On distributing coalition structure generation

Tomasz Michalak[1], Jacek Sroka[2], Michael Wooldridge[1], and Peter McBurney[1]

[1] Department of Computer Science, University of Liverpool, UK
{tomasz.michalak,mjw,mcburney}@liverpool.ac.uk
[2] Institute of Informatics, University of Warsaw, Poland
sroka@mimuw.edu.pl

Abstract. We propose an anytime algorithm for distributed coalition structure generation (CSG) in a cooperative environment. We build upon [1] and [2]. Following [2] each agent starts with a disjoint fraction of coalition input to calculate. However, we adapt this distribution so that agents are able to evaluate a number of key coalition structures using only their local knowledge. Consequently, agents independently and with little communication perform the input analysis phase of the [1] algorithm providing the worst case bound from the optimum. In the subspace search phase we propose a pre-processing method that significantly reduces the communication load between agents. For instance, for uniformly distributed coalition values and 26 agents, on average, only 9000 coalitions out of 67 million (*i.e.* 0.013%) have to be exchanged between agents. Furthermore, we show that information gathered during this pre-processing makes it possible to develop a more efficient search procedure. Importantly, this technique may improve the performance of not only the distributed but also the centralised version of the algorithm.

1 Introduction

An important issue in multi-agent system research is to generate an optimal coalition structure, *i.e.* a disjoint and exhaustive division of agents in a system, such that the joint welfare is maximized. Most of the literature analyzes this problem in the context of characteristic function games, where the performance of any coalition is not influenced by other coalitions in the system[3]. However, even under this assumption, the coalition structure generation (CSG) problem was proved to be NP-complete [3]. A number of approaches have been developed that help to circumvent this complexity. Following [4], the main approaches include:

1. **Dynamic programming:** Algorithms in this class employ dynamic programming techniques in order to avoid evaluating every coalition structure. The state-of-the-art *Improved Dynamic Programming* (IDP) algorithm of [4] advances the well-known work of [5] and guarantees that the solution will be found in $O(3^n)$ steps (where n is the number of agents).[4] However, this approach does not possess anytime properties, *i.e.* it does not output any solution until all the calculations are completed;

[3] As argued by [3] or [4] many (but, clearly, not all) real-world multiagent problems can be modelled using this representation.

[4] The terms "coalition structure" and "solution" will be used interchangeably.

2. **Heuristics:** In this approach the number of coalition structures to be searched is reduced by imposing some exogenous assumptions. For instance, [6] proposed a greedy algorithm that focuses only on coalition structures that contain coalitions of particular sizes;

3. **Anytime optimal algorithms:** 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 are $O(n^n)$), the anytime property has a number of advantages. Such algorithms deliver an outcome almost instantly and, in general, they are more robust against failure due to premature termination. The state of the art in this class is the *IP* algorithm [1];[5]

4. **Hybrid DP-anytime algorithm:** The algorithm of [8], called IDP-IP, combines dynamic programming techniques of IDP with the anytime IP algorithm exploiting the strengths of both approaches and, at the same time, avoiding their main weaknesses. This makes it the most efficient CSG algorithm reported to date.

Although, all the above approaches are centralised, it might be more efficient to distributed computations among agents. This not only works towards utilizing all the resources available to the system, but also helps to avoid the existence of a single point of failure. To this end, we propose in this paper the first distributed algorithm to solve the CSG problem. Our approach is build upon the anytime IP algorithm, which has been shown to be much faster than the IDP algorithms for distributions of coalition values which have been commonly considered in the literature. Furthermore, as demonstrated in the later part of the paper, our approach can be easily extended to the hybrid IDP-IP algorithm. Our main results can be summarized as follows:

1. We adapt the *Distributed Coalition Value Calculation algorithm* (DCVC henceforth) of [2] that is used to calculate the space of coalition values which constitute an input to the CSG problem. With our refinement agents are able to perform the first analysis of the system already during input generation. This is done with little communication and includes establishing worst case bounds from the optimal solution;
2. We propose an effective pre-processing method which reduces the communication overhead between agents. For instance, for uniformly distributed coalition values and 26 agents, on average, only 9000 coalitions out of 67 million (*i.e.* 0.013%) have to be exchanged between agents;
3. We show that this pre-processing method can be used to significantly improve the anytime search process. For normally distributed coalition values the new method is approximately 30 times faster. Crucially, this method can be efficiently used both in distributed and centralized algorithms (like IP or IDP-IP);
4. Distributed numerical simulations for up to 28 agents show that the above developments make our algorithm to take a very good advantage of distributed computing.

