A Hybrid Algorithm for Coalition Structure Generation

Talal Rahwan[1], **Tomasz Michalak**[2], **Nicholas R. Jennings**[1]
[1] School of Electronics and Computer Science, University of Southampton, UK
[2] Institute of Informatics, University of Warsaw, Poland

Abstract

The current state-of-the-art algorithm for optimal coalition structure generation is IDP-IP—an algorithm that combines IDP (a dynamic programming algorithm due to Rahwan and Jennings, 2008b) with IP (a tree-search algorithm due to Rahwan et al., 2009). In this paper we analyse IDP-IP, highlight its limitations, and then develop a new approach for combining IDP with IP that overcomes these limitations.

1 Introduction

Coalition formation, the process by which groups of agents cooperate to improve their performance, is an important form of interaction in multi-agent systems. Sample applications of coalition formation include distributed vehicle routing (Sandholm and Lesser 1997), sensor networks (Dang et al. 2006), and e-commerce (Tsvetovat et al. 2000). Coalitional games provide a formal model of coalition formation scenarios. A class of these games that received a lot of attention in the literature is *characteristic function games*, where, given a set of agents A, a *characteristic function* $v : 2^A \to \mathbb{R}$ assigns to every subset (or *coalition*) of agents a real value that reflects its performance. Since in many realistic setting multiple coalitions can co-exist, we often talk about a *coalition structure* which is a partition of the set of agents into disjoint and exhaustive coalitions. An important research question is how to find an *optimal coalition structure*, i.e. one in which the total value of all the coalitions is maximal. This problem, which is widely known as the *Coalition Structure Generation (CSG)* problem, received considerable attention in recent years, and has been shown to be computationally hard (Sandholm et al. 1999). A number of algorithms have been developed to try and combat this complexity, and the focus of this paper is on *exact* algorithms (i.e., algorithms that are guaranteed to eventually find an optimal solution). The two main classes of these algorithms are:

1. **Dynamic programming algorithms:** The basic idea of dynamic programming is to break the optimization problem into sub-problems that can be solved recursively, and then combine the results of those subproblems to produce a solution to the original problem. The state-of-the-art CSG algorithm in this class is IDP (Rahwan and Jennings 2008b) and its other version proposed by Service and Adams (2011).
2. **Tree-search algorithms:** In this class, the state-of-the-art algorithm is IP (Rahwan et al. 2009), which divides the search space into subspaces that are searched using depth-first searched combined with branch-and-bound techniques.

The main difference between IDP and IP is that the performance of IDP is independent of the values of the characteristic function; it depends solely on the number of agents. On the other hand, the performance of IP, given a fixed number of agents, may change dramatically depending on the characteristic function values. Thus, in the worst case, IP can be significantly slower than IDP. To be more precise, IP runs in $O(n^n)$, while IDP runs in $O(3^n)$, where n is the number of agents. In practice, however, IP has been shown to significantly outperform IDP for many popular test distributions of coalition values (see Section 4 for more details).

Another important difference between IDP and IP is that the latter is an *anytime* algorithm, i.e., its solution quality improves monotonically as computation time increases. This implies that it can return a solution even if it is stopped prematurely. IDP, on the other hand, only returns a solution once it runs to completion. Recently, an anytime version of IDP was proposed by Service and Adams (2011).[1]

Since each of IP and IDP has its own advantages over the other, there is scope to combine the two in the hope of obtaining the best features of both. To this end, Rahwan and Jennings (2008a) introduced a novel representation of the space of possible coalition structures (see Section 2.2 for more details). This representation provided a deeper understanding of how both algorithms (IDP and IP) worked, and provided insight into how they could be merged together. Based on this, the authors developed an algorithm, called IDP-IP, which initially starts by running IDP, and then switches to IP to continue the search. The moment in time at which the switch occurs can be controlled by a single integer parameter, called m. On one extreme, setting m to 1 means that the algorithm will switch to IP right from the start, and so IDP will never be used. On the other extreme, by setting m to $\left\lfloor \frac{2\times n}{3} \right\rfloor$, IDP will run to completion, and the switch to

[1]We postpone the discussion on this version of IDP until Section 4 to enhance the readability of the paper.

IP will never occur. By setting m anywhere between these two extremes, the user can control the number of operations that will be carried out by IDP before making the switch to IP.

While the idea of combining IDP and IP has a lot potential, we argue that the way Rahwan and Jennings combined them suffers from two major limitations:

- As we show in this paper, the effect that different values of m have on the performance of IDP-IP can be very different from one coalition-value distribution to another. In other words, without prior knowledge of the distribution, it is unknown how m should be determined in order to optimize the performance. We show that, by choosing the wrong m, the performance can deteriorate dramatically.
- The way IDP and IP were combined in IDP-IP makes it impossible to return a solution before IDP finishes the preprocessing. This implies that, for a relatively large value of m, the anytime property of IP deteriorates significantly.

