A Distributed Algorithm for Anytime Coalition Structure Generation

Tomasz Michalak[1], Jacek Sroka[2], Talal Rahwan[1], Michael Wooldridge[3], Peter McBurney[3], and Nicholas R. Jennings[1]

[1]School of Electronics and Computer Science, University of Southampton, UK
{tpm,tr,nrj}@ecs.soton.ac.uk
[2]Institute of Informatics, University of Warsaw, Poland
sroka@mimuw.edu.pl
[3]Department of Computer Science, University of Liverpool, UK
{mjw,mcburney}@liverpool.ac.uk

ABSTRACT

A major research challenge in multi-agent systems is the problem of partitioning a set of agents into mutually disjoint coalitions, such that the overall performance of the system is optimized. This problem is difficult because the search space is very large: the number of possible coalition structures increases exponentially with the number of agents. Although several algorithms have been proposed to tackle this *Coalition Structure Generation (CSG)* problem, all of them suffer from being inherently centralized, which leads to the existence of a performance bottleneck and a single point of failure. In this paper, we develop the first decentralized algorithm for solving the CSG problem optimally. In our algorithm, the necessary calculations are distributed among the agents, instead of being carried out centrally by a single agent (as is the case in all the available algorithms in the literature). In this way, the search can be carried out in a much faster and more robust way, and the agents can share the burden of the calculations. The algorithm combines, and improves upon, techniques from two existing algorithms in the literature, namely DCVC [5] and IP [9], and applies novel techniques for filtering the input and reducing the inter-agent communication load.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General

General Terms

Algorithms, Design, Experimentation

Keywords

Coalition Formation, Distributed AI, Algorithm Design

Cite as: A Distributed Algorithm for Anytime Coalition Structure Generation, T. Michalak, J. Sroka, T. Rahwan, M. Wooldridge, P. McBurney and N.R. Jennings, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. XXX-XXX.

1. INTRODUCTION

Coalition formation is an important topic in multi-agent systems, as there are many activities that coalitions can carry out better than single agents: for example, in e-commerce, coalitions often manage to buy (or sell) at lower (higher) unit prices than individuals alone [2, 3, 13]. In this context, a great deal of attention has been paid to coalition structure generation (CSG) in characteristic function games. Specifically, given a characteristic function that assigns a *coalition value* to every possible *coalition* (i.e., subset of agents), the CSG problem involves finding a *coalition structure* (i.e., a collection of pair-wise disjoint coalitions whose union yields the entire set of agents) that is *optimal* (i.e., in which the sum of coalition values is maximized). This is a well-known NP-hard combinatorial optimization problem for which a number of algorithms have been proposed in recent years. Following [6], the available CSG algorithms can be classified into three main categories, each with its own advantages and limitations:

1. **Dynamic programming** [14, 6]: Here, the basic idea is to break the optimization problem into sub-problems that can be solved recursively, and then combine the results to solve the original problem, thereby avoiding the work of recomputing the answer every time the sub-problem is encountered. While these algorithms are guaranteed, given n agents, to find an optimal coalition structure in $O(3^n)$ steps, they do not possess anytime properties. This means that no partial or interim solutions are available, which is undesirable, especially given large numbers of agents, since the time required to return an optimal solution might be longer than the time available to the agents.

2. **Heuristics** [11, 12]: These algorithms return "good" solutions relatively quickly, and scale up well with the increase in the number of agents involved. However, they provide no guarantees on the quality of their solutions, i.e., a solution could be arbitrarily worse than the optimal one.

3. **Anytime optimal algorithms** [10, 1, 9]: These algorithms initially generate a solution that is guaranteed

to be within a bound from the optimal. This result is then improved by evaluating more of the search space and establishing progressively better bounds until optimality is reached. Although these algorithms might, in the worst case, end up searching the entire space (i.e., they run in $O(n^n)$ time), their anytime property makes them more robust against failure due to premature termination.

Recently, Rahwan et al. developed a hybrid algorithm, called IDP-IP, that combines the state-of-the-art dynamic programming algorithm, IDP [6], with the state-of-the-art anytime algorithm, IP [9]. In particular, IDP-IP has been shown to exploit the strengths of both approaches and, at the same time, avoid their main weaknesses [7]. This makes IDP-IP the most efficient CSG algorithm reported to date. Therefore, we compare against it when evaluating the speed up obtained by our distributed approach.

An important point to note, here, is that *all* the aforementioned algorithms that solve the CSG problem optimally, i.e., those in categories 1 and 3, assume the existence of a center that has access to all the coalition values and carries out all the calculations. However, being centralized leads to the existence of a performance bottleneck and a single point of failure, which reduces the overall efficiency and robustness of the system. What is more desirable, then, is to be able to distribute the computation among the agents, instead of having it done centrally by one agent. Such distributed processing makes the system more robust against failure, and typically reduces the time for processing (modulo communications overheads) as a result of utilizing all the available resources in the system. However, a distributed algorithm that involves exchanging all the possible coalition values among the agents would be extremely slow due to the heavy communication overhead. Therefore, it is critical that the number of coalitions to be taken into consideration is reduced by orders of magnitude.