[5] Recently, [7] proposed an interesting anytime algorithm, however, this work is still preliminary.

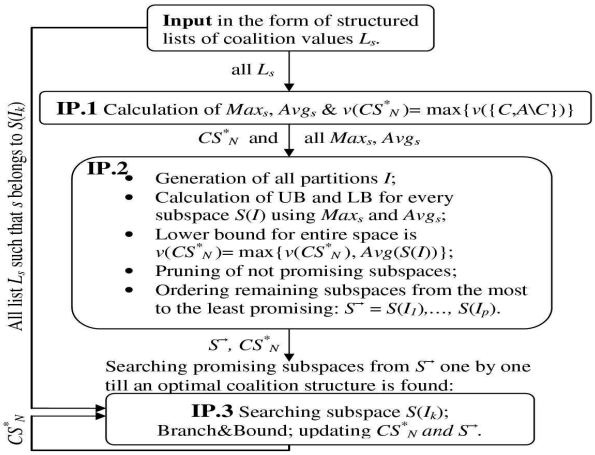

Fig. 1. Scheme of the IP algorithm

In the next section we analyze the IP algorithm. In Section 3 we present our distributed approach and in Section 4 we describe the novel coalition structure search method. Section 5 presents experimental evaluation.

2 Analyzing The IP Algorithm

In our analysis of IP we will pay special attention to the input data needed at every stage of the calculations. We use A to denote the set of agents, n the number of agents, $v(C)$ the value of coalition C and $v(CS)$ the value of coalition structure CS. IP is built upon a novel representation of the search space in which all coalition structures are partitioned into subspaces based on the sizes of the coalitions they contain. Each subspace is related to one integer partition of n denoted by I. For example, given 4 agents, the possible integer partitions (written in the form of transposed vectors) are $I_1 = [4]$, $I_2 = [1, 3]$, $I_3 = [2, 2]$, $I_4 = [1, 1, 2]$, and $I_5 = [1, 1, 1, 1]$. Each integer partition I corresponds to a subspace $S(I)$ containing all coalition structures in which the coalition sizes match the parts of this integer partition. For instance, $S(I_4)$ represents the subspace of all coalition structures containing two coalitions of size 1, and one of size 2. Figure 1 presents the main stages of IP:

[IP.1] Input Analysis

Input: L_s; **Output:** Max_s, Avg_s, $v(CS^*_N) = \max\{v(\{C, A\backslash C\})\}$

The input to the IP algorithm are ordered lists, L_s, of values for coalitions of equal size (see Figure 4 for 6-agent example). IP starts with calculating the maximum and average values of coalitions in every list, denoted Max_s and Avg_s, respectively. Also, all values of coalition structures of form $\{C, A\backslash C\}$ are evaluated. CS^*_N denotes the best coalition structure found so far.

[IP.2] Generation of subspaces and bounds

Input: Max_s, Avg_s, CS^*_N; **Output:** $S(I_1)$, $S(I_2)$, ..., $S(I_z)$,$\max\{v(CS^*_N), LB_I\}$

Information about Max_s and Avg_s is used to calculate both upper and lower bounds on all the subspaces as follows:

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

These bounds are then used to establish worst-case guarantees on the quality of the best solution found so far, and to prune any subspace that cannot contain a better solution. In particular, the lower bound for the entire system is $\max\{v(CS^*_N), LB_I\}$. All the promising subspaces are sorted by their upper bounds in a descending order. Denote the ordered set of them as $\mathcal{S}^{\rightarrow} := \{S(I_1), S(I_2), ..., S(I_z)\}$.

