Distributing Coalitional Value Calculations among Cooperative Agents

Talal Rahwan and Nicholas R. Jennings
University of Southampton. Southampton SO17 1BJ, UK.
{tr03r, nrj}@ecs.soton.ac.uk

ABSTRACT

The process of forming coalitions of software agents generally requires calculating a value for every possible coalition which indicates how beneficial that coalition would be if it was formed. Now, since the number of possible coalitions increases exponentially with the number of agents involved, having one agent calculate all the values is inefficient. Given this, we present a novel algorithm for distributing this calculation among agents in cooperative environments. Specifically, by using our algorithm, each agent is assigned some part of the calculation such that the agents' shares are exhaustive and disjoint. Moreover, the algorithm is decentralized, requires no communication between the agents, and has minimal memory requirements. To evaluate the effectiveness of our algorithm we compare it with the only other algorithm available in the literature (due to Shehory and Kraus). This shows that for the case of 25 agents, the distribution process of our algorithm took 0.00037% of the time, the values were calculated using 0.000006% of the memory, the calculation redundancy was reduced from 477826101 to 0, and the total number of bytes sent between the agents dropped from 674047872 to 0 (note that for larger numbers of agents, these improvements become exponentially better).

Introduction

Coalition formation, the process by which a group of software agents come together and agree to coordinate and cooperate in the performance of a specific task, is an important form of interaction in multi-agent systems. Such coalitions can improve the performance of the individual agents and/or the system as a whole. Now, if we view the population of agents as a set A, then every subset of A is a potential coalition (meaning that the total number of these subsets is $2^{|A|}$ - 1). Given this, a number of coalition formation algorithms have been developed to determine which of the potential coalitions should actually be formed. To do so, they typically calculate a value for each coalition, known as the *coalition value*, which provides an indication of the expected outcome that could be derived if that coalition was formed. Then, having computed all the coalitional values, the decision about the optimal coalition to form can be taken. The problem here, however, is that computing the coalitional values is exponentially complex due to the number of possible coalitions which must be considered. To help combat this computational explosion, some coalition formation algorithms only search a sub-set of the potential set of coalitions (see related work section for details). In either case, however, it is desirable to distribute the calculations of these coalitional values among the agents, rather than having it done centrally by one agent (as is the case in most extant work). In this way, the search can be done faster and the agents can share the burden of the calculations. To this end, there are a number of desiderata that we can place on such a distribution algorithm:

1. The distribution process should be decentralized. That is, no one decision maker should be required to decide which agent calculates which values, otherwise the system would have a performance bottleneck and a single point of failure.
2. Communication between the agents should be minimized.
3. The coalitional value of all the desired coalitions should be computed and the agents should minimize the number of calculations that are redundantly carried out.
4. Each agent should compute an equal number of values.
5. The amount of memory each agent requires to perform the computations should be minimized [1].

Against these requirements, we present a novel algorithm (called **DCVC**) for Distributing Coalitional Value Calculations among the constituent agents. Here, we assume that the agents are cooperative (i.e. they carry out their share of the computations and they report the results truthfully)[2].

In more detail, DCVC ensures each agent is assigned some part of the calculations such that the agents' shares are exhaustive and disjoint. Moreover, the algorithm is decentralized, requires no communication between the agents, and distributes the calculations equally. We also show how, using DCVC, the agents can calculate all the coalitional values by saving no more than one coalition each. To benchmark the effectiveness of our algorithm we compare it with the only other algorithm available in the literature (Shehory & Kraus 1998). In so doing, we show that for the case of 25 agents, the distribution process of our algorithm took 0.00037% of the time, the values were calculated using 0.000006% of the memory, the calculation redundancy was reduced from 477826101 to 0, and the total number of bytes sent between the agents dropped from 674047872 to 0. Note that for larger numbers of agents, these improvements become exponentially better.

[1] Since the number of possible coalitions is exponentially large, any algorithm that requires each agent to save all the possible coalitions in its share will require infeasibly large amounts of memory (e.g. saving a list of all the possible coalitions of 40 agents requires a total of 5120 GB of memory).

