M}| = o(3^n)$ and (2) $|\widehat{M}| = \frac{1}{2}3^{n-1} + o(3^n)$. Taken together, these claims show that

$$|M^{**}| = |\widehat{M}| + |M^{**} \setminus \widehat{M}| = \frac{1}{2}3^{n-1} + o(3^n) + o(3^n) = \frac{1}{2}3^{n-1} + o(3^n),$$

which is what we want to prove.

The first of these claims is immediate: we have

$$|M^{**} \setminus \widehat{M}| = |\{m^{C',C''} : C' \cup C'' = A\}| = 2^{n-1} - 1 = o(3^n).$$

To prove the second claim, we will show that for almost all coalition structures in $\Pi_3^A$ their pre-image under $\alpha$ consists of exactly one movement, and for the remaining coalition structures in $\Pi_3^A$ their pre-image under $\alpha$ contains at most 3 movements. To complete the proof, we then use the fact that

$$|\Pi_3^A| = S(n,3) = \frac{1}{2}\left(3^{n-1} - 2^n + 1\right) = \frac{1}{2}3^{n-1} + o(3^n)$$

(see the proof of Corollary 13).

In more detail, consider a coalition structure $CS = \{C_1, C_2, C_3\}$ in $\Pi_3^A$, let $s_1 = |C_1|$, $s_2 = |C_2|$, $s_3 = |C_3|$, and assume without loss of generality that $s_1 \leq s_2 \leq s_3$. If $s_2 < s_3$, then the only element of $\widehat{M}$ that is mapped to $CS$ by $\alpha$ is $m^{C_1,C_2}$: indeed, we have $m^{C_1,C_3} \notin \widehat{M}$, $m^{C_2,C_3} \notin \widehat{M}$. If $s_1 < s_2 = s_3$, then there are two elements of $\widehat{M}$ that are mapped to $CS$ by $\alpha$, namely, $m^{C_1,C_2}$ and $m^{C_1,C_3}$. Finally, if $s_1 = s_2 = s_3$, then there are three elements of $\widehat{M}$ that are mapped to $CS$ by $\alpha$, namely, $m^{C_1,C_2}$, $m^{C_1,C_3}$, and $m^{C_2,C_3}$.

Let

$$\begin{aligned} X &= \{\{C_1, C_2, C_3\} \in \Pi_3^A : |C_1| \leq |C_2| < |C_3|\}, \\ Y &= \{\{C_1, C_2, C_3\} \in \Pi_3^A : |C_1| < |C_2| = |C_3|\}, \\ Z &= \{\{C_1, C_2, C_3\} \in \Pi_3^A : |C_1| = |C_2| = |C_3|\}, \end{aligned}$$

and set $x = |X|$, $y = |Y|$, $z = |Z|$. Since every movement in $\widehat{M}$ is mapped to some coalition structure in $\Pi_3^A$ by $\alpha$, the argument above shows that $|\widehat{M}| = x + 2y + 3z \leq (x + y + z) + 2(y + z) = |\Pi_3^A| + 2(y + z)$. We have observed that $|\Pi_3^A| = S(n,3) = \frac{1}{2}3^{n-1} + o(3^n)$. Thus, to complete the proof, it suffices to show that $y + z = o(3^n)$.

Note that every coalition structure $\{C_1, C_2, C_3\} \in Y \cup Z$ has the property that $|C_1| \leq |C_2| = |C_3|$. To obtain such a coalition structure, we can first choose the coalition $C_1$, whose size is at most $n/3$, and then partition the remaining elements into two sets of equal size. Equivalently, we can first choose which elements will appear in $C_2 \cup C_3$, and then partition the selected elements into two sets of equal size; note that the number of elements selected at the first step needs to be even and cannot be less than $2\lfloor \frac{n}{3} \rfloor$. It follows that

$$\begin{aligned} y + z &\leq \sum_{k=\lfloor \frac{n}{3} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k}\binom{2k}{k} \\ &\leq \sum_{k=\lfloor \frac{n}{3} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k}\frac{2^{2k}}{\sqrt{\pi k}}(1 + o(1)) \\ &\leq \frac{1}{\sqrt{\pi \lfloor \frac{n}{3} \rfloor}}(1 + o(1)) \sum_{k=\lfloor \frac{n}{3} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k}2^{2k} \\ &\leq \frac{1}{\sqrt{\pi \lfloor \frac{n}{3} \rfloor}}(1 + o(1)) \sum_{j=0}^{n} \binom{n}{j}2^{j} \\ &= \frac{3^n}{\sqrt{\pi \lfloor \frac{n}{3} \rfloor}}(1 + o(1)) = o(3^n). \end{aligned}$$