[IP.3] Subspace search

Input: $S(I_1) \in \mathcal{S}^{\rightarrow}$; $L_s : s \in I_1$; **Output:** updated CS^*_N or the optimal solution

Since $S(I_1)$ is characterized by the highest upper bound, this subspace has the greatest potential to contain the optimal coalition structure and, in IP, it is searched first. This stage can be the most computationally expensive since it could ultimately involve searching all subspaces (if none of them could be pruned). The challenge then lies in avoiding the computation of invalid or redundant solutions. For example, the coalition structure $\{\{a_1a_2\}, \{a_1a_3\}\}$ is invalid as agent a_1 appears in two coalitions. An algorithm performs redundant calculations if, for instance, structure $\{\{a_1a_2\}, \{a_3a_4\}\}$ is searched twice, as $\{\{a_1a_2\}, \{a_3a_4\}\}$ and $\{\{a_3a_4\}, \{a_1a_2\}\}$. [9] presents a technique for cycling through only valid and unique coalition structures within any subspace. A sample search for $n = 6$ and subspace $[1, 2, 3]$ is depicted in Figure 2, a). Integers in lists L_1, L_2 and L_3 denote agents belonging to a coalition. Now, for every coalition of size $s = 1$ from list L_1, $\binom{6-1}{2} = 10$ coalitions from list L_2 can be added to form a partial structure. After this, there is only one remaining coalition from list L_3 which completes a coalition structure. In particular, for coalition $\{a_1\}$ from L_1, there are 10 coalitions of size 2 which can be added (any one from $\{a_2a_3\}$ to $\{a_5a_6\}$). For every such two coalitions there a is unique coalition from L_3 that completes the (proper) structure. It is easy to check that there are altogether 60 proper coalition structures in subspace $[1, 2, 3]$. In an attempt to avoid searching all such valid coalition structures, [1] proposed a branch-and-bound technique that disregards some unpromising edges between lists. Its functioning is depicted in Figure 2, b). Let $\{a_1\}$ be the currently considered coalition. If $v(\{a_1\}) + Max_2 + Max_3 \leq v(CS^*_N)$ then searching all coalition structures that start with $\{a_1\}$ can be avoided as it is not possible to add to this coalition any two coalitions of size 2 and 3, respectively, so that the value of a coalition structure improves upon the currently optimal solution CS^*_N. This branch-and-bound technique is used at any moment of constructing a coalition structure. Let $\{a_4a_6\}$ be a coalition added to $\{a_5\}$. If $v(\{a_5\}) + v(\{a_4a_6\}) + Max_3 \leq v(CS^*_N)$ then continuing to construct the coalition structure can be skipped as in L_3 there is no coalition that can improve upon CS^*_N. The process of subspace searching carries on to $S(I_2)$, $S(I_3)$, ... until $v(CS^*_N)$ exceeds the upper bound of the next promising subspace.

3 Distributing the IP Algorithm

We desire our distributed algorithm, called D-IP, to meet the following requirements: all processes should be decentralized; communication overhead should be low; redundant computations should be avoided; computational load should be balanced; and resource

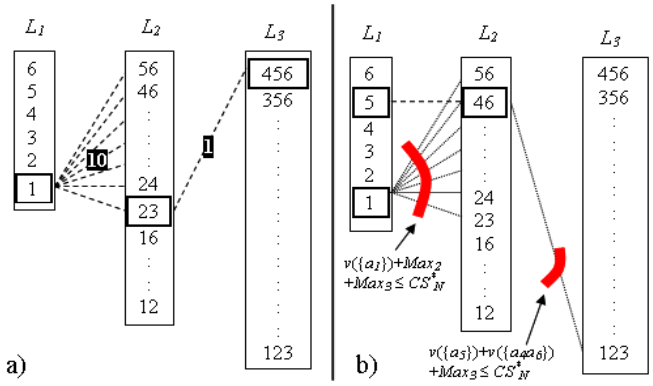

Fig. 2. Searching subspaces and branch and bound in IP

usage should be optimized. The key is to find a good balance between these requirements. Figure 3 shows the scheme of D-IP.

[D-IP.1] Distributed input generation and analysis As the DCVC algorithm is distributed, it constitutes a natural starting point for our distributed approach to the CSG problem. However, we modify DCVC in such a way that it becomes meaningful in the CSG context. In particular, agents can do the analysis phase independently and with little communication.

In the DCVC algorithm, the space of all feasible coalitions is divided into n lists $L_1, ..., L_n$. Each of them contains values that correspond to coalitions of the same size ordered lexicographically as proposed in [2] (see Figure 4). Now, from each list L_s every agent computes a segment of size $\lfloor \frac{|L_s|}{n} \rfloor$ where $|L_s|$ denotes a number of coalitions in the list. The DCVC includes an easily-implementable procedure that deals with left-over coalitions that turn up when L_s is not exactly divisible by n.