[2] However, the underlying algorithm can also be applied in environments where the agents are selfish (i.e. they act to increase their own outcome and may lie about the results they find if it is beneficial to do so). This can be achieved using an additional enforcement mechanism by which the agents are incentivized to calculate all the values they are assigned and to announce the true results they find. The exact nature of this mechanism is left for future work at this stage.

Related Work

Coalition formation has received a considerable amount of attention in recent research, and has proven to be useful in a number of real-world scenarios and multi-agent systems. In e-commerce, for example, buyers can form coalitions to purchase a product in bulk and take advantage of price discounts (Tsvetovat *et al.* 2000). In e-business and grid contexts, virtual organisations of agents can be formed in order to satisfy particular market niches (Norman *et al.* 2004).

In general, finding the optimal coalition(s) requires searching the whole set of possible coalitions, which is exponential in the number of agents. To tackle this problem, some researchers have proposed algorithms that only search a subset and produce solutions that are guaranteed to be within a finite bound of the optimal. Specifically, Sandholm et al. (1999) proved that a worst-case bound can be established by searching the bottom two levels of the *coalition structure graph*. They also presented an anytime algorithm which can meet tighter bounds by searching the rest of the graph as long as there is time left, starting from the top level downwards. However, their algorithm's computational complexity is exponential. Moreover, they search through coalition structures which, by definition, include *disjoint* coalitions where each agent is a member of only one coalition. This means that they exclude the possibility of having overlapping coalitions. Dang and Jennings (2004) presented an alternative way for searching the coalition structure graph; they first search the bottom two levels, as well as the top one. After that, however, instead of searching the remaining levels one by one, they search specific subsets of all remaining levels. They also proved that their algorithm can establish the same bounds from the optimal by searching a significantly smaller space than Sandholm et al.'s. However, the complexity of their algorithm remains exponential, and they also do not consider the case of overlapping coalitions. Shehory and Kraus (1998) set limitations on the size of the permitted coalitions which makes the formation process of polynomial complexity. They also consider environments where the coalitions are allowed to overlap. In more detail, their solutions are bounded by a logarithmic ratio bound from the optimal solution *given the limit on the coalitional size*. However, no bound can be guaranteed from the optimal solution that could have been found by searching all possible coalitions.

In either the optimal case (in which the coalitional values of all the coalitions are calculated) or the sub-optimal case (in which only a subset of the values are calculated) the issue of who performs which of these calculations is still a key concern. In Sandholm et al.'s work, a method is presented for choosing which agent searches which portion of the space. Specifically, their method assigns each agent the same *expected* amount of search. However, this still leaves some agents searching significantly more space than others. The algorithm presented by Dang and Jennings was tested centrally, and there was no description of how the search can be done in a distributed manner among the agents. Shehory and Kraus do present an algorithm for distributing the value calculations among the agents. Their method works by making the agents negotiate about which of them performs which of the calculations (see performance evaluation section for more details). However, by using their algorithm, some agents may calculate significantly more values than others, and some values can be calculated more than once. In addition, their algorithm requires high communication complexity.

To address these shortcomings, we developed an algorithm that distributes the coalitional value calculations efficiently among the agents. Like Shehory and Kraus's algorithm, ours can be applied for environments where the coalitions are allowed to overlap, and where the algorithm imposes specific limitations on the coalitional sizes (cf. Sandholm's and Dang's algorithms). Therefore, we compare our algorithm with Shehory and Kraus's algorithm (hereforth called SK).

Distributing Coalitional Value Calculations

In general, the set of possible coalitions can be divided into subsets, each containing the coalitions of a particular size. In DCVC, the distribution of all possible coalitions is carried out by distributing each of these subsets equally among the agents (i.e. agent a_1 has x coalitions of size 1 to consider, y of size 2, z of size 3, and so on, and so does a_2, a_3, and so on). This has the following advantages:

- An increase in the size of the coalition usually corresponds to an increase in the number of operations required to calculate its value. Therefore, by distributing the coalitions of every size equally among the agents, each agent will not only calculate the same number of values, but also perform the same number of operations.
- Any relevant limitations can be placed on the size of the coalitions that are allowed to form. For example, if coalitions of a particular size are not allowed to form, then the agents simply do not distribute the coalitions of this size among themselves. In such cases, the agents would still calculate the same number of values. Note that allowing for such limitations is important since the problem under investigation might only allow the formation of coalitions of particular sizes. This is also important since it makes DCVC applicable for coalition formation algorithms that reduce the complexity of the search by limiting the size of the coalitions.