In this context, the only decentralized algorithm that has been developed in the CSG literature is due to Shehory and Kraus [12]. Although this algorithm is easily implementable, it uses a simple, greedy, heuristic that provides no guarantees on finding an optimal coalition structure. Moreover, it has been shown to be extremely inefficient in terms of load balancing, as well as communication and memory requirements, not to mention the exponential number of calculations that are redundantly[1] carried out by the agents (for more details, see [5]). Against this background, there is a need for a decentralized algorithm that can efficiently distribute the computations among the agents and return an optimal solution. In order to develop such an algorithm, we need to relax the assumption of having a center containing all the coalition values. In other words, the input itself needs to be computed in a distributed manner. Given the above requirements, we advance the state of the art in the following ways:

- We propose the first decentralized algorithm, called D-IP, that solves the CSG algorithm optimally. This algorithm combines, and improves upon, techniques from two existing algorithms in the literature, namely DCVC (the state-of-the-art algorithm for distributing the coalition value calculations among the agents [5]), as well as IP (which has been shown to be extremely efficient for optimal coalition structure generation [9]).[2] We also introduce novel techniques to facilitate the process of distributing the search so that the computational load is balanced among the agents, and virtually no redundant calculations are performed.[3]

- We develop novel techniques for filtering the input and reducing the communication overhead between the agents. For instance, our experiments show that, for uniformly distributed coalition values and 26 agents, on average only 9000 coalition values out of 67 million (i.e., 0.013%) are exchanged among the agents before an optimal coalition structure is reached. These filter rules only require performing a number of calculations linear in the size of the input. More importantly, these rules can also be incorporated into other optimal CSG algorithms as a preprocessing stage.[4]

- When evaluating the speed up obtained by our distributed approach, as opposed to a centralized one, we compare DIP with the state-of-the-art centralized algorithm, IDP-IP, and show that D-IP is significantly faster. For example, given 26 agents and a uniform distribution, D-IP only takes 14.4% of the time required by IDP-IP to return an optimal solution).

The remainder of this paper is organized as follows. Section 2 contains the basic notation. Section 3 analyzes the way DCVC and IP operate. Section 4 presents our distributed algorithm. Section 5 presents experimental evaluation. Section 6 concludes and presents future work.

2. BASIC NOTATION

Let n be the number of agents, and $A = \{a_1, a_2, \dots, a_n\}$ be the set of agents. A subset of agents (i.e., a coalition) is typically denoted by $C \subseteq A$, and the set of possible coalitions is denoted by 2^A. A coalition structure, CS, is a partition of A. Let Π denote the set of all coalition structures (partitions) of A. That is, $\Pi = \{CS \subset 2^A | \cup_{C \in CS} C = A \wedge \forall C, C' \in CS : C \cap C' = \emptyset\}$. Now, given a characteristic function $v : 2^A \rightarrow \Re$ that assigns a value to every coalition, and given that the value of any coalition structure $CS \in \Pi$ is computed as follows: $V(CS) = \sum_{C \in CS} v(C)$, our goal is to find an optimal coalition structure CS^* such that: $CS^* = \arg\max_{CS \in \Pi} V(CS)$. We will be concerned

[1]Here, a redundant calculation corresponds to a coalition structure being evaluated by more than one agent.

[2]We choose IP, instead of the hybrid IDP-IP algorithm, since the latter incorporates dynamic programming techniques which are very difficult to decentralize. This is because they assume the existence of a central table in that partial solutions are stored, and this entire table is heavily used throughout the search process.

[3]In fact, only one coalition is evaluated more than once, and that is the one containing exactly n coalitions, where n is the number of agents.

[4]A very preliminary report on this line of research can be found in [4].

with algorithms that progressively search the space of possible coalition structures, and we denote by CS^*_N the best of all the solutions that have been examined at a given instance, i.e., the one with the highest value. Our goal is then to have: $V(CS^*_N) = V(CS^*)$. In cases where only a sub-optimal solution is required that is within a finite ratio bound β from the optimal, then our goal would be to have: $V(CS^*_N) \times \beta \geq V(CS^*)$. For example, if we have $\beta = \frac{4}{3}$, then we need to find a solution of which the value is *guaranteed* to be at least 75% of $V(CS^*)$.

3. ANALYZING DCVC AND IP

Since our D-IP algorithm is built upon two algorithms from the literature, namely DCVC and IP, in this section we briefly explain the way these algorithms work.

3.1 DCVC

This is the state-of-the-art algorithm for distributing the coalition value calculations among the agents. Its main idea is to represent the space of all feasible coalitions in the form of structured lists so as to facilitate an efficient distribution of the computational load among the agents. In more detail, the input is represented as n lists, namely $L_1, L_2, \ldots, L_n$, such that $L_s : s \in \{1, 2, \ldots, n\}$ contains all the coalitions of size s ordered lexicographically. Every list L_s is then divided into $\lfloor \frac{|L_s|}{n} \rfloor$ non-overlapping segments such that agent a_1 computes the values for all the coalitions in the first segment, a_2 the values for those in the second segment, and so on. Note that, in order to save memory, each agent that uses DCVC maintains only one coalition at a time instead of maintaining the entire list of coalitions. Therefore, knowing *where* an agent's share is located in L_s is not sufficient to know directly *which* coalitions are contained in this share. Therefore, in order to cycle through the coalitions one by one, each agent computes the index at which its share ends, computes the coalition located at that index, and then generates the remaining coalitions one by one. For that DCVC uses an efficient technique that computes any coalition C by either knowing the index of C in L_s, or by knowing the coalition that is located directly below C in L_s. DCVC also uses an easily-implementable procedure that deals with any left-over coalitions that could result from $|L_s|$ being not exactly divisible by n (for more details, see [5]). Figure 1 shows an example of the resulting assignment given 6 agents.[5]

3.2 IP

IP is built upon a novel representation of the search space in which all coalition structures are divided into subspaces based on the sizes of the coalitions they contain. Specifically, each subspace is represented by a unique integer partition of n (e.g., given $n = 4$, the possible integer partitions are $I_1 = [4]$, $I_2 = [1, 3]$, $I_3 = [2, 2]$, $I_4 = [1, 1, 2]$ and $I_5 = [1, 1, 1, 1]$). In more detail, an integer partition I corresponds to a subspace S_I containing all the coalition structures in which the coalition sizes match the parts in I (e.g., $S_{[1,1,2]}$ contains all the coalition structures in which two coalitions