In the original DCVC algorithm, segments in every list L_s are distributed by assigning the top segment to agent a_1, the next segment to agent a_2, *etc.* Whereas such a distribution is natural and intuitive, it adds no value in the CSG context. This is because agents are usually able to evaluate only a few coalition structures from segments that are allocated to them. For instance, the first agent always calculates values for coalitions from the top of every list. These coalitions are made of agents with the highest indices. Clearly, a_1 is not able to evaluate even a single proper solution from such shares. With this in mind, we adapt DCVC as follows:

(a) The coalition space is divided in such a way that agents are able to evaluate all coalition structures made of exactly two coalitions, *i.e.* $\{C,\ A\backslash C\}$ (as in centralized IP); and

(b) Agents evaluate all coalition structures made of a coalition from their share and singletons, *i.e.* $\{C, \{j\}_{j\in A\backslash C}\}$.

Regarding (a), for lists of sizes $s = 1, ..., \lfloor \frac{n}{2} \rfloor$ agents are assigned segments of coalitions exactly in the same way as in [2]. However, in lists indexed $s = n - \lfloor \frac{n}{2} \rfloor, ..., n-1$

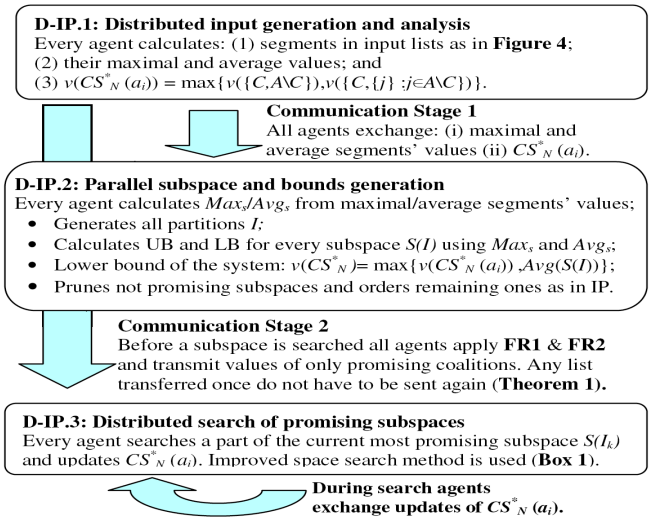

Fig. 3. Scheme of the D-IP algorithm

we assign segments in a reverse order. Due to this change, each agent is able to calculate values of coalition structures made of (i) a coalition of size s and (ii) a coalition of size $n - s$. The example of the distribution for a system of 6 agents is shown in Figure 4. Now, agent a_1 is able to match coalitions of size 1 and 2 with coalitions of size $6 - 1 = 5$ and $6 - 2 = 4$ to create coalition structures $\{\{a_6\}, \{a_1a_2a_3a_4a_5\}\}$, $\{\{a_5a_6\}, \{a_1a_2a_3a_4\}\}$ and $\{\{a_4a_6\}, \{a_1a_2a_3a_5\}\}$. The middle list L_3 is also divided in such a way that agents are able to construct proper coalition structures. For instance, a_1 calculates coalitions $\{a_4a_5a_6\}$, $\{a_3a_5a_6\}$, $\{a_1a_2a_3\}$ and $\{a_1a_2a_4\}$ and evaluates $\{\{a_1a_2a_3\}, \{a_4a_5a_6\}\}$ and $\{\{a_1a_2a_4\}, \{a_3a_5a_6\}\}$.

Regarding (b), agents distribute between themselves values of singletons at the beginning of computations, *i.e.* just after list L_1 has been calculated. Thus, all agents, after calculating a value of coalition C are able to evaluate a coalition structure $\{C, \{j\}_{j\in A\setminus C}\}$. For instance, agent a_1, after calculating $v(\{a_5a_6\})$, evaluates $\{\{a_5a_6\}, \{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}$.