Now, let A be the set of agents, and n be the number of agents (i.e. $n = |A|$). In order to allow for any limitations on the coalitional sizes, we assume there is a set S of the permitted coalitional sizes. Let L_s be the list of possible coalitions of size $s \in S$, and N_s be the number of coalitions in L_s ($i.e.\ N_s = |L_s|$). Also, let $\{i, ..., j\}$ denote the coalition of agents $a_i, ..., a_j$. Now for any $s \in S$, L_s should be ordered as follows:

- The first coalition in the list is: $n - s + 1, ..., n - 1, n$.
- The last coalition in the list is: $1, ..., s - 1, s$.
- Given any coalition c_i that is located at index i in L_s, the agent can find c_{i-1} by checking the values $c_{i,s}, c_{i,s-1}, c_{i,s-2}, ...$ until it finds a value $c_{i,x}$ such that $c_{i,x} < c_{1,x}$, then:
  - $c_{i-1,k} = c_{i,k} : 1 \leq k < x$
  - $c_{i-1,k} = c_{i,k} + 1 : k = x$
  - $c_{i-1,k} = c_{i-1,k-1} + 1 : x < k \leq s$

This means the agents know how L_s is ordered, although they do not actually maintain L_s. An example of the resulting lists is

shown in Table 1. Here we have $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$, $n = 6$, $S = \{1, 2, 3, 4, 5, 6\}$ and $N_1, N_2, N_3, N_4, N_5, N_6$ have the values $6, 15, 20, 15, 6, 1$ respectively.

L_1	L_2	L_3	L_4	L_5	L_6
6	5, 6	4, 5, 6	3, 4, 5, 6	2, 3, 4, 5, 6	1, 2, 3, 4, 5, 6
5	4, 6	3, 5, 6	2, 4, 5, 6	1, 3, 4, 5, 6	
4	4, 5	3, 4, 6	2, 3, 5, 6	1, 2, 4, 5, 6	
3	3, 6	3, 4, 5	2, 3, 4, 6	1, 2, 3, 5, 6	
2	3, 5	2, 5, 6	2, 3, 4, 5	1, 2, 3, 4, 6	
1	3, 4	2, 4, 6	1, 4, 5, 6	1, 2, 3, 4, 5	
	2, 6	2, 4, 5	1, 3, 5, 6		
	2, 5	2, 3, 6	1, 3, 4, 6		
	2, 4	2, 3, 5	1, 3, 4, 5		
	2, 3	2, 3, 4	1, 2, 5, 6		
	1, 6	1, 5, 6	1, 2, 4, 6		
	1, 5	1, 4, 6	1, 2, 4, 5		
	1, 4	1, 4, 5	1, 2, 3, 6		
	1, 3	1, 3, 6	1, 2, 3, 5		
	1, 2	1, 3, 5	1, 2, 3, 4		
		1, 3, 4			
		1, 2, 6			
		1, 2, 5			
		1, 2, 4			
		1, 2, 3			

Table 1: The lists of possible coalitions for 6 agents.

Now, for each agent $a_i \in A$, let $L_{s,i}$ be its share of L_s (i.e. the subset of L_s for which it will calculate values) and $N_{s,i}$ be the number of coalitions in $L_{s,i}$ (i.e. $N_{s,i} = |L_{s,i}|$). Given this, we can now express our distribution algorithm (see Figure 1). Here we assume each agent has a unique global identifier (UID) by which it is identified by other agents[3].