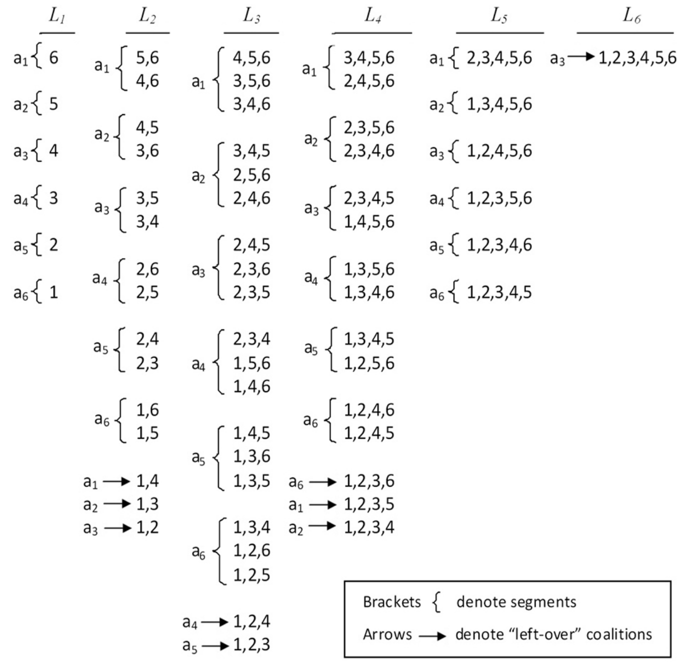


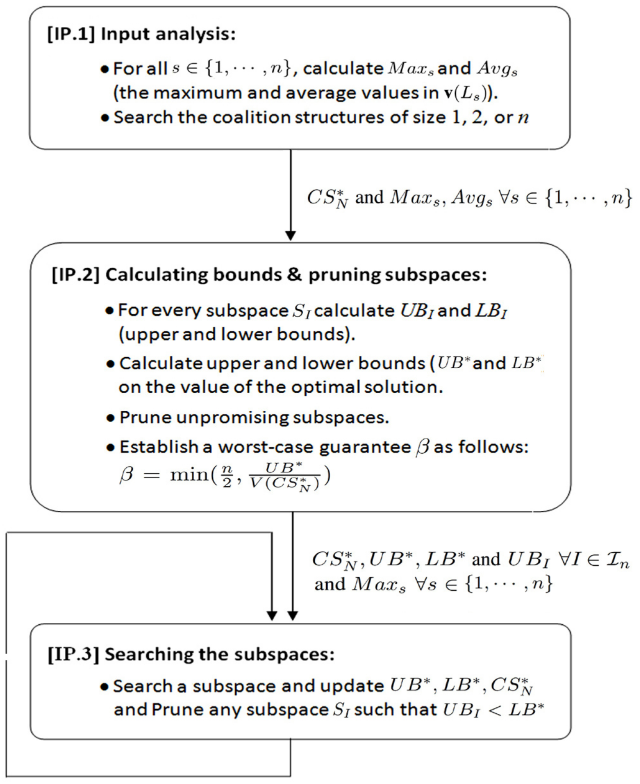
Figure 1: An example of the resulting assignment in DCVC.

are of size 1, and one coalition is of size 2). We will denote by $\mathcal{I}_n$ the set of possible integer partitions of n.

In the remainder of this section, we will analyze the way IP utilizes this integer-partition based representation. In particular, we will divide the basic operation of IP into three main stages, and focus on the input data needed at each stage (see Figure 2).

[IP.1] Input analysis: The input to the IP algorithm consists of ordered lists of coalition values, denoted $\mathbf{v}(L_s)\ \forall s \in \{1, \ldots, n\}$, where a value located at any given index in $\mathbf{v}(L_s)$ corresponds to the coalition located at that same index in L_s. For every list $\mathbf{v}(L_s)$, IP scans the input in order to obtain the maximum and average values, denoted Max_s and Avg_s respectively. While scanning the input, the algorithm also evaluates particular coalition structures. More specifically, the coalition structure that contains n coalitions is evaluated by summing the values in $\mathbf{v}(L_1)$, and the one containing exactly 1 coalition is evaluated by checking the value in $\mathbf{v}(L_n)$. Finally, the coalition structures of size 2 are evaluated while scanning two lists at a time — $\mathbf{v}(L_i)$ and $\mathbf{v}(L_{n-i})$ where $i \in \{1, 2, \ldots, \lfloor n/2 \rfloor\}$ — starting at different extremities for each list. This is based on the fact that any two complementary coalitions, $C, C' : C \cap C' = \emptyset, C \cup C' = A$, are always diametrically positioned in $L_{|C|}$ and $L_{|C'|}$ [9]. For example, given 6 agents, coalitions $\{a_1\}$ and $\{a_2, a_3, a_4, a_5, a_6\}$ are diametrically positioned in the lists L_1 and L_5 respectively, and coalitions $\{a_1, a_2, a_3\}$ and $\{a_4, a_5, a_6\}$ are diametrically positioned in the list L_3 (see Figure 1). This implies that, by summing every possible pair of values in $\mathbf{v}(L_i)$ and $\mathbf{v}(L_{n-i})$ that are located at indices j and $|L_i| - j$, IP ends up evaluating every possible coalition structure of size 2.

[5]For presentation clarity, coalitions are represented in this, and other figures in the paper, using only the indices of the agents (e.g., 1, 5, 6 represents coalition $\{a_1, a_5, a_6\}$).