Our distribution has very interesting anytime properties. As shown by [3], evaluation of all coalition structures of types $\{C, A\setminus C\}$, together with the grand coalition and the coalition structure made only of singletons, guarantees that the highest value found is not less than $2/n$ of the optimal solution. Therefore, our adapted DCVC becomes meaningful in the CSG context and, at the same time, preserves all key features of the original algorithm: (i) communication is marginal as the entire distribution process is completely decentralized (only values of singleton coalitions have to be transmitted between agents); (ii) computational load is balanced and virtually no redundant calculations are performed.[6]

[6] Computational load is balanced under the assumption that calculation of coalitions of the same size always takes the same amount of time. The only redundant calculations performed regard

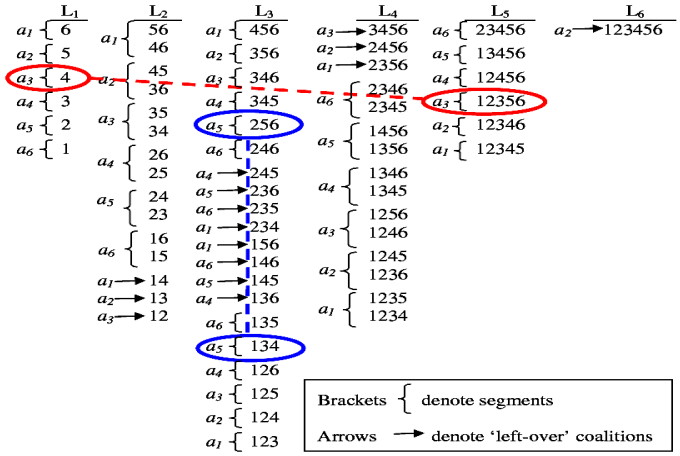

Fig. 4. Adapted DCVC: 6-agent example

Communication Stage 1: To determine promising subspaces, agents calculate UB_I and LB_I, as defined in (1), for every $S(I)$. To do this, they need to know Max_s and Avg_s for all s. But since the calculation of input is distributed, each agent knows only some segments of values within lists. Thus, agents transmit between themselves the following data: (i) the maximum and average value of any segments as well as left-over values they computed; and (ii) the best solution found thus far $CS^*_N(a_i)$.

[D-IP.2] Parallel subspace and bounds generation From data transmitted above, every agent is able to independently compute Max_s and Avg_s of every list and, consequently, UB_I and LB_I for every $S(I)$ as well as the system's lower bound $\max\{v(CS^*_N(a_i)), LB_I\}$.

Communication Stage 2 To search any subspace $S(I_k) \in \mathcal{S}^{\rightarrow}$, agents have to exchange the coalition values in all lists of sizes $s \in I_k$. Since these lists can be extremely long, *i.e.* up to $\binom{n}{\lfloor n/2 \rfloor}$, we apply pre-processing techniques to determine *a priori* which coalitions cannot be in an optimal coalition structure; and hence, do not have to be exchanged with other agents. We apply the following filter rule:

Definition 1 (FR1) *If the value of a coalition C is smaller than the combined value of single agents who belong to C then this coalition is said to be unpromising.*

For instance, if $v(\{a_1a_2\}) < v(\{a_1\}) + v(\{a_2\})$ then, clearly, $\{a_1a_2\}$ cannot be a part of the optimal solution because it is always more efficient to split this coalition. The effectiveness of **FR1** depends on the value of single agents w.r.t. values of coalitions. Although, for both Normal and Uniform distributions of coalition values this rule will filter, on average, 50% of all the coalitions, in the worst case, not even one coalition may be classified as unpromising. Consequently, we propose the second filter rule:

coalition structure made of only singletons. Specifically, every agent calculates this coalition structure by itself.

Definition 2 (FR2) *Coalition $C : |C| \in I_k$ for some subspace $S(I_k)$ is said to be unpromising if*

$$v(C) + UB_{I_k} - Max_{|C|} \leq v(CS^*_N). \quad (2)$$

The logic behind **FR2** in (2) is as follows: instead of using the maximum of a given list $L_{|C|}$ in computing UB_{I_k} we use the exact value of a given coalition C from list $L_{|C|}$. Thus, we obtain the tighter bound on all the coalition structures that belong to $S(I_k)$ and contain coalition C. If this new bound is smaller than CS^*_N then coalition C cannot improve upon the current result, *i.e.* C is unpromising in a given subspace $S(I_k)$. Conceptually, **FR2** is similar to the branch-and-bound technique of [1]. However, there are two important differences. Firstly, **FR2** is applied to all coalitions in a subspace and not only to those encountered during the search process. Secondly, it is applied not during but before the search of a subspace.

