Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 171 (2007) 535–567

Artificial Intelligence

www.elsevier.com/locate/artint

An algorithm for distributing coalitional value calculations among cooperating agents

Talal Rahwan *, Nicholas R. Jennings

School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK

Received 4 May 2006; received in revised form 31 January 2007; accepted 6 March 2007

Available online 12 March 2007

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, instead of having a single agent calculate all these values (as is typically the case), it is more efficient to distribute this calculation among the agents, thus using all the computational resources available to the system and avoiding the existence of a single point of failure. 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, has minimal memory requirements, and can reflect variations in the computational speeds of the agents. To evaluate the effectiveness of our algorithm, we compare it with the only other algorithm available in the literature for distributing the coalitional value calculations (due to Shehory and Kraus). This shows that for the case of 25 agents, the distribution process of our algorithm took less than 0.02% of the time, the values were calculated using 0.000006% of the memory, the calculation redundancy was reduced from 383229848 to 0, and the total number of bytes sent between the agents dropped from 1146989648 to 0 (note that for larger numbers of agents, these improvements become exponentially better).

Keywords: Coalition formation; Multi-agent systems

1. 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 set of tasks, 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, especially when tasks cannot be performed by a single agent or when a group of agents performs the tasks more efficiently. Now, if we view the population of agents as a set A, then every non-empty 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

* Corresponding author.
E-mail address: tr03r@ecs.soton.ac.uk (T. Rahwan).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2007.03.002

could be derived if that coalition was formed.[1] 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 that 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 Section 2 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 faster and the agents can share the burden of the calculations.[2] To this end, there are a number of desiderata that we can place on such a distribution algorithm:

(a) 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.
(b) Communication between the agents should be minimized, particularly when the agents have limited communication bandwidth.
(c) 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.
(d) In order to minimize the time taken, the computational load should be balanced among the agents. In other words, if the agents have equal processing capabilities, each one of them should compute an equal number of values, and if they have unequal capabilities, the faster computational agents should take on a greater burden of the calculations.
(e) The amount of memory that each agent requires in order to perform the computations should be minimized, particularly when the agents have memory constraints.[3]

Moreover, in most practical situations, the agents continuously form coalitions whenever new ones are necessary and formed coalitions are dissolved whenever it is beneficial to do so. As a result of this continuous change, the process of calculating the coalitional values is not a one-shot activity. For example, additional tasks may be requested from the collective, or the resources available to the agents may change. In either case, the values need to be re-calculated to take these changes into consideration. Note that this re-calculation process might differ from the initial calculation process in the sense that some of the agents might no longer be able to join subsequent coalitions. For example, in cases where the coalitions are not allowed to overlap (e.g. [13]), then after a coalition is formed, every agent that has joined that coalition is no longer available to join other coalitions (until that coalition is dissolved). Another example is in the case where each agent requires a certain number of resources in order to join a coalition (e.g. [14,15]). In this case, the agents that have now used all of their resources in one or more coalitions, can no longer be considered. Based on this, whenever a coalition needs to be formed, the agents must only take into consideration that subset of agents which is currently available and eligible.

Against this background, we present a novel algorithm (called DCVC) for Distributing Coalitional Value Calculations among the constituent agents. Here, the agents are assumed to be cooperative (i.e. they carry out their share of computations, and they report the results truthfully). However, the underlying algorithm can also be applied in environments where the agents are non-cooperative (i.e. they act to increase their own outcome and may lie about the results if they find 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.

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,

[1] The way this value is calculated depends on the problem domain, and the complexity of this calculation varies correspondingly from linear to exponential.

[2] Note that the calculations are independent of one another and thus can be done in a distributed manner if an effective distribution algorithm can be found.

[3] 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).

distributes the calculations equally,[4] and enables each agent to perform its share of calculations without having to maintain in memory more than one coalition. We also show how the algorithm can be modified for the case where the agents have different computation speeds, and prove that the resulting distribution minimizes the computation time. To benchmark the effectiveness of our algorithm, we compare it with the only other algorithm available in the literature [15]. In so doing, we show that for the case of 25 agents, the distribution process of our algorithm took less than 0.02% of the time, the values were calculated using 0.000006% of the memory, the calculation redundancy was reduced from 383229848 to 0, and the total number of bytes sent between the agents dropped from 1146989648 to 0. Note that for larger numbers of agents, these improvements become exponentially better.

Particularly, this paper advances the state of the art in the following ways:

- We present a new algorithm for efficiently distributing the coalitional value calculations.
- We show why, in order to let the agents finish at the same time, it is not sufficient to consider *how many* coalitions an agent is assigned, but also *which* coalitions an agent is assigned.
- We show how DCVC can be modified to reflect the variations in the agents' computational speeds, and prove that the resulting distribution minimizes the required time.
- We analyze the different cases in which only a subset of agents is available or eligible to join a coalition, and discuss a number of methods for distributing this subset.
- We calculate the exact number of operations required by these methods, and show that DCVC requires significantly fewer operations, compared to the other methods.
- We benchmark DCVC against Shehory and Kraus' algorithm and show that it significantly outperforms it on all relevant dimensions.

The remainder of this paper is organized as follows. Section 2 discusses related research. In Section 3, we present the basic DCVC algorithm, given that every agent is able to join a coalition, while, in Section 4, we show how DCVC can be generalized to the case where only a subset of the agents in the system can join a coalition.[5] In Section 5, we calculate the complexity of DCVC, and in Section 6, we evaluate its performance. Section 7 concludes and presents future work.