Each agent a_i should perform the following:
- Sort the set of agents based on the agents' UID
- For every $s \in S$, do the following:

1. If $\mathbf{s = 1}$:
Calculate the value of the coalition $\{i\}$

2. If $\mathbf{1 < s < n - 1}$:

2.1. Calculate the size of your share: $N_{s,i} = \lfloor N_s/n \rfloor$

2.2. Calculate the index of the last coalition in your share: $index_{s,i} = i * N_{s,i}$

2.3. Calculate the values of each coalition in your share.

2.4. Calculate the number of additional values that need calculation: $N' = N_s - n * \lfloor N_s/n \rfloor$

2.5. If $(N' > 0)$ then:
find the sequence of agents A' in which each agent should calculate one additional value. And if you are a member of A' then calculate the appropriate value. This is done as follows :
- If $(\alpha + N' - 1 \leq n)$ then: $A' = (a_\alpha, a_{\alpha+1}, ..., a_{\alpha+N'-1})$ else: $A' = (a_\alpha, a_{\alpha+1}, ..., a_n, a_1, ..., a_{(\alpha+N'-1)-n})$.
- If $(a_i \in A')$ then calculate one of the additional values based on your position in A'.

If $(\alpha + N' \leq n)$ then: $\alpha = \alpha + N'$ else: $\alpha = \alpha + N' - n$

3. If $\mathbf{s = n - 1}$:
Calculate the value for the coalition $\{1, ..., i-1, i+1, ..., n\}$

4. If $\mathbf{s = n}$:

4.1. If $(\alpha = i)$ then calculate the value of the coalition $\{1, ..., n\}$

4.2. Set: $\alpha = \alpha + 1$

Figure 1: The DCVC algorithm.

In more detail, each agent starts by sorting the set of agents according to their UID. Note that this is done using a unique key, which means that each agent will end up with the same sequence, denoted by $\vec{A}$. Moreover, the agents implicitly agree on $\vec{A}$ without contacting each other; this is because every agent knows that every other agent also has $\vec{A}$. Note that sorting the set of agents is only performed once. For the remainder of this paper, we will denote by a_i the agent located at position i of the resulting sequence $\vec{A}$. Now by having an agreement on $\vec{A}$, each agent can know which of the calculations it should perform based on its position in $\vec{A}$, this is done as follows:

1. For $s = 1$, there exists n possible coalitions. Therefore, the calculations are distributed such that each agent calculates one value. This is done by having each agent a_i calculate the value of the coalition in which it is the only member (i.e. $\{i\}$).

2. For any $1 < s < n - 1$, each agent a_i starts by calculating the number of coalitions in $L_{s,i}$ as follows:

$$N_{s,i} = \lfloor N_s/n \rfloor$$

The agent then calculates the index in L_s at which $L_{s,i}$ ends (denoted by $index_{s,i}$). This is done as follows:

$$index_{s,i} = i * N_{s,i}$$

The agent now calculates the values of all the coalitions in $L_{s,i}$. This is done without maintaining L_s, or even maintaining $L_{s,i}$. Instead, the agent calculates the values by saving one coalition at a time; this is done by allocating a space of memory, denoted by M, which is enough to save the maximum size coalition. Basically, the agent sets M to be the last coalition in $L_{s,i}$, and after calculating its value, the agent sets M to be the coalition before it. It then calculates its value, and so on until all the values in $L_{s,i}$ are calculated. As mentioned earlier, given a coalition in L_s, the agent can always find the coalition before it. Then, in order to know the coalitions in $L_{s,i}$, it is enough for the agent to know the last coalition in $L_{s,i}$. Note that the agent so far knows only the index in L_s at which $L_{s,i}$ ends. However, since the agent does not maintain L_s, then knowing the index does not give the coalition directly. Therefore, the agent needs to be able to find the coalition by only knowing its index in L_s.

Generally, the number of all possible coalitions of size s (i.e. the coalitions that contain s agents) out of n agents, is given by the following equation:

$$|n, s| = \frac{n!}{(n-s)! * s!} \tag{1}$$

Now let $P(i, \{i+1, ..., n\})$ be the list of all possible coalitions of agents $a_{i+1}, ..., a_n$ after adding a_i in the beginning of each coalition. Also, let $P_s(i, \{i+1, ..., n\})$ be the list of all coalitions in $P(i, \{i+1, ..., n\})$ that are of size s. From (1) we find that the number of coalitions in $P_s(i, \{i+1, ..., n\})$ is given as follows:

$$|P_s(i, \{i+1, ..., n\})| = |n - i, s - 1| \tag{2}$$

Now for every $1 < s < n-1$, if L_s was ordered as specified earlier, then L_s will contain $P_s(i, \{i+1, ..., n\})$ with i running from $n - s + 1$ down to 1. For example, for 6 agents, L_4 will contain $P_4(3, \{4, 5, 6\})$, then $P_4(2, \{3, 4, 5, 6\})$ and finally $P_4(1, \{2, 3, 4, 5, 6\})$ (see Table 1). Therefore, any coalition in L_s which starts with $(n - s + 1) - i + 1$ must