This completes the proof. □

## Appendix E. Analysing 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.

Assume that ODP has already finished evaluating all movements $m^{C,C'} \in M^{**}$ with $|C|+|C'| \in \{2,\dots,s^*\}$. Then, for any given subspace $\Pi_I^A$ with $I=\{i_1,\dots,i_k\}$, we modify IP so that, instead of searching for a coalition structure in $\arg\max_{CS\in\Pi_I^A} V(CS)$, it performs the following steps:

1. **Identify $\mathcal{X}^*$**—the set of all integer partitions whose corresponding subsets have not yet been searched and are reachable from $\Pi_I^A$ 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 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 in Section 5.4, one can choose either to split a single integer in $I$ or to split multiple integers at once. We will consider both cases. Specifically, if exactly one integer will be split, pick an integer $s\in I$ so that splitting $s$ allows for reaching *the largest number of integer partitions in* $\mathcal{X}^*$, and set $I^*=\{s\}$. On the other hand, if multiple integers will be split, choose $I^*$ so that splitting the integers in $I^*$ 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). On the other hand, 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, followed by $i_1^*$, then $i_2^*$, and so on until $i_{|I^*|}^*$. Similarly, for every $I'\in\mathcal{Y}^*$, change the order of integers 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 $\Pi_I^A$, where every $\{C_1,\dots,C_k\}\in\Pi_I^A$ is evaluated as follows:**

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

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

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

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

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

The result of this search is a coalition structure $\{C_1^*,\dots,C_k^*\}\in\Pi_I^A$ that maximises (E.1).

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

At first glance, it may seem that partitioning multiple integers is better than partitioning a single integer, because the former approach enables us to search more subspaces simultaneously. Surprisingly, however, it can actually be faster to partition one integer only. We will now explain why this may be the case.

The difficulty with splitting multiple integers is that it may interfere with our branch-and-bound technique. Specifically, recall that when we search subspaces one by one, we prune branches of the search tree by checking inequality (1) (reproduced below for convenience).

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

In contrast, when searching multiple subspaces simultaneously, we use inequality (E.2), which holds less frequently than (1), because the left-hand side in (E.2) is greater than that in (1). This increase (in the left-hand side) can be seen as the price that must be paid in order to avoid searching every $\Pi_{I'}^A$ with $I' \in \mathcal{Y}^*$ separately later on.

The problem, however, is that this price is often too large. To see why, let us analyse the two modifications that are behind this increase.

- The first modification is when $|I \setminus I^*| < d$. In this case, every $C_j$ with $j \in \{|I \setminus I^*|, \ldots, d\}$ is evaluated as $f_{M^*}(C_j)$ rather than as $v(C_j)$.
- 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 $\Pi_{I'}^A$ with $I' \in \mathcal{Y}^*$ is searched simultaneously with $\Pi_I^A$, the upper bound in (E.2) becomes $UB_I^d$ instead of $\sum_{j=d+1}^{|I|} Max_{i_j}$.

A key point here is that $\mathcal{Y}^*$ does not necessarily 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* $\Pi_I^A$ *as well as* $\Pi_{I'}^A$ *with* $I' \in \mathcal{Y}^*$. However, in the first modification, every $C_j$ with $j \in \{|I \setminus I^*|, \ldots, d\}$ is evaluated as $f_{M^*}(C_j)$—the value of the best partition of $C_j$ in all the subspaces that are reachable from $\Pi_I^A$, *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 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 of size $|I|$ containing $C_1, \ldots, C_d$—it suffices to put all the remaining agents in a coalition of their own. Thus, 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, inequality (E.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 $\Pi_I^A$.

## Appendix F. Pseudocode of the Inclusion-Exclusion Algorithm

While Björklund et al. [8] provide a detailed description of their inclusion-exclusion algorithm, they do not include pseudocode for it. We therefore provide pseudocode for their algorithm in this appendix. Note that the algorithm is

defined for situations where (1) the goal is to find the best partition containing at most $k$ subsets, (2) there are $k$ evaluation functions, $f_1, \ldots, f_k$, and (3) the value of the $i$-th subset is given by the $i$-th evaluation function, $f_i$. In our setting, we have $k = n$ (since coalition structures are allowed to contain up to $n$ coalitions) and $f_i = v$ for all $i \in \{1, \ldots, k\}$ (since all coalitions are evaluated using the same function, $v$). Thus, the pseudocode below is for the case where $k = n$ and $f_i = v$ for all $i \in \{1, \ldots, k\}$.

Pseudocode for the inclusion-exclusion algorithm.