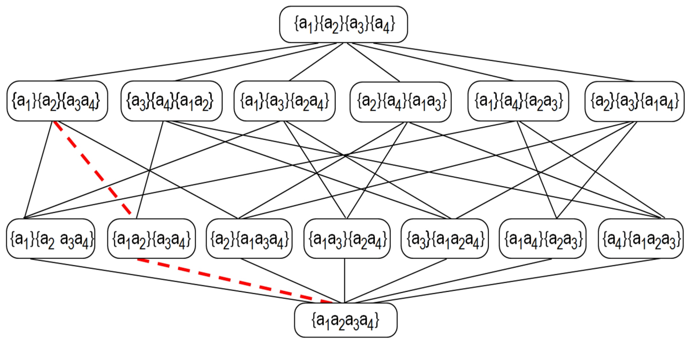
Against this background, the contributions of this paper can be summarized as follows:

- We develop IDP-IP*—an algorithm that combines IDP and IP without any control parameters. Instead, it automatically adjusts itself between its two parts (IDP and IP) so as to reflect the relative strength of each part with regards to the problem instance at hand (i.e., the more effective a part is, compared to the other, the more search operations will be assigned to that part). This adjustment is done without any prior knowledge of the coalition value distribution.
- IDP-IP* does not use IDP for preprocessing and, thus, avoids the need to wait until IDP terminates. This way, the combination of IDP and IP retains the desirable anytime property of IP.
- We show that, for the majority of distributions considered in this paper, IDP-IP* outperforms IDP-IP *even if the latter was adjusted to its optimal setting* (i.e., even if the optimal value of m was known). As for the remaining distributions, we show that IDP-IP* is relatively close to the optimal setting of IDP-IP.

The remainder of the paper is organized as follows. Section 2 describes the IDP and IP algorithms, as well as IDP-IP. Our algorithm is presented in Section 3 and evaluated in Section 4. Related work is discussed in Section 5. Finally, Section 6 concludes the paper and outlines future work.

2 Explaining IDP, IP and IDP-IP

In this section we describe the IDP, IP and IDP-IP algorithms. However, before we do that, we need to present a particular representation of the space of the possible coalition structures, since this representation will provide insight into the way these algorithms work. In particular, this representation is known as the *coalition structure graph* (Sandholm et al. 1999), which is an undirected graph where every node represents a coalition structure. The nodes (i.e., coalition structures) are categorized into levels according the number of coalitions in each node. An edge connects two



$\{a_1\}\{a_2\}\{a_3\}\{a_4\}$
$\{a_1\}\{a_2\}\{a_3a_4\}$
$\{a_3\}\{a_4\}\{a_1a_2\}$
$\{a_1\}\{a_3\}\{a_2a_4\}$
$\{a_2\}\{a_4\}\{a_1a_3\}$
$\{a_1\}\{a_4\}\{a_2a_3\}$
$\{a_2\}\{a_3\}\{a_1a_4\}$
$\{a_1\}\{a_2\ a_3a_4\}$
$\{a_1a_2\}\{a_3a_4\}$
$\{a_2\}\{a_1a_3a_4\}$
$\{a_1a_3\}\{a_2a_4\}$
$\{a_3\}\{a_1a_2a_4\}$
$\{a_1a_4\}\{a_2a_3\}$
$\{a_4\}\{a_1a_2a_3\}$
$\{a_1a_2a_3a_4\}$

Figure 1: The coalition structure graph of 4 agents.

coalition structures iff: (1) they belong to two consecutive levels, and (2) one of the coalition structures can be obtained from the other by splitting one coalition into two. An example is shown in Figure 1.[2]

When explaining the algorithms, we will use the following notations. Let A be the set of agents, and n be the number of agents. Also, let Π^A be the set of possible coalition structures (i.e., partitions over A) and, for any coalition structure CS, let $V(CS)$ denote the *value* of CS, where: $V(CS) = \sum_{C \in CS} v(C)$. Furthermore, let CS^* denote an *optimal* coalition structure. That is, $CS^* \in \arg\max_{CS \in \Pi^A} V(CS)$. Finally, let CS^{**} denote the best coalition structure found by an algorithm at any point in time.

2.1 IDP

IDP is an improved version of a previous algorithm, called DP (Yeh 1986). Hence, to better understand IDP, we need to first give a brief description of DP. To this end, for any coalition $C \subseteq A$, let Π^C be the set of possible *partitions of* C, where a partition $P = \{P_1, \cdots, P_{|P|}\} \in \Pi^C$ is a set of disjoint coalitions of which the union equals C. Moreover, let $V(P)$ be the *value of partition* P, where $V(P) = \sum_{P_i \in P} v(P_i)$. Finally, let $f(C)$ be the value of the optimal partition of C, i.e., $f(C) = \max_{P \in \Pi^C} V(P)$. Then, DP is based on the following recursive formula to compute $f(C)$:

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

In more detail, DP iterates over all the coalitions of size 1, and then over all those of size 2, and then size 3, and so on until size n. For every such coalition C, it computes $f(C)$ using equation (1). As can be seen, whenever $|C| > 1$, the equation requires comparing $v(C)$ with $\max_{\{C',C''\} \in \Pi^C} f(C') + f(C'')$. The result of this comparison is stored in a table, t, which has an entry for every coalition. In particular, if $v(C)$ was greater, then the algorithm sets $t[C] = C$ so that it can later on remember that it is not beneficial to split C into two coalitions. Otherwise, it sets $t[C] = \arg\max_{\{C',C''\} \in \Pi^C} f(C') + f(C'')$ to remember

[2]The reason for highlighting some of the edges will be made clear in the following sub-section.

the best way of splitting C into two coalitions. By the end of this process, $f(A)$ will be computed, which is by definition equal to $V(CS^*)$. What remains is to compute CS^* itself. This is done recursively using the t table, e.g.:

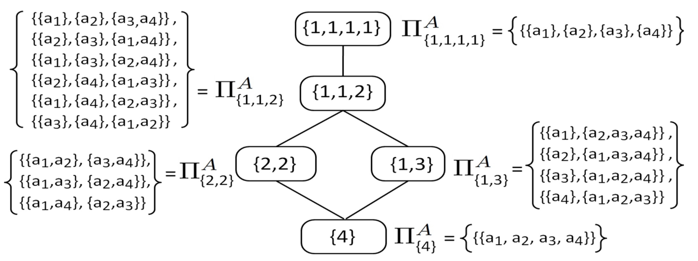
Example 1. *Given $A = \{a_1, a_2, a_3, a_4\}$, suppose that $t[A] = \{\{a_1, a_2\}\{a_3, a_4\}\}$, i.e., it is most beneficial to split A into $\{a_1, a_2\}$ and $\{a_3, a_4\}$. Moreover, suppose that $t[\{a_1, a_2\}] = \{\{a_1\}, \{a_2\}\}$, while $t[\{a_3, a_4\}] = \{a_3, a_4\}$, i.e., it is most beneficial to split $\{a_1, a_2\}$, but it is not beneficial to split $\{a_3, a_4\}$. In this case, $CS^* = \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$.*

Rahwan and Jennings (2008b) showed how the operation of DP can be visualized on the coalition structure graph. To this end, observe that every movement upwards in the graph corresponds to the splitting of one coalition into two. The way DP works is by determining, for every coalition $C \subseteq A$, whether it is beneficial to split C, and if so what is the best such split (the answer to this question is stored in $t[C]$). This way, every time the algorithm reaches a node (i.e., a coalition structure) that contains C, it can determine (based on $t[C]$) whether it is beneficial to make a movement that involves splitting C. Once the possible movements have been evaluated (by computing $t[C]$ for all $C \subseteq A$), the algorithm moves upwards in the graph through a series of connected nodes (called a *"path"*) until an optimal node is reached, after which no movement is beneficial. For instance, the way DP reached $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ in Example 1 can be visualized as movements through the dashed path in Figure 1, 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 $\{a_2\}$.

Now, given three fixed integers values i_1, i_2, i_3, where $1 \leq i_1 \leq i_2 < i_3 < n$, and $i_1 + i_2 = i_3$, Rahwan and Jennings asked the following question: what if, for every coalition $C : |C| = i_3$, DP did not evaluate the possible ways of splitting C into two coalitions C_1, C_2 where $|C_1| = i_1$ and $|C_2| = i_2$? They showed that this corresponds to the removal of every edge that represents the splitting of one coalition of size i_3 into two coalitions of sizes i_1 and i_2. More importantly, they showed that DP will still be able to find the best of all the nodes that are still reachable from the bottom one. Based on this, they identified a subset of edges, called E^*, that is sufficient to have a path from any node in the graph to the bottom one, and they developed IDP, which is very similar to DP except it only evaluated the edges in E^* instead of evaluating all edges.

2.2 IP

The IP algorithm is based on the *integer partition-based representation* (Rahwan et al. 2007a) of the space of possible coalition structures. This representation divides the space into subspaces that are each represented by an *integer partition* of n.[3] For example, given $n = 4$, the possible integer partitions are $\{4\}$, $\{1, 3\}$, $\{2, 2\}$, $\{1, 1, 2\}$, $\{1, 1, 1, 1\}$, and each one of these represents a subspace consisting of



{1,1,1,1}
$\Pi^A_{\{1,1,1,1\}} = \{\{\{a_1\},\{a_2\},\{a_3\},\{a_4\}\}\}$
{1,1,2}
$\{\{\{a_1\},\{a_2\},\{a_3,a_4\}\}, \{\{a_2\},\{a_3\},\{a_1,a_4\}\}, \{\{a_1\},\{a_3\},\{a_2,a_4\}\}, \{\{a_2\},\{a_4\},\{a_1,a_3\}\}, \{\{a_1\},\{a_4\},\{a_2,a_3\}\}, \{\{a_3\},\{a_4\},\{a_1,a_2\}\}\} = \Pi^A_{\{1,1,2\}}$
{2,2}
$\{\{\{a_1,a_2\},\{a_3,a_4\}\}, \{\{a_1,a_3\},\{a_2,a_4\}\}, \{\{a_1,a_4\},\{a_2,a_3\}\}\} = \Pi^A_{\{2,2\}}$
{1,3}
$\Pi^A_{\{1,3\}} = \{\{\{a_1\},\{a_2,a_3,a_4\}\}, \{\{a_2\},\{a_1,a_3,a_4\}\}, \{\{a_3\},\{a_1,a_2,a_4\}\}, \{\{a_4\},\{a_1,a_2,a_3\}\}\}$
{4}
$\Pi^A_{\{4\}} = \{\{\{a_1, a_2, a_3, a_4\}\}\}$