[3]The existence of such an identifier is a reasonable assumption since all agents need to be uniquely identifiable so that messages can be routed corrrectly.

have an index k such that:

$$k > \sum_{j=1}^{i-1} |P_s((n-s+1)-j+1, \{(n-s+1)-j+2, ..., n\})|$$

$$k \leq \sum_{j=1}^{i} |P_s((n-s+1)-j+1, \{(n-s+1)-j+2, ..., n\})|$$

For example, for 6 agents, any coalition in L_4 that starts with 1 must have an index k such that:

$$k > |P_s(3, \{4,5,6\})| + |P_s(2, \{3,4,5,6\})| = 1 + 4 = 5$$

$$k \leq |P_s(3, \{4,5,6\})| + |P_s(2, \{3,4,5,6\})| + |P_s(1, \{2,3,4,5,6\})|$$

From (2) we know that any coalition in L_s which starts with $(n-s+1)-i+1$ must have an index k such that:

$$k > \sum_{j=1}^{i} |s+j-2, s-1| \ , \quad k \leq \sum_{j=1}^{i+1} |s+j-2, s-1|$$

Based on this, agent a_i can set M to be the coalition located at $index_{s,i}$ without maintaining L_s as follows. Each agent first forms what we call a *Pascal array* which is of size: $n-1 * n-1$. The array includes values from Pascal's triangle[4] and is calculated as follows:

$Pascal[i,1] = 1 : \forall i \in \{1, ..., n-1\}$
$Pascal[1,j] = j : \forall j \in \{2, ..., n-1\}$
$Pascal[i,j] = Pascal[i-1,j] + Pascal[i,j-1] : \forall i,j \in \{2, ..., n-1\}$

By this, the following equation holds:

$$Pascal[s,i] = \sum_{j=1}^{i} |s+j-2, s-1|$$

Therefore, the agent can find the first member in the required coalition by checking the values: $Pascal[s,1], Pascal[s,2], ...$ until it finds a value $Pascal[s,x]$ such that $Pascal[s,x] \geq index_{s,i}$. The first member would then be $(n-s+1)-x+1$. (Step 1 in Figure 2 shows how to find the first member in a coalition that is located at index 46 in the list L_5 for 9 agents).

Now since the first member is $(n-s+1)-x+1$, then the rest of the members must be located in the sub-list which contains all the coalitions that start with $(n-s+1)-x+1$ after removing the first member. This sub-list is similar to L_{s-1}. However, it contains $P_s(i, \{i+1, ..., n\})$ with i running from $n-s+2$ down to $(n-s+2)-x+1$ instead of 1 (see the list in Figure 2, step 2). Note that in this sub-list, the index of the required coalition becomes $index_{s,i} - Pascal[s, x-1]$ (in our example, the index of the required coalition becomes: $46-21=25$). Based on this, the agent can find the next member in the coalition by checking the values: $Pascal[s-1,1], Pascal[s-1,2], ...$ until it finds a value $Pascal[s-1,x] \geq index_{s,i} - Pascal[s,x-1]$, the next member would then be $(n-(s-1)+1)-x+1$.

Similarly, all the members of the coalition can be found. Note that as the agent checks the values in *Pascal array* in order to find some member M_j, if it finds a value that is equal to the required index, then the agent can find M_j, as well as all the members after it as follows: $M_{k+1} = M_k + 1 : k = j, ..., s-1$. Figure 2 shows a complete example for setting M to be the coalition at $index = 46$ in the list L_5 for 9 agents.