[IP.1] Input analysis:
• For all $s \in \{1, \cdots, n\}$, calculate Max_s and Avg_s (the maximum and average values in $v(L_s)$).
• Search the coalition structures of size 1, 2, or n
CS^*_N and $Max_s, Avg_s\ \forall s \in \{1, \cdots, n\}$
[IP.2] Calculating bounds & pruning subspaces:
• For every subspace S_I calculate UB_I and LB_I (upper and lower bounds).
• Calculate upper and lower bounds (UB^* and LB^* on the value of the optimal solution.
• Prune unpromising subspaces.
• Establish a worst-case guarantee β as follows: $\beta = \min(\frac{n}{2}, \frac{UB^*}{V(CS^*_N)})$
CS^*_N, UB^*, LB^* and $UB_I\ \forall I \in \mathcal{I}_n$ and $Max_s\ \forall s \in \{1, \cdots, n\}$
[IP.3] Searching the subspaces:
• Search a subspace and update UB^*, LB^*, CS^*_N and Prune any subspace S_I such that $UB_I < LB^*$

Figure 2: The main stages of IP

[IP.2] Calculating bounds and pruning subspaces: For every subspace S_I, the algorithm computes upper and lower bounds, denoted UB_I and LB_I respectively, on the value of the best coalition structure that could be found in that subspace:

$$UB_I = \sum_{s \in I} Max_s \quad \text{and} \quad LB_I = \sum_{s \in I} Avg_s \tag{1}$$

By using these bounds, it is possible to compute an upper bound UB^* and a lower bound LB^* on the value of the optimal coalition structure as follows:

$$UB^* = \max\left(V(CS^*_N),\ \max_{I \in \mathcal{I}_n} UB_I \right) \tag{2}$$

$$LB^* = \max\left(V(CS^*_N),\ \max_{I \in \mathcal{I}_n} LB_I \right) \tag{3}$$

These bounds, in turn, enable IP to identify (and consequently prune) any subspaces that have no potential of containing an optimal coalition structure. Finally, IP establishes a worse-case guarantee $\beta = \min(\frac{n}{2}, \frac{UB^*}{V(CS^*_N)})$ on the value of the best solution found so far (see [9] for more details).

[IP.3] Searching the subspaces: The order by which IP searches the subspaces is based on their upper bounds, starting with the one with the highest upper bound, and then the second highest, and so on. Now, given an integer partition $I = \{i_1, i_2, \ldots, i_m\}$, one way of searching the corresponding subspace S_I is by simply going through the cartesian product of the lists $L_{i_1}, L_{i_2}, \ldots, L_{i_m}$ and checking every combination to determine whether it is *invalid* or *redundant*. More specifically, an invalid combination is one in which the coalitions overlap, and a redundant combination is one that has already been examined with a different ordering of the coalitions (e.g., given $S_{\{2,2,3\}}$, and having examined $\{\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6, a_7\}\}$, the following combination no longer needs to be examined and is, therefore, considered redundant $\{\{a_3, a_4\}, \{a_1, a_2\}, \{a_5, a_6, a_7\}\}$). This search technique, while correct, is inefficient since it basically involves searching through the cartesian product of $L_{i_1}, \ldots, L_{i_m}$, and this is a significantly bigger space compared to S_I (e.g., given the integer partition $\{1, 2, 3, 4, 5, 6, 7\}$, the cartesian product of $L_1, \ldots, L_7$ is nearly 1.3×10^{10} times bigger than $S_{\{1,2,3,4,5,6,7\}}$). To avoid this, IP uses a *depth-first* search technique that cycles through L_{i_1} and, for every coalition $C_1 \in L_{i_1}$, cycles through only the coalitions in L_{i_2} that do not overlap with C_1 and are guaranteed not to lead to redundant coalition structures. Similarly, given any two coalitions $C_1 \in L_{i_1}, C_2 \in L_{i_2}$, it only cycles through the relevant coalitions in L_{i_3}, and so on until it reaches L_{i_m}.

To speed up the search ever further, IP applies a *branch-and-bound* technique. In particular, given some coalitions $C_1 \in L_{i_1}, C_2 \in L_{i_2}, \ldots, C_k \in L_{i_k} : k < m$, and before cycling through the relevant coalitions in $L_{i_{k+1}}, \ldots, L_{i_m}$, the algorithm checks whether:

$$v(C_1) + \cdots + v(C_k) + Max_{i_{k+1}} + \cdots + Max_{i_m} < V(CS^*_N) \tag{4}$$

Now if the above condition holds, then this means none of the coalition structures containing $C_1, \ldots, C_k$ can improve upon the quality of the best solution found so far, in which case, the algorithm skips this part of the search (again see [9] for more details).

4. THE D-IP ALGORITHM

Having explained how DCVC and IP work, we now present our distributed algorithm, D-IP, which builds upon the above two algorithms. In particular, after dividing the operation of IP into three stages and identifying the required data for each one of them, we now modify and improve upon these stages so that they can be carried out more efficiently and in a distributed manner, and then introduce new communication stages to ensure that each agent has the required data for each computational stage. This modification involves (1) incorporating a modified version of DCVC to distribute the first stage of IP, (2) using new filter rules to significantly reduce the communication bottleneck that occurs between the second and third stage, (3) introducing a new technique to reduce the computational bottleneck that occurs in the third stage, and (4) incorporating a load balancing technique that is necessary since the number of computations involved in each agent's share of the search could differ significantly due to the branch-and-bound technique used during the search. The algorithm's main stages, which can be seen in Figure 3, are each discussed in detail in the following subsections.

[D-IP.1] Distributed input generation and analysis: Since the DCVC algorithm is inherently distributed, it constitutes a natural starting point for our distributed approach