2. Literature review

Coalition formation has received a considerable amount of attention in recent research, and it has the potential to be useful in a number of scenarios and multi-agent systems. In e-commerce, buyers can form coalitions to purchase a product in bulk and take advantage of price discounts [16]. In e-business, groups of agents can be formed in order to satisfy particular market niches [8]. In distributed sensor networks, coalitions of sensors need to work together to track targets of interest [4]. In distributed vehicle routing, coalitions of delivery companies can be formed to reduce the transportation costs by sharing deliveries [11]. Coalition formation can also be used for information gathering, where several information servers form coalitions for answering queries [6].

Generally speaking, however, the coalition formation process can be considered to include three main activities [12]:

(a) *Coalition structure generation*: partitioning the set of agents into exhaustive and disjoint coalitions. Such a partition is called a *coalition structure*. For example, given a set of agents $A = \{a_1, a_2, a_3\}$, there exist seven possible coalitions: $\{a_1\}$, $\{a_2\}$, $\{a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_2, a_3\}$, $\{a_1, a_2, a_3\}$, and five possible coalition structures: $\{\{a_1\}, \{a_2\}, \{a_3\}\}$, $\{\{a_1\}, \{a_2, a_3\}\}$, $\{\{a_2\}, \{a_1, a_3\}\}$, $\{\{a_3\}, \{a_1, a_2\}\}$, $\{\{a_1, a_2, a_3\}\}$.

[4] When the total number of coalitions is not exactly divisible by the number of agents, the size of the agents' shares will differ by one. However, this additional calculation is assigned to the agents such that the average size of the shares is exactly equal. Therefore, through out this paper, we will refer to the agents' shares as being equal.

[5] Note that this latter case is the most general one because a subset can be smaller than, or equal to, the total set of agents. This means that the algorithm we developed for the specific subset case can be used both when every agent is able to join a coalition and when only a subset of the agents can join. Despite this fact, for explanatory purposes, we first deal with the simpler case in which the algorithm deals with all the agents (in Section 3), before dealing with this generalized case (in Section 4).

(b) *Optimizing the value of each coalition*: pooling the tasks and resources of the agents in every coalition in the coalition structure in order to maximize the coalition value.
(c) *Payoff distribution*: dividing the value of each coalition among its members so as to achieve stability or fairness.

Now, clearly, these activities interact [12]. For example, the coalition that an agent wants to join depends on the payoff that it is likely to receive in each potential coalition (activities (a) and (c)). Note, however, that in the case of cooperative environments, the agents are concerned with maximizing the system outcome, and thus are willing to join the coalition that maximizes the social welfare, regardless of their share of the coalition value. Therefore, payoff distribution is less important, and the main concern is generating a coalition structure so as to maximize the social welfare.

To date, much of the existing research on coalition formation in game theory has focused on payoff distributions, where it is usually assumed that coalitions have already been formed. In this context, many solutions have been proposed based on different *stability* concepts (e.g. the core, the Shapley value, and the kernel [9]). Here, stability refers to the state where, once a stable solution is found, the agents have no incentive to deviate from the coalitions to which they belong (or little incentive in weaker types of stability). This is desirable because it avoids the costs associated with having the agents evaluate reasons for leaving their current coalitions and joining other ones. Note, however, that game theory does not provide algorithms that the agents can use in order to actually form the coalitions [15]. Moreover, game theoretic approaches are typically centralized, and do not take into consideration the resource constraints of a computational environment (such as communication bandwidth and limited computation time). Given our focus on computational multi-agent systems, this is a serious shortcoming.

Traditionally, much of the research on coalition formation has focused on *super-additive environments*, in which any combination of two groups of agents into a new group is beneficial [5,17]. In such environments, the process of searching for the coalition that maximizes the system welfare is trivial, since this coalition will be the one in which every agent is a member (commonly known as the *grand coalition*). This assumption, however, does not hold for many real-world applications, due to the intra-coalition coordination and communication costs which increase with the size of the coalition [11].

However, in *non-super-additive environments*, finding the optimal coalition(s) requires searching the whole set of possible coalition structures. This is computationally complex due to the size of the set 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. In this context, we briefly discuss the most widely used algorithms in the literature.

Specifically, Sandholm et al. [12] have proved that no bound from the optimal can be established without first searching through an exponential subset of the *coalition structure graph*.[6] They also present an anytime algorithm that can establish a worst-case bound by searching the bottom two levels of the graph, and 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 [3] 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 again they do not consider the case of overlapping coalitions. To combat this complexity, Shehory and Kraus [15] set limitations on the size of the permitted coalitions which, in turn, 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 [12].

Now 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

[6] For more details on the coalition structure graph see Sandholm et al. [12].

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 more space than others (they justify this by presenting an enforcement mechanism that motivates the agents to search exactly what they are assigned, no matter how unfair the assignment is). Moreover, their distribution does not take into consideration the different computational speeds of the agents. 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 [15] do present an algorithm for distributing the value calculations among the agents (the same distribution algorithm, with slight differences, was also mentioned in [13,14]). Specifically, their method works by making the agents negotiate about which of them performs which of the calculations (see Section 6 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 (again see Section 6 for details).

To address these shortcomings, we developed an algorithm that distributes the coalitional value calculations efficiently among the agents. Like Shehory and Kraus' algorithm, ours can be applied for environments where the coalitions are allowed to overlap, where some agents might no longer be available to join new coalitions, and where the algorithm imposes specific limitations on the coalitional sizes (cf. Sandholm et al.'s algorithm). Therefore, when it comes to benchmarking performance, we compare our algorithm with Shehory and Kraus' algorithm (henceforth called SK).