Figure 2: The integer partition graph of 4 agents.

all the coalition structures within which the coalition sizes match the parts of the integer partition. For instance, if we denote by $\mathcal{I}^n$ the set of integer partitions of n, and by Π^A_I the subspace that corresponds to $I \in \mathcal{I}^n$, then $\Pi^A_{\{1,1,2\}}$ is the subspace containing all the coalition structures within which two coalitions are of size 1, and one coalition is of size 2. A different version of this representation was later on proposed where every subspace is represented by a node in an undirected graph called the *integer partition graph* (Rahwan and Jennings 2008a). In this graph, two nodes representing $I, I' \in \mathcal{I}^n$, where $|I| > |I'|$, are connected via an edge iff there exists two parts $i, j \in I$ such that $I' = (I \setminus \{i, j\}) \uplus \{i + j\}$. Given 4 agents, for example, Figure 2 shows the integer partition graph and the subspace that corresponds to every node in the graph.

With this representation, it is possible to compute upper and lower bounds on the value of the best coalition structure that can be found in each subspace. To this end, for every coalition size $s \in \{1, 2, \ldots, n\}$, let L_s denote the list of all the possible coalitions of size s. Moreover, let Max_s and Avg_s be the maximum and average values of the coalitions in L_s, respectively. Then, for all $I \in \mathcal{I}^n$, it is possible to compute an upper bound UB_I on the value of the best coalition structure in Π^A_I as follows: $UB_I = \sum_{s \in I} I(s) \cdot Max_s$, where $I(s)$ is the multiplicity of s in I. Similarly, a lower bound LB_I on the value of the best coalition structure in Π^A_I can be computed as follows: $LB_I = \sum_{s \in I} I(s) \cdot Avg_s$. These bounds are then used to establish worst-case guarantees on the quality of CS^{**}, and to prune the subspaces that have no potential of containing a solution better than CS^{**}. As for the remaining subspaces, IP searches them one at a time, unless a coalition structure is found that has a value greater than, or equal to, the upper bound of some subspace, in which case that subspace no longer needs to be searched. The process of searching a subspace, say $\Pi^A_I : I = \{i_1, \ldots, i_{|I|}\}$, is done in a depth-first manner: the algorithm iterates over the coalitions in L_{i_1} and, for every $C_1 \in L_{i_1}$ that the algorithm encounters, it iterates over the coalitions in $L^A_{i_2}$ that do not overlap with C_1. Similarly, for every $C_2 \in L^A_{i_2}$ that the algorithm encounters, it iterates over the coalitions in $L^A_{i_3}$ that do not overlap with $C_1 \cup C_2$, and so on. This process is repeated until the last list, $L^A_{i_{|I|}}$, is reached. For every $C_{|I|} \in L^A_{i_{|I|}}$ that the algorithm encounters, it would have selected a combination of $|I|$ coalitions, namely $\{C_1, C_2, \ldots, C_{|I|}\}$, that is guaranteed to be a coalition structure in Π^A_I. Eventually, all the coalition structures

[3]An *integer partition* of n is a multiset of positive integers, or *parts*, of which the sum equals n (Andrews and Eriksson 2004).

in Π_I^A are examined.

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

$$\sum_{j=1}^{d} v(C_j) + \sum_{j=k+1}^{|I|} Max_{i_j} < V(CS^{**}) \qquad (2)$$

Now if the inequality in (2) holds, then this means none of the coalition structures that contain $C_1, \ldots, C_d$ can improve upon the quality of the best solution found so far, and so these coalition structures can be skipped during the search.

2.3 IDP-IP

As discussed earlier in Section 2.1, the operation of IDP can be visualized as movements through edges in the *coalition structure graph*, and if certain splittings were not evaluated, then this can be visualized by removing the corresponding edges from that graph. Importantly, however, the way IDP works can also be visualized on the integer partition graph. Basically, by making a movement from one coalition structure CS' to another CS'', IDP is actually making a movement (in the integer partition graph) from one integer partition $I' : \Pi_{I'}^A \ni CS'$ to another $I'' : \Pi_{I''}^A \ni CS''$. Moreover, by removing all the edges that correspond to the splitting of a coalition of size s into two coalitions of sizes s' and s'', we are actually removing every edge that connects an integer partition $I : I \ni s$ to another $I' = (I \setminus \{s\}) \uplus \{s', s''\}$. This visualization provides the link between IDP and IP since the latter deals with subspaces that are represented by integer partitions.