All coalitions which satisfy **FR1** or **FR2** do not have to be transferred. Both rules are used to filter lists that belong to subsequently searched subspaces. The key question is whether any coalition that has been rendered unpromising for one subspace can be promising in another subspace. For **FR1** this is, obviously, not possible as this rule is independent of a subspace. But **FR2** depends on the current subspace $S(I_k)$. Thus, at first sight, it seems possible that a coalition rendered unpromising in the context of one subspace might be still promising if it is combined with other types of coalitions from another subspace. However, against this intuition the following theorem holds.

Theorem 1. *Let $S(I_{k<m}) \in \mathcal{S}^{\rightarrow}$ be the currently most promising subspace and let $s \in I_k \cap I_m$. If any coalition C from list $L_{|C|}$ such that $|C| \in I_k$, is unpromising in subspace $S(I_k)$, due to **FR2,** then it will **always** be unpromising in subspace $S(I_{m>k})$ such that $|C| \in I_m$.*

Proof 1 *Let CS^*_N, CS^{**}_N be the current best solutions before searching I_k and $I_{m>k}$, respectively. Clearly, $v(CS^*_N) \leq v(CS^{**}_N)$. Suppose coalition C from list $L_{|C|\in I_k}$ is unpromising in I_k, i.e. $v(C) + UB_{I_k} - Max_{|C|} \leq v(CS^*_N)$. From the definition of $\mathcal{S}^{\rightarrow}$, $UB_{I_m} \leq UB_{I_k}$. Thus, $v(C) + UB_{I_m} - Max_{|C|} \leq v(C) + UB_{I_k} - Max_{|C|} \leq v(CS^*_N) \leq v(CS^{**}_N)$, which gives **FR2** condition (2) for coalition C that holds in subspace I_m.* ■

Theorem 1 implies that after promising coalitions from a list have been transferred, no coalition from this list will have to be transferred again.[7]

D-IP.3 Balancing computational load Considering the search method presented in Figure 2, it is natural to assume that agents allocate work by dividing among themselves the first list of a subspace. For instance, for $n = 6$ let the subspace to search be $S(I_k = [1, 2, 3])$. Referring to segments in Figure 4, agent a_1 would be responsible for calculating all coalition structures that start with coalition $\{a_6\}$, a_2 for coalition structures starting with $\{a_5\}$, *etc*. Since each list is divided into n segments of the same size, then without branch-and-bound technique each agent would have to search exactly the same number of coalition structures. However, the division of this list, as in Figure 4,

[7] Of course, FR2 can be still applied by individual agents to filter additional coalitions from lists that have already been transferred.

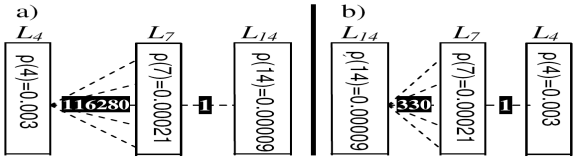

Fig. 5. Different ways to search $\{4, 7, 14\}$.

is no longer reasonable due to the filter rule pre-processing. In particular, we cannot assume that the same number of coalitions will be filtered out from all agents' segments. Thus, we calculate the segments again, however, now only for the promising elements in the first list of every subspace.[8] When the search progresses further into the subspace, branch and bound is going to prune away unpromising search directions (see Figure 2 b), possibly more for some agents than for others. Then, the search process becomes irregular and another, dynamic load balancing policy has to be applied. We propose the following easily-implementable technique that turns out to be comparatively effective in empirical evaluation.

Computational load balancing protocol *An agent that has just finished its calculations sends a signal to another agent with a randomly chosen index. If the second agent is still in the process of calculation then it answers with a partial coalition structure that shows the current progress of search in his assignment. Then, this assignment is split between both agents into two (nearly equal) parts.*

4 Improved Space-Search Method

In IP coalition structures are constructed using coalitions of ascending size. However, this process can be also performed with an another order. For instance, in Figure 2 I can be defined as $[3, 2, 1]$ or $[2, 3, 1]$, not $[1, 2, 3]$. If branch and bound (and filtering) is not used, then the order of the lists in a subspace does not matter, since all the correct combinations of the coalitions into structures are considered in every case, just in different order. Yet, if branch and bound is active, the earlier it stops a construction of a structure the better because more work is saved. This is demonstrated in Figure 2 b). Branch and bound applied in the first list prunes away 10 search direction, whereas branch and bound applied in the second list prunes only single one. Thus, more profitable are these orders in which branch and bound is more effective for early lists, *i.e.*, in which it dismisses higher percent of values from early lists.