3. The DCVC algorithm

For illustrative purposes, we start by presenting a basic version of DCVC in which the differences between the agents' shares are minimized (Section 3.1). After that, we show how the required time can be further reduced, by modifying *which* values an agent calculates, rather than *how many* values an agent calculates (Section 3.2). Finally, we show how DCVC can be modified for the case where the agents have different computation speeds, and prove that the resulting distribution minimizes the computation time (Section 3.3). Note that in this section, we assume that every agent is able to join a coalition (Section 4 deals with situations where this is not the case).

3.1. The basic algorithm

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 be formed. For example, if coalitions of a particular size are not allowed to be formed, then the agents simply do not distribute the coalitions of this size among themselves. In such cases, the agents still calculate the same number of values. This is important since it makes DCVC applicable for coalition formation algorithms that reduce the complexity of the search by limiting the size of the coalitions (as discussed in Section 2).

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. Also, let L_s be an ordered list of possible coalitions of size $s \in S$, and N_s be the number of coalitions in L_s (i.e. $N_s = |L_s|$). Finally, let $C_i = \{c_{i,1}, \ldots, c_{i,s}\}$ denote the coalition located at index i in the list L_s, where each element $c_{i,j}$ is an integer representing agent $a_{c_{i,j}}$ (for example, $C_i = \{2, 3, 5\}$ corresponds to the coalition of agents a_2, a_3, a_5). Now, for any $s \in S$, we define the order in the list L_s as follows:

- The first coalition in the list is: $\{n - s + 1, \ldots, n\}$.

Table 1
The lists of possible coalitions for 6 agents

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			

- The last coalition in the list is: $\{1, \ldots, s\}$.
- Given any coalition C_i, the agent can calculate C_{i-1} by checking the values $c_{i,s}, c_{i,s-1}, c_{i,s-2}, \ldots$ until it finds a value $c_{i,x}$ such that $c_{i,x} < c_{1,x}$, then:
  - $c_{i-1,k} = c_{i,k}$: $1 \leqslant k < x$
  - $c_{i-1,k} = c_{i,k} + 1$: $k = x$
  - $c_{i-1,k} = c_{i-1,k-1} + 1$: $x < k \leqslant s$

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

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 Fig. 1). Here, each agent is assumed to know the total number of agents, as well as the set of permitted sizes of coalitions; we also assume that each agent has a unique global identifier (UID) by which it is identified by other agents. The existence of such an identifier is a reasonable assumption since all agents need to be uniquely identifiable so that messages can be routed correctly.

In more detail, each agent starts by sorting the set of agents according to their UID in an ascending order. 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 $\vec{A}$. 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. Each agent a_i starts by calculating the number of coalitions in $L_{s,i}$:

$$N_{s,i} = \lfloor N_s / n \rfloor \tag{1}$$

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:

$$index_{s,i} = i \times N_{s,i}$$

The agent now calculates the values of all the coalitions in $L_{s,i}$. This is done without maintaining L_s in memory, or even maintaining $L_{s,i}$. Instead, the agent allocates a space of memory, denoted by $M = \{m_1, \ldots, m_n\}$, which is

Each agent a_i should perform the following:

- Sort the set of agents based on the agents' UID in an ascending order.
- Set: $\alpha = 1$.
- For every $s \in S$, do the following:
  - **1. If** $(N_s \geqslant n)$ then:
    - **1.1.** Calculate the size of your share: $N_{s,i} = \lfloor N_s/n \rfloor$
    - **1.2.** Calculate the index of the last coalition in your share: $index_{s,i} = i \times N_{s,i}$
    - **1.3.** Calculate the values of each coalition in your share.
    - **1.4.** Calculate the number of additional values that need to be calculated: $N' = N_s - (n \times N_{s,i})$

    Otherwise:
    - **1.5.** Calculate the number of additional values that need to be calculated: $N' = N_s$
  - **2. If** $(N' > 0)$ then:
    - **2.1** 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 $(\alpha + N' - 1 \leqslant n)$ then: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_{\alpha+N'-1})$ else: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_n, a_1, \ldots, a_{(\alpha+N'-1)-n})$
      - If $(a_i \in A')$ then calculate one of the additional values based on your position in A'
      - If $(\alpha + N' \leqslant n)$ then: $\alpha = \alpha + N'$, else: $\alpha = \alpha + N' - n$

Fig. 1. The DCVC algorithm (basic version).

sufficient to maintain one coalition at a time. Basically, the agent starts by setting M to the last coalition in $L_{s,i}$ (i.e., to the one located at: $index_{s,i}$) and calculates its value. After that, the agent sets M to the coalition before it (i.e. to the coalition located at: $index_{s,i} - 1$) and calculates its value, and so on. This is done until the value of every coalition in $L_{s,i}$ is calculated.

Note that the agent so far has calculated the number of coalitions in its share, as well as the index in L_s at which its share ends. This information alone would be sufficient for the agent to directly know which coalitions belong to its share if it maintained L_s. However, since this is not the case, then knowing *where* the coalitions are located in L_s does not imply knowing *what* those coalitions are. Now, from the way the list is ordered, given a coalition in L_s, the agent can always find the coalition before it. Based on this, the agent would only need to set M to the last coalition in $L_{s,i}$, and this would be sufficient for it to find all the coalitions in $L_{s,i}$. But again, since the agent does not maintain L_s, then knowing the index of the last coalition does not give the coalition directly. For this reason, we show how an agent can find a 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. Here, $n!$ represents n factorial (i.e., if $n > 0$ then: $n! = 1 \times 2 \times \cdots \times n$, and if $n = 0$ then $n! = 1$):

$$C_s^n = \frac{n!}{(n-s)! \times s!} \tag{2}$$

Now let $P(i, \{i+1, \ldots, n\})$ be the list of all possible coalitions of agents $a_{i+1}, \ldots, a_n$ after adding a_i in the beginning of each coalition. Also, let $P_s(i, \{i+1, \ldots, n\})$ be the list of all the coalitions in $P(i, \{i+1, \ldots, n\})$ that are of size s.[7] From (2) we find that the number of coalitions in $P_s(i, \{i+1, \ldots, n\})$ is:

$$\left|P_s\big(i, \{i+1, \ldots, n\}\big)\right| = C_{s-1}^{n-i} \tag{3}$$

Now, since L_s is ordered as specified earlier, then L_s contains $P_s(i, \{i+1, \ldots, n\})$ 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 that starts with $(n-s+1) - i + 1$ must have an index k such that:

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

[7] In other words, $P_s(i, \{i+1, \ldots, n\})$ would be the list of all possible coalitions of size $(s-1)$ that contain agents $a_{i+1}, \ldots, a_n$, after adding a_i in the beginning of each coalition. By this, each coalition in the list becomes of size s.

1. Set $j = 1$, $index = index_{s,i}$, $s_1 = s$.
2. Check the values: $Pascal[s_1, 1]$, $Pascal[s_1, 2]$, ... until you find a value: $Pascal[s_1, x] \geqslant index$
3. Set $m_j = (n - s_1 + 1) - x + 1$.
4. If $(Pascal[s_1, x] = index)$ then:
- Set the rest of the coalition as follows: $m_{k+1} = m_k + 1$: $k = j, \ldots, s - 1$

Otherwise:
- Set: $j = j + 1$, $index = index - Pascal[s_1, x - 1]$, $s_1 = s_1 - 1$
- Move to step 2.

Fig. 2. Setting M to the coalition located at $index_{s,i}$ in L_s.

$$k \leqslant \sum_{j=1}^{i} \left| P_s\big((n - s + 1) - j + 1, \big\{(n - s + 1) - j + 2, \ldots, n\big\}\big) \right|$$

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

$$k > \left| P_4\big(3, \{4, 5, 6\}\big) \right| + \left| P_4\big(2, \{3, 4, 5, 6\}\big) \right| = 1 + 4 = 5$$
$$k \leqslant \left| P_4\big(3, \{4, 5, 6\}\big) \right| + \left| P_4\big(2, \{3, 4, 5, 6\}\big) \right| + \left| P_4\big(1, \{2, 3, 4, 5, 6\}\big) \right| = 1 + 4 + 10 = 15$$

Therefore, based on (3), we know that any coalition in L_s that starts with $(n - s + 1) - i + 1$ must have an index k such that:

$$k > \sum_{j=1}^{i} C_{s-1}^{s+j-2}$$
$$k \leqslant \sum_{j=1}^{i+1} C_{s-1}^{s+j-2}$$

Based on this, we present an algorithm for setting M to the coalition located at $index_{s,i}$ without maintaining L_s (see Fig. 2).

At first, the agents form what we call a *Pascal matrix* which is of size: $(n - 1) \times (n - 1)$. The matrix includes values from Pascal's triangle[8] and is calculated as follows:

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

By this, the following equation holds:

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

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 $Pascal[s, x] \geqslant index_{s,i}$. The first member would then be $(n - s + 1) - x + 1$. (Step 1 in Fig. 3 shows how to find the first member in a coalition that is located at index 46 in the list L_5 for 9 agents.)

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 $(n - s + 1) - x + 1$ without including the first member of each coalition. This sub-list is similar to L_{s-1}, the only difference is that it contains $P_s(i, \{i + 1, \ldots, n\})$ with i running down to $(n - s + 2) - x + 1$ instead of running down to 1 (see the list in Fig. 3, 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

[8] More details about Pascal triangles can be found in [2].