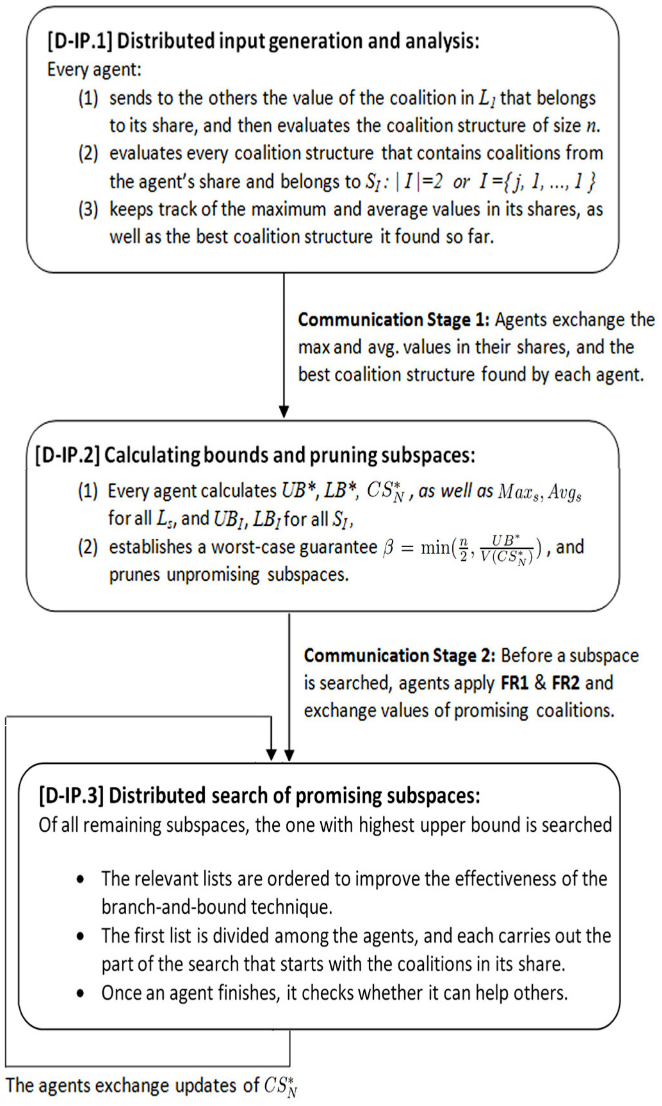

[D-IP.1] Distributed input generation and analysis:
Every agent:
(1) sends to the others the value of the coalition in L_1 that belongs to its share, and then evaluates the coalition structure of size n.
(2) evaluates every coalition structure that contains coalitions from the agent's share and belongs to S_I: $|I|=2$ or $I=\{j,1,\ldots,1\}$
(3) keeps track of the maximum and average values in its shares, as well as the best coalition structure it found so far.
Communication Stage 1: Agents exchange the max and avg. values in their shares, and the best coalition structure found by each agent.
[D-IP.2] Calculating bounds and pruning subspaces:
(1) Every agent calculates UB^*, LB^*, CS^*_N, as well as Max_s, Avg_s for all L_s, and UB_I, LB_I for all S_I,
(2) establishes a worst-case guarantee $\beta = \min(\frac{n}{2}, \frac{UB^*}{V(CS^*_N)})$, and prunes unpromising subspaces.
Communication Stage 2: Before a subspace is searched, agents apply FR1 & FR2 and exchange values of promising coalitions.
[D-IP.3] Distributed search of promising subspaces:
Of all remaining subspaces, the one with highest upper bound is searched
The relevant lists are ordered to improve the effectiveness of the branch-and-bound technique.
The first list is divided among the agents, and each carries out the part of the search that starts with the coalitions in its share.
Once an agent finishes, it checks whether it can help others.
The agents exchange updates of CS^*_N

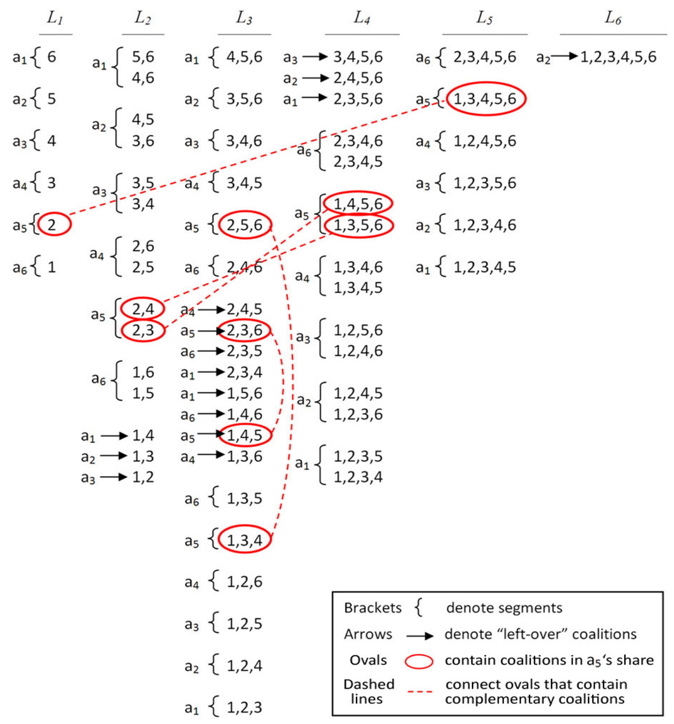
Figure 3: The main stages of D-IP

to the CSG problem. Based on this, we modify DCVC in such a way that it becomes possible for the agents to perform the input-analysis stage of IP in a distributed manner and with very little communication. In this context, recall that the original DCVC algorithm divides each list L_s into $\left\lfloor\frac{|L_s|}{n}\right\rfloor$ consecutive segments, and assigns the first segment to a_1, the second to a_2, and so on. Whereas such a distribution is natural and intuitive, it makes it difficult, if not impossible, for each agent to evaluate any coalition structures using only the coalitions in its share. This is because the agent needs to search through the possible combinations of these coalitions, which is not trivial due to the exponential number of such combinations, and there are no guarantees that any of these combinations would be a valid coalition structure. In Figure 1, for example, not even one coalition structure can be constructed using only the coalitions in agent a_5's share. With this in mind, we modify DCVC as follows:

- While every list $L_i : i \in \{1, \ldots, \lceil\frac{n}{2}\rceil - 1\}$ is divided into segments and left-over coalitions as in DCVC, the list L_{n-i} is divided in a reversed order. That is, the segments start from the bottom of the list, instead of the top, and the left-over coalitions, if there are any, appear at the top of the list. Moreover, if n is an even number, then the list $L_{n/2}$ is split in two, where the upper half is assigned as in DCVC (i.e., from top to bottom), and the lower one is assigned in a reversed order (i.e., from bottom to top). Figure 4 shows an example for 6 agents. This modification will be used to enable each agent to efficiently evaluate certain coalition structures using only the coalitions in its share (see below for more details).
- At the beginning of the computations, each agent a_j sends to the others the value of the coalition in L_1 that belongs to a_j's share. By so doing, every agent would have the values of all singletons, i.e., coalitions that contain exactly one agent. After that, each agent evaluates the coalition structure of size n, and that is by summing the values of singletons.
- Each agent cycles simultaneously through its share of L_i and $L_{n-i} : i \in \{1, \ldots, \lfloor\frac{n}{2}\rfloor\}$, starting at different extremities for each share. Note that, due to our modified assignment of the shares, there will be no overlap between any two coalitions $C' \in L_i, C'' \in L_{n-i}$ for which the agent is simultaneously calculating the values (in Figure 4, see how the connected coalitions in agent a_5's share do not overlap). Based on this, and after calculating the values of C' and C'', the agent evaluates the coalition structure $\{C', C''\}$ by summing these two values. Moreover, since the agent already knows the values of singletons, it also evaluates $\{C', \{a_j\}_{a_j \in C''}\}$ and $\{\{a_j\}_{a_j \in C'}, C''\}$.
- Each agent keeps track of the maximum and average values in its share of every list, as well as the best coalition structure that it found so far.

The above modifications ensure that the coalition structures of sizes 1, 2, and n are evaluated in a decentralized manner. This is particularly interesting since it has been shown that, by evaluating exactly these coalition structures, it is possible to establish a worst-case guarantee $\beta = \frac{n}{2}$ on the quality of the best solution found [10]. Our modified version of DCVC also enables the agents to search, in a decentralized manner, through $S_{I \in \mathcal{I}_n} : I = \{k, 1, 1, \ldots, 1\}$ for all $k \in \{2, 3, \ldots, n-1\}$.

Communication stage 1: Now that the agents have carried out the first stage of IP in a distributed manner, they next need to carry out the second one, and in order to do so, they each need to have the required data at hand. In particular, as mentioned in Section 3.2, the second stage of IP requires CS^*_N as well as Max_s and Avg_s for all $s \in \{1, \ldots, n\}$ (see Figure 2). Note that each agent at this point knows only the maximum and average values of the coalitions in its share, as well as the best of all the coalition structures that it found, but this is not sufficient to compute the required data, especially since each agent does not know the values



L1
L2
L3
L4
L5
L6
Brackets { denote segments
Arrows → denote "left-over" coalitions
Ovals contain coalitions in a5's share
Dashed lines connect ovals that contain complementary coalitions

Figure 4: The resulting distribution of our modified version of DCVC given 6 agents.

of the coalitions in the other agents' shares. Therefore, the agents exchange their maximum and average values as well as the best coalition structure that each of them found and, using this information, each agent then independently computes CS^*_N and $Max_s, Avg_s\ \forall s \in \{1,\dots,n\}$.

[D-IP.2] Calculating bounds and pruning subspaces: Using the data transmitted above, each agent is able to independently compute Max_s and Avg_s for every list L_s and, consequently, can compute UB_I and LB_I for every subspace S_I as in equation (1). Moreover, each agent independently computes CS^*_N as well as UB^* and LB^* using equations (2) and (3) respectively. Finally, it establishes a worst-case guarantee $\beta = \min(\frac{n}{2}, \frac{UB^*}{V(CS^*_N)})$ and prunes any unpromising subspaces.

Communication stage 2: At this point, the agents would have performed the first two stages of IP in a distributed manner, and they now need to search through the remaining subspaces if there are any. Specifically, in order to search through a subspace S_I, the agents first need to exchange the values of the coalitions in the lists $L_s : s \in I$. Now since these lists can be exponentially long, we apply what we call *filter rules* to determine *a priori* which coalitions cannot be in an optimal coalition structure so that the agents avoid exchanging them among themselves. More specifically, we use two filter rules, namely **FR1** and **FR2**:

Filter Rule 1 (FR1): *Given a coalition $C \in 2^A$, if the following condition holds then do not consider C since it cannot be part of an optimal coalition structure:*

$$v(C) < \sum_{a_j \in C} v(\{a_j\})$$

For instance, if $v(\{a_1a_2\}) < v(\{a_1\}) + v(\{a_2\})$, then any coalition structure $CS \ni \{a_1a_2\}$ cannot be optimal because we can replace $\{a_1a_2\}$ with $\{a_1\},\{a_2\}$ and end up with another coalition structure CS' such that $V(CS') > V(CS)$. Based on this, any coalition that satisfies **FR1** is not exchanged among the agents. Note, however, that the effectiveness of **FR1** depends on the values of singletons with respect to the values of other coalitions. While, for both normal and uniform distributions, this rule filters an average of 50% of all coalitions, in the worst case not even one coalition is filtered. Based on this, we propose the second filter rule:

Filter Rule 2 (FR2): *Given a subspace S_I and a coalition $C : |C| \in I$, if the following condition holds then do not consider C since the best coalition structure, i.e., the one with the highest value, in S_I cannot contain it:*

$$v(C) + UB_{I_k} - Max_{|C|} < V(CS^*_N) \qquad (5)$$

The logic behind **FR2** is as follows. While $Max_{|C|}$ is used when computing UB_I, we replace it with $v(C)$ and, hence, obtain a tighter upper bound on the values of all the coalition structures in S_I that contain C. Now, if this new bound happens to be smaller than $V(CS^*_N)$, then any coalition structure containing C can be skipped while searching S_I. Conceptually, **FR2** is similar to the branch-and-bound technique of [9] except that **FR2** is applied before, and not during, the search of a subspace.