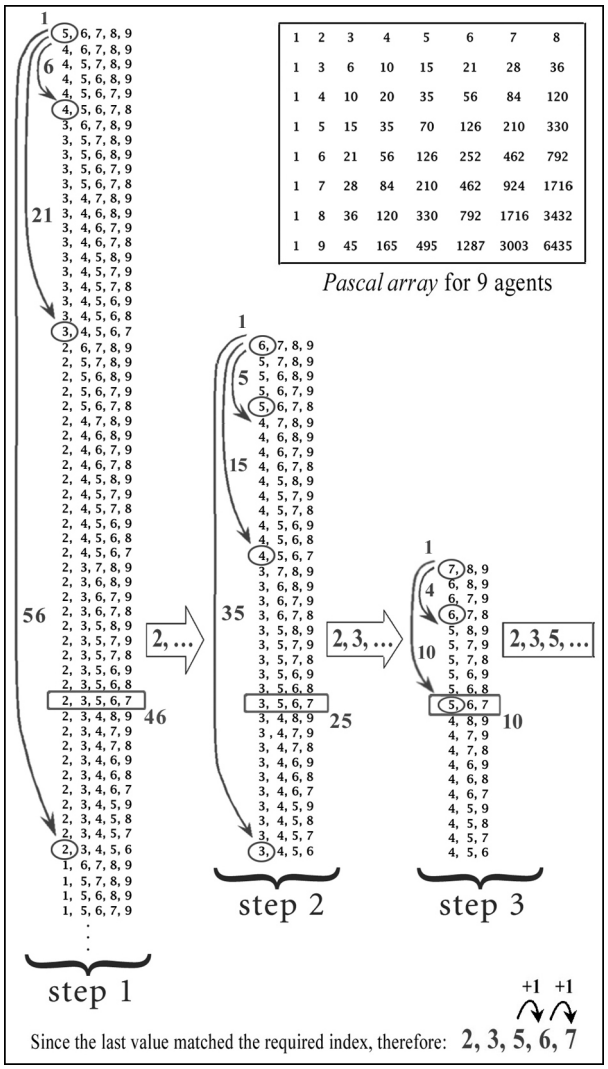

Figure 2: Finding a coalition at $index = 46$ in the list L_5 of coalitions of 9 agents.

Now that each agent a_i has set M to be the last coalition in $L_{s,i}$, it repeatedly performs the following:

- Calculate the value of M[5]
- Set M to be the coalition before it. This is done by first checking the values $M_s, M_{s-1}, M_{s-2}, ...$until it finds a

[4]More details about Pascal triangles can be found in (Conway & Guy 1996).

[5]The details of how to calculate a value are left for the developers to decide; this is because the evaluation of different metrics (e.g. time, quality, cost etc.) can differ based on the problem under investigation.

value M_β such that $M_\beta < c_{1,\beta}$, then:

- $M_k = M_k + 1 : k = \beta$
- $M_k = M_{k-1} + 1 : \beta < k \leq s$

This process should be repeated until all the coalitional values in $L_{s,i}$ are calculated. Note that after each agent calculates the values in its share, some values might remain uncalculated. This is because N_s might not be exactly divisible by the number of agents, and in this case, the agents' equal shares will not cover all the required values. In particular, the number of the remaining values would be:

$$N' = N_s - \sum_{j=1}^{n} N_{s,j} = N_s - n * \lfloor N_s/n \rfloor \qquad (3)$$

And the coalitions that need their values to be calculated would be: $c_{N_s - N' + i} : i \in 1, ..., N'$. Note that $N' < n$, and that each agent so far has calculated the same number of values. Therefore, in order to calculate these additional values and keep the distribution as fair as possible, each value should be calculated by a different agent; the agents should agree on a sequence A' which contains N' agents and in which each agent calculates one additional value. This can be done by maintaining a value α, initially set to 1, then for any list L_s, if there are additional values (i.e. if $N' > 0$) then A' would contain N' agents, starting from a_α. Then, each agent in A' calculates one additional value based on its position in A' (if we denote by a'_i the agent located at index i of A', then a'_i should calculate the value of coalition $c_{N_s - N' + i}$). Note that after these values are calculated, the agents need to update α so that for other lists, the next N' agents perform any additional calculations. This way, given any set S, the total number of values calculated by each agent will either be equal, or differ by only one value. Updating α is done as follows:

$$If(\alpha + N' < n)\ then\ \alpha = \alpha + N',\ else\ \alpha = \alpha + N' - n$$