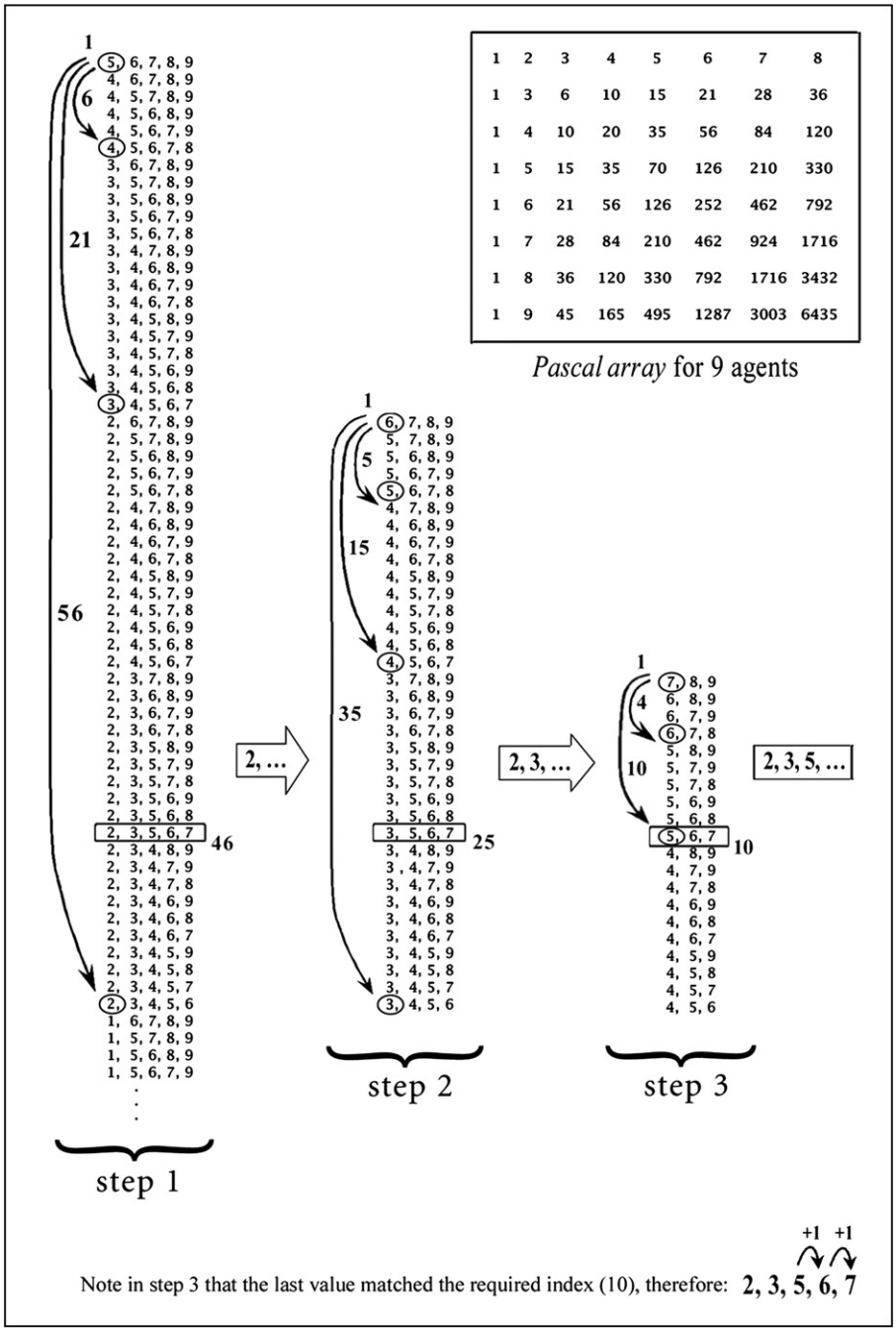

Fig. 3. Finding a coalition at *index* = 46 in the list L_5 of coalitions of 9 agents.

becomes: $46 - 21 = 25$). Based on this, the agent can find the next member in the coalition by checking the values: $Pascal[s-1,1], Pascal[s-1,2], \ldots$ until it finds a value $Pascal[s-1,x] \geqslant index_{s,i} - Pascal[s,x-1]$, 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 the *Pascal matrix* 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: $m_{k+1} = m_k + 1$: $k = j, \ldots, s-1$. Fig. 3 shows a complete example for setting M to the coalition at *index* = 46 in the list L_5 for 9 agents.

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

- Calculate the value of M.[9]
- Set M to the coalition before it. This is done by checking the following values: $m_s, m_{s-1}, m_{s-2}, \ldots$ until it finds a value m_β such that $m_\beta < c_{1,\beta}$. Then, the values m_k: $k < \beta$ remain unchanged, while the remaining values are calculated as follows:
  - $m_k = m_k + 1$: $k = \beta$
  - $m_k = m_{k-1} + 1$: $\beta < k \leqslant s$

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:

$$N' = N_s - \sum_{j=1}^{n} N_{s,j} = N_s - \left(n \times \lfloor N_s/n \rfloor\right) \tag{4}$$

Here, the coalitions that need their values to be calculated would be: $C_{N_s-N'+i}$: $i \in 1, \ldots, N'$. 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 equal 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. In 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 (which is insignificant compared to the general problem). Updating α is done as follows:

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

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

$$\begin{aligned} &\text{If } (\alpha + N' - 1 < n) \quad \text{then } A' = (a_\alpha, a_{\alpha+1}, \ldots, a_{\alpha+N'-1}) \\ &\qquad\qquad\qquad\qquad\quad \text{else } A' = (a_\alpha, a_{\alpha+1}, \ldots, a_n, a_1, \ldots, a_{\alpha+N'-n}) \end{aligned}$$

For example, if we have 6 agents, then from Eq. (4) we find that for the list 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, and for L_4, we have $N' = 3$. Therefore, A' would be (a_6, a_1, a_2) and α becomes 3. Finally, for L_6, we have $N' = 1$, and therefore, A' would be (a_3) and α becomes 4 (see A' in Fig. 4). 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.[10]

[9] The details of how to calculate this value are left for the developers to decide, based on the problem under investigation. For example, in an electronic marketplace, the value of a coalition of buyers can be calculated as the difference between the sum of the reservation costs of the coalition members and the minimum cost needed to satisfy the requests of all the members [7]. In information gathering, the coalition value can be designed to represent a measure of how closely the information agents' domains are related [6]. And in cases where the agents' rationality is bounded due to computational complexity, the value of a coalition may represent the best value it can get given limited computational resources for solving the problem [11].

[10] Given a large number of agents, the number of additional values would be insignificant, compared to the total number of values calculated by each agent. However, for a small number of agents, it is worthwhile to add this extra step to distribute these values, rather than simply having every agent a_i: $i \leqslant N'$ calculate one additional value. For example, if we have 6 agents, then by using this extra step, the average number of values calculated by agents: $a_1, a_2, a_3, a_4, a_5, a_6$ would be: 10.5, 10.5, 10.5, 10.5, 10.5, 10.5 respectively, while it would have been: 13, 12, 11, 11, 9, 9, 9 if the agents used the simpler distribution.