Having presented the link between IDP and IP, we now show how they are combined in IDP-IP. Basically, instead of setting IDP to evaluate the possible splittings of the coalitions of size $s \in \{1, 2, \ldots, n\}$, we can set it to $s \in \{1, 2, \ldots, m, n\}$, where $m < n$. As for the coalitions of the remaining sizes, we simply set $f[C] = v(C)$ and $t[C] = C$. This corresponds to the removal of the edges (in the integer partition graph) that involve replacing an integer $i : m < i < n$ with two smaller ones. Recall that IDP knows how to make the best movements through the remaining edges. Unfortunately, however, those movements are always made starting from the bottom node, i.e., $\Pi_{\{n\}}^A$. Thus, while IDP can still find the best solution in any of the subspaces that are reachable from $\Pi_{\{n\}}^A$, the part of the graph that is now disconnected from $\Pi_{\{n\}}^A$ will no longer be searched. To search this part, Rahwan and Jennings use a modified version of IP. This is based on the observation that, for any CS, one can compute the value of the best coalition structure reachable from CS as follows: $\sum_{C \in CS} f(C)$. Now since IP can search any subspace and find the coalition structure that maximizes $\sum_{C \in CS} v(C)$, then by simply replacing v with f we can use IP to determine, for every subspace, the coalition structure from which the best movements can be made. IDP can then be used to make those movements using the t table. Similarly, every subspace that has no edge leading to it must be searched by IP, followed by IDP.

When m is set to 1, every subspace will have no edge leading to it, and so must be searched by IP. Thus, IDP-IP become identical to IP. On the other hand, when m is set to $\left\lfloor \frac{2 \times n}{3} \right\rfloor$, every subspace would have an edge leading to it. Thus, IP will not be used, and IDP-IP becomes identical to IDP. More importantly, by setting m anywhere between those two extremes, one can determine how much of IDP, and how much of IP, to have in the performance of IDP-IP.

3 Introducing IDP-IP*

The way IDP and IP were combined in IDP-IP was in a sequential manner; IDP operates first while IP stands idle, and then IP starts operating, at which point IDP becomes idle. As we will show in Section 4.1, such a combination can end up performing significantly worse than its constituent parts. Furthermore, the only way to determine when to switch from IDP to IP is by experimenting with all possible settings of the m parameter. The optimal moment in time to make this switch can be very different from one value distribution to another.

To overcome this limitation, we developed IDP-IP*—an algorithm that combines IDP and IP in such a way that allows them to work in a more interactive manner. More specifically, it runs IDP and IP simultaneously, and enables each one of them to enhance the performance of the other without ever delaying it. This way, IDP-IP* always adjusts itself such that the proportion of search assigned to each of its two constituent parts (IDP and IP) reflects the relative strength of that part with respect to the problem instance at hand. This adjustments happens *automatically*, without the need for any *a priori* knowledge of the coalition value distribution.

The remainder of this section describes IDP-IP* in more detail. We first show how IDP enhances the performance of IP, and then show how IP enhances IDP's performance.

3.1 Improving IP with IDP

As mentioned earlier in Section 2.2, every time IP reaches a certain depth d in the search tree of a subspace Π_I^A, it adds a coalition C_d to a set of disjoint coalitions $\{C_1, \cdots, C_{d-1}\}$. The way IP prunes the branch $\{C_1, \ldots, C_d\}$ is by determining whether $C_{d+1}, \ldots, C_{|I|}$ are promising (see equation (2)). However, we show how to modify IP such that, even if this is not the case, IP may still be able to prune the branch. The basic idea is to keep, as much as possible, track of the value of the best partition that has been encountered for any subset of agents (let us denote this value as $v'(C)$ for coalition C). This way, before IP adds the coalition C_d, it checks whether:

- $v'(C_d) > v(C_d)$: If this holds, then the branch $\{C_1, \cdots, C_d\}$ can be pruned. This is because we are interested in finding the optimal coalition structure CS^*, and we know that CS^* cannot contain a coalition of which a better partition exists.
- $v'(\cup_{j=1}^{d} C_j) > \sum_{j=1}^{d} v(C_j)$: If this holds, then IP can prune the branch $\{C_1, \cdots, C_d\}$ (following the same rationale).

Note that, for any subset of agents $C \subseteq A$, if $v'(C)$ is found to be smaller than $v(C)$, then $v'(C)$ is updated by simply setting $v'(C) = v(C)$.

In order to use the above pruning technique, IP needs a table in which to store v' for every subset of agents. However, with the help of IDP, this can be done without the need for any extra memory requirements, since $v'(C)$ can be stored in $f(C)$. To be more precise, IDP initially sets $f(C) = v(C)$ for every $C \subseteq A$. After that, IP stores in f the values of the best partitions that it had encountered so far, and uses those values to prune branches of the search tree whenever possible. Meanwhile, IDP enhances the values in f (since IDP is very effective at computing the values of the best partitions of coalitions, especially the small ones). This way, for any $C \subseteq A$, if IDP hasn't yet computed $f(C)$, then $v'(C)$ will be the value of the best partition encountered, and stored, by IP. As soon as IDP computes $f(C)$, $v'(C)$ becomes the value of the best possible partition of C.