And forming A' such that it contains N' agents, starting from a_α, is done as follows:

$$If(\alpha + N' - 1 < n)\ then\ A' = (a_\alpha, a_{\alpha+1}, ..., a_{\alpha+N'-1})$$
$$else\ A' = (a_\alpha, a_{\alpha+1}, ..., a_n, a_1, ..., a_{\alpha+N'-n})$$

For example if we have 6 agents, then from equation (3) we find that for L_2 we have $N' = 3$. Therefore, A' would be: (a_1, a_2, a_3) and α becomes 4. Then for L_3 we have $N' = 2$. Therefore, A' would be (a_4, a_5) and α becomes 6. Finally for L_4 we have $N' = 3$. Therefore, A' would be (a_6, a_1, a_2) and α becomes 3.

3. For $s = n - 1$, there exists n possible coalitions. Therefore, the calculations are distributed such that each agent calculates one value. This is done by having each agent a_i calculate the value of the coalition in which it is not a member, and every other agent is a member (i.e. $\{1, ..., i-1, i+1, ..., n\}$).

4. For $s = n$, there exists one coalition: $\{1, ..., n\}$. This is similar to the case where $N' = 1$. Therefore, the value of this coalition is calculated by a_α. In our example of 6 agents, this value would be calculated by a_3 and α becomes 4. Note that after all the values are calculated, the value of α remains 4 instead of being initialized to 1. This means that in order to form other coalitions, any additional calculations will start from a_4. By this, the average number of values calculated by each agent becomes equal.

Performance Evaluation

To evaluate the performance of the DCVC algorithm we compare it against the SK algorithm (see Figure 3).

Each agent a_i should perform the following:

- Put in P_i the set of potential coalitions that include up to k agents including a_i.
- While P_i is not empty do:
  - Contact an agent a_j that is a member of a potential coalition in P_i.
  - Commit to the calculation of the values of a subset S_{ij} of the common potential coalitions (i.e. a subset of the coalitions in P_i in which a_i and a_j are members).
  - Subtract S_{ij} from P_i. Add S_{ij} to your long-term commitment list.
  - For each agent a_k that has contacted you, subtract from P_i the set S_{ki} of the potential coalitions for which it had committed to calculate values.
  - Calculate the values for the coalitions you have committed to (Sij).
  - Repeat contacting other agents until $P_i = a_i$ (i.e., no more agents to contact).

Figure 3: The SK algorithm.

Specifically, we tested the performance of DCVC and SK for different numbers of agents[6]. The results presented in Table 2 are for the case where coalitions of any size are allowed to form (which means in our terms $S = \{1, ..., n\}$, and in SK's terms: $k = n$). Note that the results for SK were taken as an average of running a number of times; this is because their algorithm gives different results based on the order by which the agents contact each other.

The results show the differences in the performance of both algorithms in terms of:

1. Distribution time: The agents performed significantly faster when using DCVC. This is because in DCVC each agent can start processing its share of coalitions immediately, while in SK, each agent had to start with a list of all the coalitions in which it is a member, and then repeat the process of negotiating with other agents and committing to some coalitions and deleting others, until there were no more agents to contact.

2. Redundant calculations performed: Here by redundant we mean having the value of the same coalition calculated by more than one agent, while it was enough for one agent to calculate it. The table shows that using DCVC results in no redundant calculations (because each agent knows the precise bounding of the calculations it should perform, and these are disjoint). In contrast, SK results in an exponentially large number of redundant calculations; this is because each agent's commitment to a set of coalitions is done with very limited knowledge about the other agents' commitments. For example, agent a_i's knowledge about agent a_j's commitments is restricted to the set S_{ji} that a_j sends to a_i. This means that a_i is not aware of the coalitions to which a_j has committed by contacting other agents. This results in having the agents

[6]The PC on which we ran our simulations had a processor: Pentium(R)4 2.80 GHz, with 1GB of RAM.