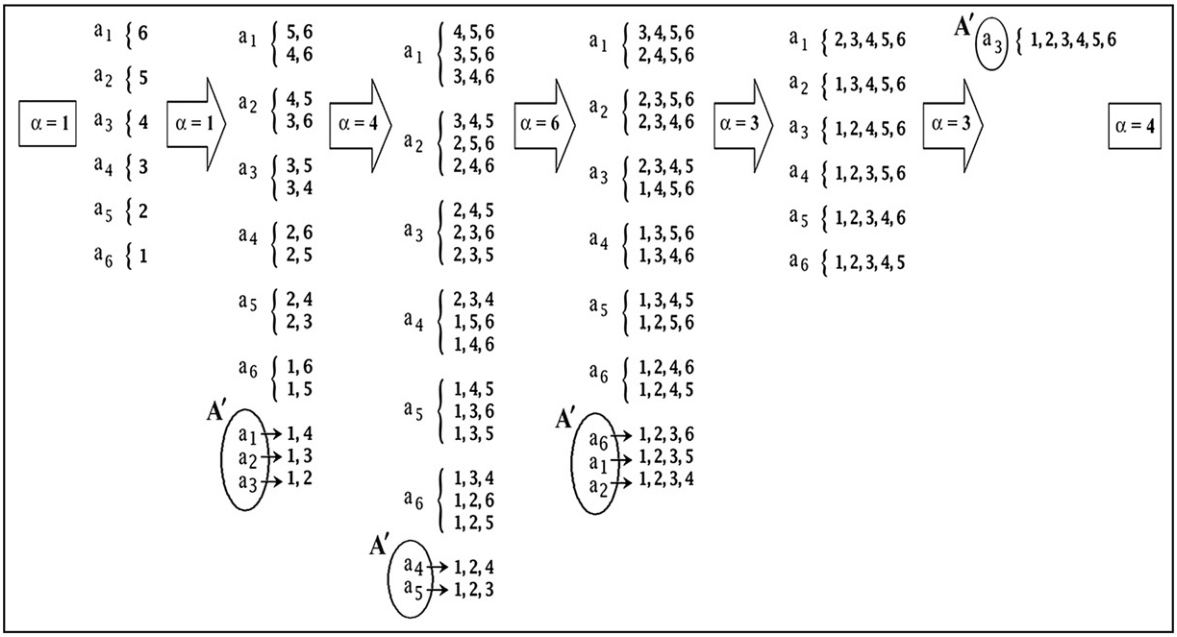

Fig. 4. The resulting distribution for all possible coalitions of 6 agents.

To illustrate how DCVC works, Fig. 4 shows an example of the resulting distribution among 6 agents. As shown in the figure, the agents' shares are exhaustive and disjoint. Note that this distribution was done without any communication between the agents and without any central decision maker. Moreover, each agent only needed to allocate a space of memory which is sufficient to save one coalition.

Finally, note that although DCVC gives every agent the ability to save one coalition at a time, the agent can still choose to maintain its entire share of coalitions, provided that it has sufficient memory space. This way, the agent avoids performing the operations required to set M from one coalition to another, throughout the list, every time a coalition is formed.

3.2. Modifying the coalitions to which an agent is assigned

By using the distribution process specified earlier, any two agents require an equal time to calculate their share of values. The distribution, however, is done without taking into consideration the time required for each agent to set M from one coalition to another (i.e. the time required to shift M up in the list by 1 coalition). In more detail, after an agent calculates the value of a coalition, it needs to set M to the coalition before it. This is done by performing a number of comparisons and additions as shown earlier in Section 3.1. Specifically, changing x values in M requires performing x comparisons, as well as x additions, which gives a total of $(2 \times x)$ operations. See Fig. 5 for an example.

As shown in the figure, the agent first searches for β, and then changes the values $m_\beta, m_{\beta+1}, \ldots, m_s$. This means that the agent would perform more operations for smaller values of β. Note that by ordering L_s as specified earlier in Section 3.1, β would generally have smaller values in the coalitions that are located at smaller indices in L_s. Now since the distribution is done such that the agents located at smaller indices in $\vec{A}$ calculate the values of the coalitions located at smaller indices in L_s, these agents would generally perform more operations. For example, for the case of 7 agents, Fig. 6 shows how agents a_2 and a_6 set M from one coalition to another through the lists $L_{4,2}$ and $L_{4,6}$ respectively. As shown in the figure, a_2 requires changing a total of 9 values, and thus performs 18 operations, while a_6 requires changing a total of 6 values, and thus performs only 12 operations. Therefore, although they both calculate the same number of coalitional values, agent a_2 would finish after a_6.

The differences between the agents grow with the number of agents involved. For example, for the case of 31 agents, Fig. 7(A) shows the time required for each agent to set M to the coalitions in its share. As shown in the figure, the required time differs considerably from one agent to another. Now in order to reduce these differences, the

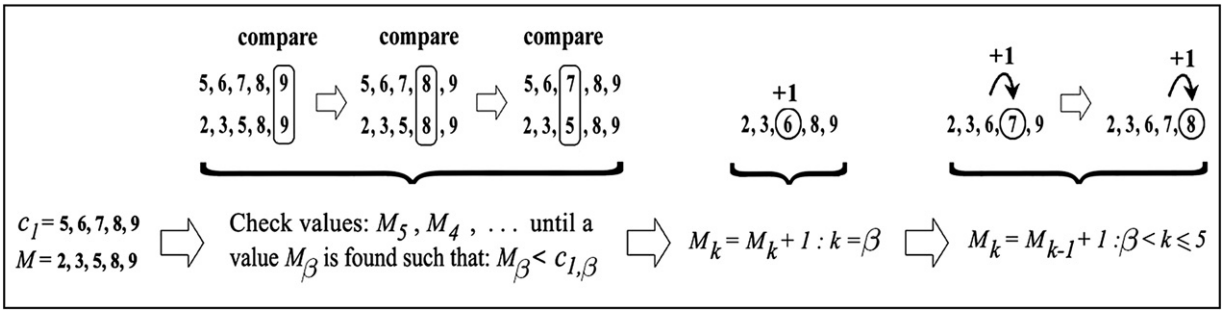

Fig. 5. An example for setting M to the coalition before it in the list L_5 of 9 agents.