A key point to note here is that, unlike **FR1** which is independent of any subspace, **FR2** depends on the subspace being searched. Therefore, it might intuitively seem that any coalition filtered using **FR2** while searching a particular subspace can still be useful while searching another, in which case it still needs to be exchanged among the agents. However, against this intuition the following lemma holds.

LEMMA 1. *If a coalition C is filtered using **FR2** given a particular subspace S_I, then it will also be filtered using **FR2** given any other subspace $S_J : UB_J \leq UB_I$.*

Now since the subspaces are searched starting with the one with the highest upper bound then the second highest etc., the above lemma implies that, after the promising coalitions from a list have been exchanged (while searching some subspace S_I), no other coalition from this list will have to be exchanged again while searching other subspaces.

A key point to note, here, is that the above two filter rules can also be incorporated into other optimal CSG algorithms as a preprocessing stage to reduce the number of coalitions taken into consideration, especially since these rules only require performing a number of calculations that grows linearly with the size of the input.

[D-IP.3] Distributed search of promising subspaces: The order by which the subspaces are searched is similar to IP, i.e., based on their upper bounds. As mentioned above, before a subspace $S_{\{i_1,\ldots,i_m\}}$ can be searched in a distributed manner, the agents first exchange their filtered shares of the lists $L_{i_1},\ldots,L_{i_m}$. Now, given the depth-first nature of the search technique used by IP, it is possible to distribute the calculations among the agents by simply dividing the filtered coalitions of the first list (i.e., L_{i_1}) into segments and leftovers (as in DCVC) among the agents. Then, every agent a_i cycles through the coalitions in its share of L_{i_1} and, for every such coalition, cycles through the relevant coalitions in $L_{i_2},\ldots,L_{i_m}$ as in IP (see section 3.2 for more details). In order to speed up this distributed search, we modify it as follows:

- Although the agents' shares of L_{i_1} are equal, this does not necessarily mean that the agents will each evaluate the same number of coalition structures, and that is mainly due to the branch-and-bound technique being applied during the search (i.e., some agents might prune more coalition structures than others). Based on this, once an agent a_i finishes its calculations, it sends a signal to another randomly-chosen agent a_j which, in turn, responds with the number of coalitions that are yet to be examined in its share of L_{i_1}. These coalitions are then divided between a_i and a_j without the need for further communication, and that is because a_i (like every other agent) already has the values of all filtered coalitions in L_{i_1}, including those in a_j's share.

- While the ordering of the lists does not affect the correctness of the search (i.e., it doesn't matter which list comes first and which comes second etc.), it does indeed affect its efficiency. This is due to the branch-and-bound technique being used, which is based on equation (4). In particular, by looking at the equation, we can see that any coalition structure pruned given a particular ordering of its coalitions might not be pruned at all given another ordering, or might be pruned but at a later stage of the search (i.e., it might be pruned while cycling through the i^{th} list, instead of the j^{th}, where $i > j$). Based on this, we optimize the ordering of the lists before any subspace is searched. More specifically, for every list, we estimate the number of coalitions at which the branch-and-bound technique is *activated* (i.e., at which the depth-first search stops going any deeper into the search tree). These estimations are then used to order the lists based on the proportion of coalitions that activate the branch-and-bound technique in each list. We first put the list with the highest proportion, and then the second highest, and so on. Our estimations are based on the following lemma, which comes from comparing equations (4) and (5):

  LEMMA 2. *If a coalition is filtered by **FR2** given a particular subspace, then this coalition will activate branch-and-bound if it is encountered during the search of that subspace.*

  This implies that, for any given list L_s, and for any subspace $S_I : I \ni s$, the number of coalitions that are filtered by **FR2** is actually an upper bound on the number of coalitions that activate branch-and-bound in that list. Against this background, we assume in our heuristics that the lists with higher upper bounds have higher proportions of coalitions activating branch-and-bound. This rather simple heuristic can lead to substantial improvements (see Section 5 for more details). More importantly, this rules can be incorporated into other centralized algorithms as a preprocessing stage to reduce the number of coalitions that are taken into consideration.

Having explained every stage of D-IP in detail, we now evaluate its performance in the following section.

5. EXPERIMENTAL EVALUATION

In order to evaluate the performance of D-IP, we implemented it using JADE[6] (Java Agent DEvelopment Framework), and used 14 Intel Core 2 Duo 2.0 GHz workstations that are connected via a 1Gb Ethernet network to simulate up to 28 agents.

Following, among others, Sandholm et al. [10] and Rahwan et al. [8], we consider two probability distributions of coalition values:

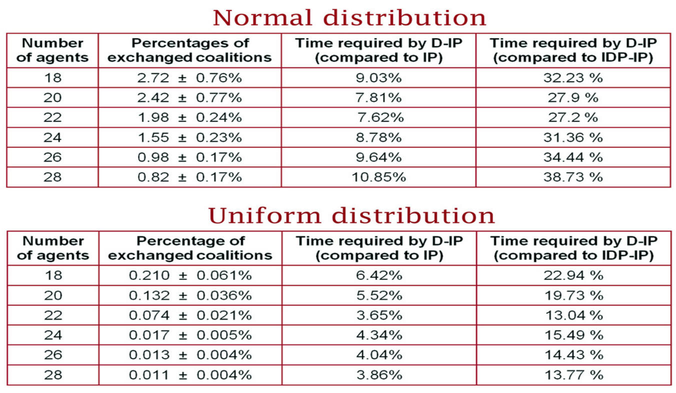
- Normal: $v(C) = max(0, |C| \times p)$, for $p \in N(1, 0.1)$;
- Uniform: $v(C) = max(0, |C| \times p)$, for $p \in U(0, 1)$;

In our experiments, we measure the time required by D-IP to run to completion, comparing with both IP (the centralized version of D-IP) as well as IDP-IP (the current state-of-the-art algorithm).[7] We also calculate the percentage of coalitions that need to be exchanged among the agents. In this context, note that the main cost to our distributed approach, compared to other centralized ones, is the communication required among the agents. Therefore, it is crucial that we reduce this cost as much as possible. Finally, regarding the anytime property, instead of showing how the solution quality and worst-case guarantees grow over the running time of the algorithm, we simply refer the reader to [9], and that is because D-IP is almost identical to IP in this aspect.