Now, **FR2** not only reduces communication load but it can be also used to predict the effectiveness of the branch and bound. This prediction is more accurate for first lists in a given subspace and less accurate for the last lists. In fact, if the first list of a subspace is concerned, **FR2** and branch and bound dismiss exactly the same coalitions. In other words, this prediction is perfect. Yet, for the second list all the coalitions filtered out by **FR2** would be also dismissed by branch and bound but branch and bound may be

[8] The number of not filtered coalitions in the first list is divided by the number of agents. Left-over coalitions are then distributed in the same way as in DCVC.

more effective and dismiss some further coalitions. This is because from the first list it considers a concrete value and not the maximum as **FR2** does (compare Figure 2 b) to formula (2)). Although, following the same reasoning, for the third and later lists such a prediction is even less accurate, we neglect these differences as being marginal. What we keep in mind is that **FR2** predicts a higher bound on the effectiveness of branch and bound in a given list. As the order of a subspace we choose such that the percentage of unpromising coalitions is the highest for the first list, the second highest for the second list and so on. This ensures that branch and bound is nearly the most effective as it prunes away the biggest parts of the subspace at very early stages. We call this an *improved order* of constructing coalition structures in a given subspace.

The following example demonstrates that the gains from the improved order can be substantial. Figure 5 is based on simulations results for $n = 25$ and coalition values drawn from the normal distribution. Both rules determined that the proportions of promising coalitions in L_4, L_7, and L_{14}, are $\rho(4) = 0.05$, $\rho(7) = 0.00021$ and $\rho(14) = 0.00009$, respectively. Order $[4, 7, 14]$ yields the expected number of about 73 millions edges (Figure 5 a) between L_4 and L_7. This number is obtained by multiplying the number of promising coalitions in L_4, *i.e.* $|L_4|\rho(4)$, by the number of edges between one coalition in L_4 and L_7. At the same time, order $[14, 7, 4]$ yields only about 140 thousands edges (Figure 5 b)! Clearly, it is much more efficient to start search from the promising coalitions in L_{14} than from these in L_4.

5 Experimental Evaluation

In this section, we evaluate the performance of the D-IP algorithm that we implemented using JADE (Java Agent DEvelopment Framework).[9] In this framework, the information, the initiative and the resources can be fully distributed between mobile or fixed terminals. Specifically, 14 Intel Core 2 Duo, 2.0 GHz workstations connected by a 1Gb Ethernet network were used to simulate a maximum of 28 agents (each agent runs on one core).

Following, among others [3], [1], we consider two probability distributions of coalition values:

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

We focus on the main contributions of this paper. The second/fifth column in Figure 6 presents the effectiveness of filter rules as a percentage of coalition input exchanged by agents. To show the effectiveness of our distributed approach we compare D-IP (without an improved order) to the potential ideal distributed IP algorithm, *i.e.* to the situation when IP would be distributed without cost. In other words, in the third/sixth column we compare running time of D-IP (without an improved order) to running time of IP divided by n. Finally, let us denote by IP+ the IP algorithm with added improved search space method. The forth/seventh column shows the relative performance of IP+ w.r.t. IP.

[9] See http://jade.tilab.com.

	Normal Distribution			Uniform Distribution		
n	**Transfers**	**D-IP vs. IP/n**	**IP+ vs. IP**	**Transfers**	**D-IP vs. IP/n**	**IP+ vs. IP**
18	2.72 ± 0.76%	3.15 ±0.60	25 ±08	0.210 ± 0.061%	2.43 ±0.31	1.030±0.03
20	2.42 ± 0.77%	3.06 ±0.67	32 ±12	0.132 ± 0.036%	2.47 ±0.25	1.028±0.03
22	1.98 ± 0.24%	3.03 ±0.55	36 ±12	0.074 ± 0.021%	2.44 ±0.34	1.032±0.04
24	1.55 ± 0.23%	3.04 ±0.62	34 ±10	0.017 ± 0.005%	2.43 ±0.26	1.027±0.03
26	0.98 ± 0.17%	3.21 ±0.58	35 ±06	0.013 ± 0.004%	2.30 ±0.29	1.029±0.03
28	0.82 ± 0.17%	3.11 ±0.55	32 ±07	0.011 ± 0.004%	2.48 ±0.32	1.034±0.04