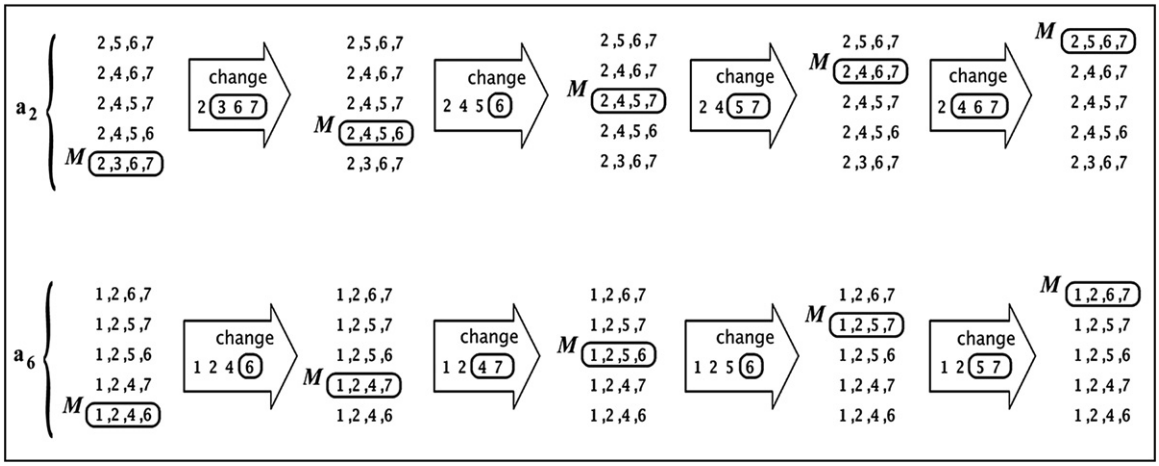

Fig. 6. For the case of 7 agents, the figure shows how a_2, a_6 set M from one coalition to another through the lists $L_{4,2}$, $L_{4,6}$ respectively.

distribution needs to be modified. This involves modifying *which* values an agent calculates, rather than *how many* values it calculates. In more detail, instead of having agent a_i calculate a list of sequential coalitions, a_i's share can be divided into two sub-lists $L^1_{s,i}$ and $L^2_{s,i}$, where each sub-list is located at a different position in L_s. This provides the ability to reduce the differences between the agents by adjusting both the size and the position of the sub-lists of every agent.

To this end, let $N^j_{s,i}$ be the number of coalitions in $L^j_{s,i}$ (i.e. $N^j_{s,i} = |L^j_{s,i}|$), and let $index^j_{s,i}$ be the index in L_s at which $L^j_{s,i}$ ends. The modification can then be expressed as follows:

- Each agent a_i calculates $N_{s,i}$ using Eq. (1), and then calculates the number of coalitions in the sub-lists $L^1_{s,i}$ and $L^2_{s,i}$ as follows:[11]

$$N^1_{s,i} = \lfloor N_{s,i} \times 0.4 \rfloor \tag{5}$$

$$N^2_{s,i} = \lceil N_{s,i} \times 0.6 \rceil \tag{6}$$

- After that, the agent calculates N' using Eq. (4), and then calculates the indices at which the sub-lists end; this is done as follows:

$$index^1_{s,i} = i \times N^1_{s,i}$$

$$index^2_{s,i} = N_s - N' - \left((i-1) \times N^2_{s,i}\right)$$

[11] The values 0.4 and 0.6 that are presented in the equations were determined via empirical studies. In this case, a range of different values were explored for a range of coalition scenarios, and these values were consistently the most efficient.