Our simulation results are shown in Figure 5. As can be seen, our filter rules significantly reduce the percentage of coalitions that are exchanged among the agents, e.g., given 26 agents and a uniform distribution, on average only 9000 out of 67 million coalitions need to be exchanged, that is 0.013%. Moreover, this percentage monotonically decreases as the number of agents increases for both normal and uniform distributions. It can also be seen from the figure that D-IP is significantly faster than IP (e.g., given 26 agents, it only takes 9.6% of the required time given a normal distribution, and only 4.04% given a uniform distribution). These improvements come from our more-informed way of ordering the lists, as well as the fact that the computations are now done in parallel, and involve a smaller number of coalitions,

[6]For more details, see `http://jade.tilab.com`.

[7]Although Rahwan et al. did not explicitly specify in [9] and [7] how the lists were ordered in their simulations, the authors have informed us that the lists were actually ordered in a descending order based on the size of the coalitions in each list.



Normal distribution

Number of agents	Percentages of exchanged coalitions	Time required by D-IP (compared to IP)	Time required by D-IP (compared to IDP-IP)
18	2.72 ± 0.76%	9.03%	32.23 %
20	2.42 ± 0.77%	7.81%	27.9 %
22	1.98 ± 0.24%	7.62%	27.2 %
24	1.55 ± 0.23%	8.78%	31.36 %
26	0.98 ± 0.17%	9.64%	34.44 %
28	0.82 ± 0.17%	10.85%	38.73 %

Uniform distribution

Number of agents	Percentage of exchanged coalitions	Time required by D-IP (compared to IP)	Time required by D-IP (compared to IDP-IP)
18	0.210 ± 0.061%	6.42%	22.94 %
20	0.132 ± 0.036%	5.52%	19.73 %
22	0.074 ± 0.021%	3.65%	13.04 %
24	0.017 ± 0.005%	4.34%	15.49 %
26	0.013 ± 0.004%	4.04%	14.43 %
28	0.011 ± 0.004%	3.86%	13.77 %

Figure 5: Simulation results

i.e., only the filtered ones. Finally, the figure shows that D-IP is consistently faster than IDP-IP, the fastest available algorithm in the literature. This means that, by developing D-IP, we do not only provide the first decentralized algorithm for optimal coalition structure generation, but we also set a new benchmark in terms of the time required to solve this problem optimally.

6. CONCLUSIONS & FUTURE WORK

In this paper, we have proposed the first decentralized algorithm for optimal coalition structure generation. The distribution is performed in such a way that the computational load is balanced among the agents, and virtually no redundant calculations are performed. We have also introduced novel filtering rules that significantly reduce the communication requirements of our decentralized approach, and can also be incorporated into other centralized approaches as a preprocessing stage to reduce the number of coalitions taken into consideration if, for example, a significantly improved algorithm became available. Moreover, we have proposed a more-informed way of ordering the lists, compared to IP, when searching different subspaces. When evaluating the speed up obtained by our distributed approach, as opposed to centralized ones, we compared it with the state-of-the-art IDP-IP algorithm and showed that it is significantly faster. By using D-IP, the agents can now solve the CSG problem in a much faster and more robust manner, without having a performance bottleneck and a single point of failure.

Although we implement a more informed way of ordering the coalition lists during a subspace search, we believe this could still be improved even further. In particular, in our future work, we would like to determine whether there is an optimal way of ordering the lists. Moreover, while our filter rules significantly reduce the number of coalitions that need to be exchanged between the agents, we would like to try and reduce the communication overhead even further, by developing additional rules. Finally, we would like to consider other value distributions besides Normal and Uniform in our experiments, e.g. the NDCS distribution recently introduced by Rahwan et al. [9].

7. ACKNOWLEDGMENTS

Tomasz Michalak, Peter McBurney and Michael Wooldridge acknowledge support from the EPSRC under the project Market Based Control of Complex Computational Systems (GR/T10657/01). Tomasz Michalak acknowledges support from the EPSRC under the project ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC strategic partnership. We also thank the anonymous reviewers for their valuable comments. Finally, we thank Andy Dowell for his helpful discussions.

8. REFERENCES

[1] V. D. Dang and N. R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS-04*, pages 564–571, 2004.

[2] C. Li, U. Rajan, S. Chawla, and K. Sycara. Mechanisms for coalition formation and cost sharing in an electronic marketplace. In *ICEC '03: Proceedings of the 5th international conference on Electronic commerce*, pages 68–77, 2003.

[3] C. Li and K. Sycara. Algorithm for combinatorial coalition formation and payoff division in an electronic marketplace. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 120–127, 2002.

[4] T. P. Michalak, A. Dowell, P. McBurney, and M. Wooldridge. Pre-processing techniques for anytime coalition structure generation algorithms. In J.-J. C. Meyer and J. Broersen, editors, *KRAMAS*, volume 5605 of *LNCS*, pages 99–113. Springer, 2008.

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

[6] T. Rahwan and N. R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *AAMAS-08*, pages 1417–1420, 2008.

[7] T. Rahwan and N. R. Jennings. Optimal coalition structure generation: Anytime optimization meets dynamic progamming. In *AAAI-08*, pages 156–161, 2008.

[8] T. Rahwan, S. D. Ramchurn, A. Giovannucci, V. D. Dang, and N. R. Jennings. Anytime optimal coalition structure generation. In *AAAI-07*, pages 1184–1190, 2007.

[9] 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.

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

[11] S. Sen and P. Dutta. Searching for optimal coalition structures. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 286–292, 2000.

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

[13] J. Yamamoto and K. Sycara. A stable and efficient buyer coalition formation scheme for e-marketplaces. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 576–583, 2001.

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