**Algorithm** `Inclusion-Exclusion`$(v, A)$

$B \leftarrow n^n + 1$
**foreach** $S \subseteq A$ **do** `// for every coalition`
  **foreach** $c \in \{1, \ldots, n\}$ **do** `// for every color`
    $f'_c(S) \leftarrow B^{v(S)}$
$optimalValue \leftarrow$ `findMaxWeight`$(f'_1, \ldots, f'_n)$`// see the pseudo code of findMaxWeight`
**foreach** $i \in \{1, \ldots, n\}$ **do** `// for every agent` $a_i \in A$
  **foreach** $c = 1$ ***to*** $n$ **do** `// try to assign color` $c$ `to` $a_i$`, starting from color 1, then 2, etc. (order matters)`
    **foreach** $S \subseteq A$ **do** `// define function` $\tilde{f}'_c(S)$ `for every coalition` $S$
      $\tilde{f}'_c(S) \leftarrow f'_c(S)$ if $a_i \in S$, and $\tilde{f}'_c(S) \leftarrow B^0$ otherwise.
    **foreach** $j \in \{1, \ldots, n\}$ **do** `// define function` $F'_j$ `for every color` $j$
      $F'_j \leftarrow \tilde{f}'_j$ if $j = c$, and $F'_j \leftarrow f'_j$ otherwise.
    $valueAfterColoring \leftarrow$ `findMaxWeight`$(F'_1, \ldots, F'_n)$`// see the pseudo code of findMaxWeight`
    **if** $valueAfterColoring = optimalValue$ **then**
      $color(i) \leftarrow c$`// set` $c$ `to be the color of` $a_i$
      **foreach** $S \subseteq A : a_i \in S$ **do**
        **foreach** $j \in \{1, \ldots, n\} \setminus \{c\}$ **do** `// for every color` $j \neq c$
          $F'_j(S) \leftarrow B^0$
      $f' \leftarrow F'$ `// Now,` $f'$ `is updated to reflect that` $c$ `is now the color of` $a_i$
      **break**
**foreach** $c \in \{1, \ldots, n\}$ **do** `// for every color` $c$
  $S_c \leftarrow \{a_i \in A : color(i) = c\}$
**return** $\{S_1, ..., S_n\}$ `// Some of these subsets may be empty.`

Pseudocode of all procedures used by the inclusion-exclusion algorithm.

**Procedure** `findMaxWeight` $(f_1, \ldots, f_n)$

$t \leftarrow p_n(f_1, \ldots, f_n)$; $B \leftarrow n^n + 1$; $r \leftarrow 0$
**while** $B^r < t$ **do**
  $r \leftarrow r + 1$
**return** $r - 1$ `// this is simply` $\lfloor log_B(t) \rfloor$

**Procedure** $p_n$ $(f_1, \ldots, f_n)$ `// sum of weighted partitions of` $A$ `into` $n$ `parts (some of which may be empty)`

$t \leftarrow 0$
**foreach** $X \subseteq A$ **do**
  $t \leftarrow t + (-1)^{|X|} b_n(X, f_1, \ldots, f_n)$
**return** $t$

**Procedure** $b_n$ $(X, f_1, \ldots, f_n)$ `// auxiliary function for computing` $p_n$

**return** $g(1, n, n, X, f_1, \ldots, f_n)$ `// initial call of the recursive function` $g$

**Procedure** $g$ $(s, t, m, X, f_1, \ldots, f_n)$ `// auxiliary function for computing` $b_n(X)$

**if** $s = t$ **then**
  **return** $zeta(A \setminus X, m, f_1, dots, f_n)$
**else**
  $r \leftarrow \lfloor (s + t)/2 \rfloor$; *temp* $\leftarrow 0$
  **foreach** $m_0 = 0$ ***to*** $m$ **do**
    $m_1 \leftarrow m - m_0$
    $temp \leftarrow temp + g(s, r, m_0, X, f_1, \ldots, f_n) * g(r + 1, t, m_1, X, f_1, \ldots, f_n)$
**return** *temp*

**Procedure** $zeta$ $(Y, \ell, f_1, \ldots, f_n)$ `// zeta-transform of` $f$ `that only sums over subsets of size` $\ell$

**return** $z(n, Y, \ell, f_1, \ldots, f_n)$

**Procedure** $z$ $(i, Y, \ell, f_1, \ldots, f_n)$ `// auxiliary function for computing` $zeta(f, Y, \ell)$ `using the fast Mobius transform`

**if** $i = 0$ **then**
  **if** $|Y| = \ell$ **then**
    **return** $f(Y)$
  **else**
    **return** *0*
**else**
  **if** $i \in Y$ **then**
    $t \leftarrow z(f, i - 1, Y, \ell) + z(f, i - 1, Y \setminus \{i\}, \ell)$
  **else**
    $t \leftarrow z(f, i - 1, Y, \ell)$
**return** $t$