Fig. 6. Results of numerical simulations

As can be seen, the filter rules significantly reduce the number of coalitions to be transferred. For the normal distribution this is, on average, less than 1% for $n = 26$. For the uniform distribution these results are even better. For instance, for $n = 26$, from 67 millions of coalitions that would have to be transferred without filter rules only about 9000 are usually determined to be promising! The improved search space method is the most effective for the normal distribution, when it speeds up IP by a factor of 35. However, it does not offer much of an improvement for the uniform distribution. This is because, due to the nature of this distribution, **FR1** and **FR2** remove nearly the same proportion of coalitions from every list.[10] Consequently, the original ascending order in IP is nearly optimal. In fact, all orders deliver similar results for the uniform distribution.

Our distributed approach, due to significant reductions in communication load and computational load balancing policy, is, for the normal distribution, slower than the ideal distributed IP only by a factor of 3.1. This happens when we do not take into account the improved space search method. However, if we run the full version of D-IP, then, for the normal distribution, our algorithm is about 12 times faster than the ideal distributed IP! Furthermore, for the uniform distribution, the performance of D-IP is slower than the ideal only by a factor of 2.5. This may be attributed to much lower communication load. On the other hand, as observed earlier, the improved space search method cannot enhance the result for this distribution.

6 Conclusions

In this paper we proposed a novel, distributed approach to coalition structure generation. We build upon both the DCVC algorithm in [2] and the anytime centralized IP algorithm in [1]. Numerical simulations show that our algorithm significantly limits transfer load between agents and efficiently takes advantage of the distributed computing. Importantly, some our ideas can be successfully applied to the centralised version of the algorithm and may significantly improve its performance.

The sketch of proof of correctness of IP can be found in [1]. Two issues should be additionally considered w.r.t. our distribution: filter rules and balancing of computational load. Since filter rules remove only such coalitions that cannot be a part of an optimal solution they do not affect the outcome of the search process. The same

[10] See the discussion in the previous section.

concerns balancing of computational load. As the procedure of splitting the search assignment follows the way in which IP constructs coalition structures, no solution to be checked is omitted.

Our distributed approach can also be extended to the IDP-IP algorithm. In short, following dynamic programming principle, this algorithm checks for all coalitions of size $m < n$, what are their most profitable splits into smaller coalitions. This allows us to avoid searching many subspaces that contain not optimal splits as the optimal ones are already known (see [8] for more details). Remaining subspaces are searched in the same way as in IP. In general, the bigger m the fewer subspaces must be searched. The challenge to distribute IDP-IP lays in performing the initial splits of coalitions from L_m. To do this, agents need to know values of all coalitions in lists $L_{j<m}$ which have to be transmitted after the input is generated. Thus, in the distributed version of IDP-IP there is a trade-off between transmission of initial lists of coalitions and number of subspaces to search later on. Interestingly **FR1** can be still applied to reduce number of transmitted coalitions. Importantly, both the filter rules as well as the improved procedure of searching subspaces can be used in the search phase of the IDP-IP algorithm as it exactly corresponds to the search in IP.

References

[1] Rahwan, T., Ramchurn, S.D., Giovannucci, A., Dang, V.D., Jennings, N.R.: Anytime optimal coalition structure generation. In: AAAI-07. (2007) 1184–1190

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

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

[4] Rahwan, T., Jennings, N.R.: An improved dynamic programming algorithm for coalition structure generation. In: AAMAS-08. (2008)

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

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

[7] Sombattheera, C., Ghose, A.: A best-first anytime algorithm for computing optimal coalition structures. In: AAMAS '08. (2008)

[8] Rahwan, T., Jennings, N.R.: Optimal coalition structure generation: Anytime optimization meets dynamic progamming. In: AAAI-08. (2008)

[9] Rahwan, T., Ramchurn, S.D., Dang, V.D., Jennings, N.R.: Near-optimal anytime coalition structure generation. In: IJCAI-07. (2007) 2365–2371