To better understand the effect that IDP has on our proposed branch-and-bound technique, let us consider an example of 19 agents. In this example, since we have 19 agents, our results show that IDP can compute the value of the best partition of all coalitions of size $s \in \{1, \dots, 9\}$ in less than one second (see Section 4). Now, suppose that, after one second from the start time, IP reached the following branch: $\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_7, a_8\}, \{a_9\}$. By this time, IDP has already computed the optimal partition of each of the following subsets of agents: $\{a_1, a_2\}$, $\{a_3, a_4\}$, $\{a_5, a_6\}$, $\{a_7, a_8\}$, $\{a_1, \dots, a_4\}$, $\{a_1, \dots, a_6\}$, $\{a_1, \dots, a_8\}$, $\{a_1, \dots, a_9\}$. Now since we modify IP such that it checks the aforementioned conditions at every depth, then the branch will be pruned unless every one of those subsets has a value that is equal to its optimal partition. This is less likely to happen when the number of possible partitions is large. For instance, there is a total of 21147 possible partitions of $\{a_1, \dots, a_9\}$. Therefore, if at least one of those partitions has a value greater than that of $\{a_1, \dots, a_9\}$, then the entire branch will be pruned. Similarly, there is a total of 4140 possible partitions of $\{a_1, \dots, a_8\}$, and so the branch will be pruned if one of those partitions has a value greater than that of $\{a_1, \dots, a_8\}$.

Having presented a new branch-and-bound technique, we will now present another technique that we use to enhance IP with the help of IDP. To this end, without loss of generality, let us assume that the integers in I are ordered. Furthermore, let us denote by m^* the maximum size of coalitions for which IDP has finished computing the f values. Now, for any integer partition $I = \{i_1, \cdots, i_{|I|}\}$, we will show how IP can avoid going in the search tree of Π_I^A beyond depth d^*, where:

$$d^* = \max_{k \in \{1, \dots, |I|\} : \sum_{j=k}^{|I|} i_j > m^*} k$$

Basically, IP searches Π_I^A as usual, but as soon as it reaches the depth d^*, it does not try to construct all the possible coalition structures in Π_I^A that start with $C_1, \dots, C_{d^*}$. Instead, it simply selects the coalition structure $\{C_1, \dots, C_{d^*}, \cup_{j=d^*+1}^{|I|} C_j\}$, and evaluates it as follows: $v(C_1) + \cdots + v(C_{d^*}) + f(\cup_{j=d^*+1}^{|I|} C_j)$.

Unlike IDP-IP, which groups as many integer as possible in I into bigger integers (that are smaller than, or equal to, m^*), we only group the integers at the end of I. This ensures that the effectiveness of applying branch-and-bound based on equation (2) is not influenced negatively (see Section 4.1 for a detailed discussion on how such a negative influence is created in IDP-IP). Avoiding the search beyond a certain depth of the tree is very promising, especially since the size of the tree drops exponentially as the tree becomes shorter.

3.2 Improving IDP with IP

In this subsection, we show how IP can be modified so as to help IDP. This is done by simply changing the order through which IP searches the subspaces. Recall that, of all remaining subspaces, IP typically starts with the one that has the highest upper bound. As for IDP, recall that after it computes the f values of all coalitions of size m^*, it can find the optimal solution among all subspaces that are still reachable from the bottom node, where the edges represent a split of an integer $i \leq m^*$ into two smaller integers. Furthermore, as m^* increases, more subspaces become reachable from the bottom node, i.e., IDP covers more subspaces. Against this background, we modify IP such that it starts with the subspaces that IDP would cover last, i.e., the subspaces that can only be covered by IDP once $m^* = \lfloor \frac{2 \times n}{3} \rfloor$. Once IP finishes searching those subspaces, IDP has no need to continue with its search until $m^* = \lfloor \frac{2 \times n}{3} \rfloor$. Instead, it can terminate once $m^* = \lfloor \frac{2 \times n}{3} \rfloor - 1$. The same process is repeated, i.e., IP searches the subspaces that would only be covered by IDP when $m^* = \lfloor \frac{2 \times n}{3} \rfloor - 1$, and then those covered when $m^* = \lfloor \frac{2 \times n}{3} \rfloor - 2$ and so on. This way, even if IDP happens to be more effective than IP for a certain distribution, IP can still take some of the burden off IDP. This also gives IDP-IP* the ability to calibrate itself automatically such that the amount of search assigned to each of its two constituent parts (IDP and IP) reflects the relative strength of that part with respect to the value distribution at hand.