	Time (in seconds)		Redundancy		Communication (in bytes)		Memory (in bytes)		Difference	
	DCVC	SK	DCVC	SK	DCVC	SK	DCVC	SK	DCVC	SK
16 agents	< 0.01	2.27	0	513452	0	735408	2	65536	1	8424
17 agents	< 0.01	6.11	0	1208715	0	2350481	3	196608	1	12886
18 agents	< 0.01	13.76	0	2583828	0	4974743	3	393216	1	26071
19 agents	< 0.01	32.29	0	5506420	0	10538129	3	786432	1	52890
20 agents	< 0.01	72.27	0	11659720	0	22152227	3	1572864	1	103484
21 agents	< 0.01	159.89	0	24605666	0	46512635	3	3145728	1	208931
22 agents	< 0.01	372.58	0	52170535	0	97957698	3	6291456	1	454812
23 agents	< 0.01	881.64	0	108933551	0	204911555	3	12582912	1	880428
24 agents	0.01	2280.43	0	210504067	0	429009502	3	25165824	1	2191528
25 agents	0.02	5298.52	0	477826101	0	1188779705	4	67108864	1	3043149

Table 2. Simulation results.

commit to coalitions without knowing that other agents have already committed to them.

3. Communication between the agents: Communication is usually necessary in order for each agent to know its share of the calculations to perform. SK requires sending an exponentially large number of bytes between the agents; this is mainly because if an agent a_i commits to a set S_{ij} of coalitions, then a_j would have to subtract this set from its list, and in order to do so, a_i would have to send S_{ij} to a_j. In contrast, DCVC requires no communications between the agents because each agent knows its share of calculations by using the provided equations, and not by negotiating with other agents.

4. Memory requirements: Any coalition of n agents can be saved in memory using n bits, where each bit indicates whether an agent is a member of the coalition. However, since the minimum unit of memory that can be allocated is one byte, we can say that the memory required per coalition is $\lceil n/8 \rceil$ bytes. Given this, Table 2 shows the number of bytes required per agent to save the necessary coalitions. As can be seen, the memory requirements grow exponentially for SK. This is because their algorithm cannot be applied without having each agent start with a list of all the possible coalitions in which it is a member. However, when using DCVC, each agent only needs to maintain in memory one coalition at a time. This makes DCVC particularly suitable for domains where very little memory space is available for the agents (e.g. agents located on mobile devices). In our case, for example, one kilobyte of memory per agent would be enough for up to 8192 agents, while each agent would have required more than $5.2 * 10^{2459}$ Gigabytes if it used SK.

5. Equality of agents' shares: Table 2 shows the difference between the agent that had the biggest share of the calculations and the one that had the smallest. DCVC has a maximum difference of 1 (because of the way it maintains and updates α). However with SK, the difference grows exponentially with the number of agents. This is because the agents' shares were arbitrarily determined based on the order in which they contacted each other. Thus, some agents were contacted by more agents than others, and so removed more coalitions from their list, and ended up with smaller shares. On the other hand, some agents contacted more agents than others, and thus committed to more coalitions, and ended up with larger shares.

Conclusions and Future Work

In this paper, we developed a novel algorithm for distributing the coalitional value calculations among cooperative agents. We then benchmarked the performance of our algorithm against the only available one in the literature. This comparison showed that our algorithm is significantly faster, requires significantly less memory space, and requires much less communication. These improvements stem from the fact that our algorithm performs no redundant calculations and distributes the calculations equally among the agents. Thus, DCVC can be seen to represent a significant advance in the state of the art.

For future work, we will concentrate on developing the enforcement mechanism so that DCVC can be applied in environments where the agents are selfish. In such cases, the agents might not necessarily perform all the calculations they are assigned or they might lie about the results they found in order to improve the outcome for themselves. The enforcement mechanism should motivate the agents to calculate the values they are assigned and to truthfully reveal the results they find.

References

Conway, J. H., and Guy, R. K. 1996. *The Book of Numbers*. New york, USA: pub-COPERNICUS.

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

Norman, T. J.; Preece, A.; Chalmers, S.; Jennings, N. R.; Luck, M.; Dang, V. D.; Nguyen, T. D.; Deora, V.; Shao, J.; Gray, A.; and Fiddian, A. 2004. Agent-based formation of virtual organisations. *International Journal of Knowledge Based Systems* 17(2–4):103–111.

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.

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. P.; Chen, Y.; and Ying, J. 2000. Customer coalitions in the electronic marketplace. In *AAAI/IAAI*, 1133–1134.