4 Performance evaluation

In this section we evaluate IDP-IP (Section 4.1) and IDP-IP* (Section 4.2) with the following coalition value distributions:

- **Uniform**, as studied in (Larson and Sandholm 2000): $v(C) \sim U(a, b)$ where $a = 0$ and $b = |C|$;
- **Normal**, as studied in (Rahwan et al. 2007b): $v(C) \sim N(\mu, \sigma^2)$ where $\mu = 10 \times |C|$ and $\sigma = 0.1$;
- **Modified Uniform**, as studied in (Service and Adams 2010): $v(C) \sim U(0, 10 \times |C|)$, and every $v(C)$ is increased by a random number $r \sim U(0, 50)$ with probability 20%;
- **Modified Normal**, introduced in this paper: $v(C) \sim N(10 \times |C|, (0.1)^2)$, and every $v(C)$ is increased by a random number $r \sim U(0, 50)$ with probability 20%;
- **NDCS** (Normally Distributed Coalition Structures), as studied in (Rahwan et al. 2009): $v(C) \sim N(\mu, \sigma^2)$, where $\mu = |C|$ and $\sigma = \sqrt{|C|}$;

- **Agent-based**, introduced in this paper: Each agent a_i is given a random power $p_i \sim U(0, 10)$ reflecting its average performance over all coalitions. The agent's power in any particular coalition C is denoted as $p_i^C \sim U(0, 2p_i)$. Then, the value of a coalition is the sum of the powers of its member. That is, $v(C) = \sum_{a_i \in C} p_i^C$;

Time is plotted on a *log scale*, and measured as in milliseconds on a PC with an Intel i7 (3.40GHz) and 12GB of RAM.

4.1 Evaluating IDP-IP

Observe that, when running IP alone, the only information available to IP are the values of the coalitions. However, by running IDP for every coalition of size $1, \ldots, m$, IP obtains extra information; it knows the value of the best partition of each of these coalitions. The only cost to obtain this extra information is that IP has to wait for IDP to terminate. It is important to underline the fact that this cost is *insignificant* for relatively small values of m; e.g., given $n = 20$ it takes only 0.05 seconds to obtain the best partition of every coalition up to size $m = 6$. Surprisingly, however, with this extra information IP becomes significantly slower for certain distributions. To demonstrate this point, we tested IDP-IP with different distributions and different values of m, and that is for $n = 20$ (see Figure 3). As can be seen, given NDCS, Modified Unifrom, and Modified Normal, IDP-IP is significantly slower with $m = 6$ compared to the case where IP has no extra information, i.e., the case where $m = 1$. This raises the following question: *how can IP underperform given extra information?* The answer is that, in IDP-IP, the extra information is used to help IP avoid searching as many subspaces as possible. However, as a result of this process, the effort that IP has to put when searching the remaining subspaces increases (compared to the effort that it would have put to search those subspaces without any extra information). The reason behind this increase can be found in equation (2)—the equation based on which IP applies branch-and-bound (see Section 2.2 for more details). Specifically, the branch that starts with $C_1, \ldots, C_d$ is deemed unpromising by IP if the inequality in (2) holds. In IDP-IP, however, a different version of this equation is used, where f is used instead of v. More specifically, $v(C_j)$ is replaced with $f(C_j)$, and Max_{i_j} is calculated as $max_{C \in L_{i_j}} f(C)$ instead of $max_{C \in L_{i_j}} v(C)$. Now since $f(C) \geq v(C)$ for all $C \subseteq A$, using f makes the left-hand side of (2) larger, which decreases the probability of satisfying the inequality. In fact, this decrease can be so significant that the overall performance becomes slower by orders of magnitude (as is the case, for instance, with NCSG and Modified Uniform).

4.2 Evaluating IDP-IP*

For every distribution, and for every number of agents between 15 and 25, we benchmark IDP-IP* against IDP-IP with *the best choice of* m, *the worst choice*, and *the average choice* (see Figure 4).[4] Here, by the average choice we mean the performance of IDP-IP averaged over all possible values of m. Such a comparison is particularly relevant

[4]Results that require more than 24h to run were extrapolated. Error bars were omitted to enhance the readability of the figure.

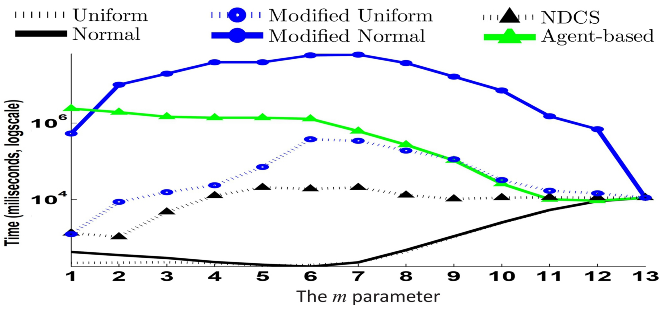


Figure 3: IDP-IP given $n = 20$ and different values of m (*log scale*).

when the optimal value of m is unknown in advance. As can be seen, for NDCS, Modified Uniform, Modified Normal, and Agent-based Uniform, IDP-IP* outperforms IDP-IP *even when the latter is configured with the best possible choice of* m. As for the Normal and Uniform, IDP-IP* is close to IDP-IP for the best choice of m, but is consistently faster than IDP-IP averaged over all possible choices of m. The Figure also shows that, without knowing how to best set m, the user can end up with an extremely poor performance when running IDP-IP instead of IDP-IP* (see how IDP-IP* is faster than the worst setting of IDP-IP by orders of magnitude for all distributions).

Finally, we benchmark IDP-IP* against an anytime version of IDP that was proposed by Service and Adams (2011). The basic idea of this algorithm is to add a preprocessing stage to IDP such that, if it is interrupted before completion, then the extra information obtain from preprocessing would be sufficient to construct a coalition structure that is guaranteed to be within a finite bound from CS^*. However, this algorithm requires 4 time as much memory compared to IDP or to IDP-IP*. Furthermore, as Figure 4 shows, IDP-IP* is significantly faster for all distributions.

5 Related Work

Our focus in this paper was on exact, anytime, CSG algorithms. Several anytime algorithms, other than the ones explained in this paper, were developed in the literature (see, e.g., (Sandholm et al. 1999; Dang and Jennings 2004; Rahwan, Michalak, and Jennings 2011)). These algorithms focus on (1) proposing a criteria based on which to divide the search space into subsets, and (2) a sequence in which these subsets are searched, such that the worst case bound on solution quality is guaranteed to improve after each subsets. In practice, however, the bounds generated by IP were shown to be significantly better.

Another class of CSG algorithms are *metaheuristics* which do not guarantee that an optimal solution is ever found, nor do they provide any guarantees on the quality of their solutions. However, they can usually be applied for very large problems (see, e.g., (Sen and Dutta 2000; Keinänen 2009; Shehory and Kraus 1998; Mauro et al. 2010)).

For completion, we refer to work on the CSG problem under compact representations of coalitional games. Ueda et al. (2010) considered the CSG problem under the

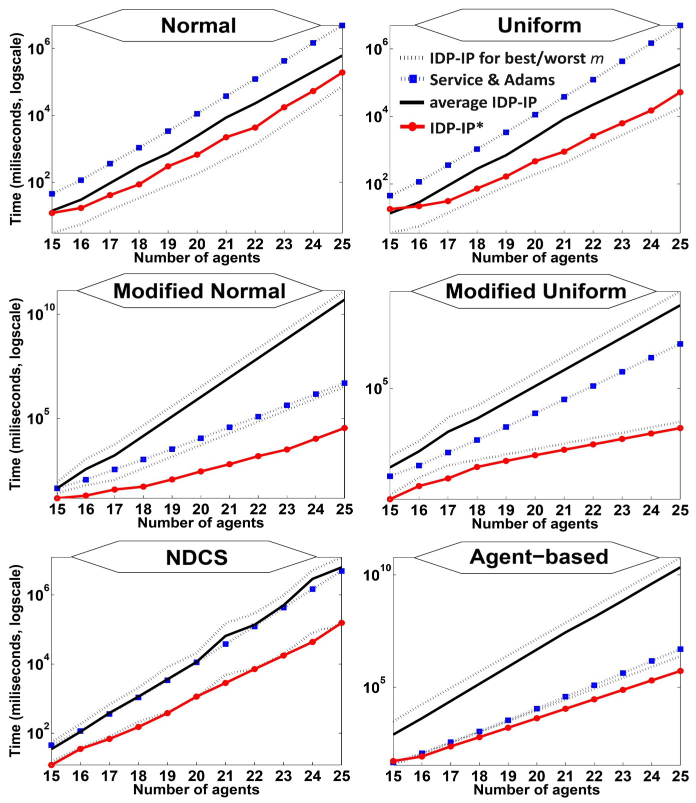


Figure 4: IDP-IP* vs. IDP-IP and Service&Adams (*log scale*).

DCOP (Distributed Constraint Optimization Problem) representation, while Ohta et al. (2009) considered it under the Marginal Contribution-net representation of Ieong and Shoham (2005). The skill-game settings were studied by Ohta et al. (2006) and Bachrach and Rosenschein (2008).

6 Conclusions

We presented IDP-IP*—a new combination of IDP and IP. Compared to the previous combination, i.e., IDP-IP, our algorithm is often faster, does not need prior knowledge of the value distribution, and does not hinder the anytime property of IP. Future work will focus on analysing the worst-case complexity of IDP-IP*, and on combining IP with the version of IDP that was proposed by Service and Adams (2011).

References

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

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

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

Dang, V. D.; Dash, R. K.; Rogers, A.; and Jennings, N. R. 2006. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In *AAAI-06*, 635–640.

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

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

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

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

Ohta, N.; Iwasaki, A.; Yokoo, M.; Maruono, K.; Conitzer, V.; and Sandholm, T. 2006. A compact representation scheme for coalitional games in open anonymous environments. In *AAAI'06: 21st National Conference on Artificial Intelligence*, 697–702.

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

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

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

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

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

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

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

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

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

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

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

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

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

Tsvetovat, M.; Sycara, K.; Chen, Y.; and Ying, J. 2000. Customer Coalitions in the Electronic Marketplace. In *ICMAS-00, New York*.

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

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