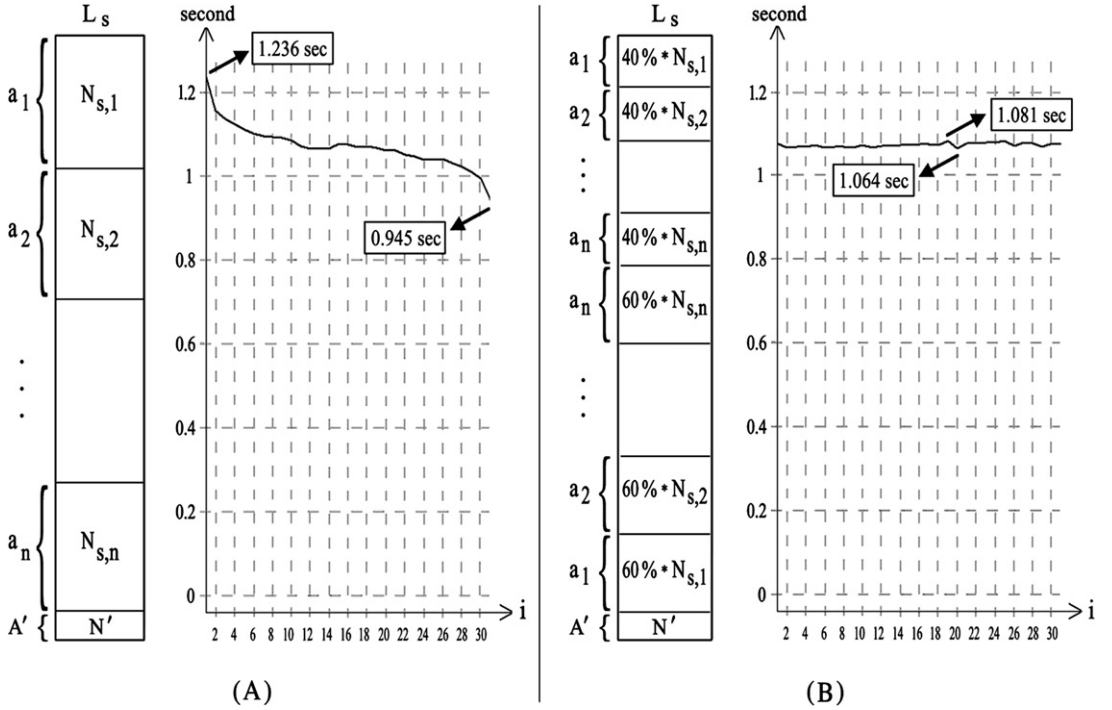

Fig. 7. For the case of 31 agents with equal computational speeds, the figure shows the time required for each agent to set M to the coalitions in its share. (A) shows the case where each agent's share consists of a set of sequential coalitions, while (B) shows the case where each agent's share is divided into two sub-lists.

- The calculation of the coalitional values is then handled using the same method specified in Section 3.1.

In more detail, the modification works as follows. For each agent a_i, the sub-list $L^1_{s,i}$ is located at the upper part of L_s, and the sub-list $L^2_{s,i}$ is located at the lower part. Moreover, the higher $L^1_{s,i}$ is, the lower $L^2_{s,i}$ is. By this, the total number of operations performed for both sub-lists is broadly the same for all the agents. For example, for the case of 31 agents with equal computational speeds, Fig. 7(B) shows the modified distribution, as well as the time required for each agent to set M to the coalitions in its share.[12] As shown in the figure, the differences between the agents were considerably reduced. In particular, the difference between the first agent to finish and the last dropped from 291 to 17 milliseconds (i.e. it was reduced by 94.2%). Moreover, the time required for the distribution process dropped from 1.236 to 1.081 (i.e. it was reduced by 12.5%).

One could argue that the differences between the agents can be further reduced by having more than two sub-lists. However, since setting M to a coalition located at a particular index requires more operations than setting M from one coalition to another, and since every sub-list requires calculating the index at which it ends, as well as setting M to the coalition located at that index, then, by having more sub-lists, the number of operations required becomes greater, and adjusting both the size and the position of each sub-list becomes more complicated. Based on this, as well as the fact that having two sub-lists reduces the differences considerably between the agents, we only divide the agents' shares into two sub-lists.

3.3. Considering different computational speeds

As mentioned earlier, the DCVC algorithm distributes the required calculations such that each agent gets an equal share (with a possible difference of at most one calculation). This distribution is efficient if the agents do not need to take into consideration the differences in computational speeds (e.g. because all agents have the same computational

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

speed or because the differences are insignificant). However, in the case where the agents do have significantly different computational speeds, it is inefficient to have each agent calculate the same number of values. In such cases, the distribution needs to be done with respect to the agents' relative computation speed. In order to do so, we present the required modifications to the algorithm.

After sorting the set of agents, and setting α to 1, each agent calculates the time it requires to perform a particular number of operations; the number and type of these operations should be pre-determined by the developers. This can then be used to indicate the agent's relative computation speed. For example, the developers can agree on having each agent perform 10000 additions. Then, if agent a_i took a time $t_i = 20$ milliseconds, while a_j took a time $t_j = 40$, then this indicates that a_i has a computational speed twice as fast as a_j.

Now that each agent a_i has calculated t_i, it sends this value to every other agent. By this, each agent a_i would have t_j for every $j \neq i$. Note that this step is only performed once. The algorithm then distributes the calculations as follows:

Instead of calculating the number of coalitions in its share, each agent a_i calculates the number of coalitions in every agent's share. This is done using the following equation:

$$N_{s,j} = \left\lfloor \frac{N_s}{t_j \times \sum_{k=1}^{n} 1/t_k} \right\rfloor$$

The agent then calculates the values $N^1_{s,i}$ and $N^2_{s,i}$ using Eqs. (5) and (6), and then calculates the values $index^1_{s,i}$ and $index^2_{s,i}$ using the following equations:

$$index^1_{s,i} = \sum_{j=1}^{i} N^1_{s,j}$$

$$index^2_{s,i} = N_s - N' - \sum_{j=1}^{i-1} N^2_{s,j}$$

After each agent calculates the values of the coalitions in its share, some values might remain uncalculated. In this case, the number of the remaining coalitions is given as follows:

$$N' = N_s - \sum_{j=1}^{n} N_{s,j}$$

The calculation of these values is then handled using the method defined in Section 3.1. We now prove that by using the modified algorithm, we minimize the time required for all the values to be calculated.

Theorem 1. *For any size $s \in S$, the distribution specified in the DCVC algorithm minimizes the time required for all the values to be calculated.*

Proof. Since the agents perform the calculations in parallel, then the time required for all the values to be calculated is equal to the time required for the last agent to finish calculating its share. Therefore, minimizing the required time is equal to minimizing the value: $\max_{j=1}^{n}(t_j \times N_{s,j})$. Now since we need to distribute the calculation of N_s values among n agents, we can define the space of possible solutions as the space of vectors $V \subseteq R^n$ in which for every vector $\vec{v} \in V$, we have $\sum_{i=1}^{n} v_i = N_s$. Then, in order to minimize the value: $\max_{j=1}^{n}(t_j \times N_{s,j})$, we need to find a vector $x \in V$ that satisfies the following condition:

$$\forall(\vec{y} \in V)\big(\exists i \in \{1,\ldots,n\}\big)\colon t_i \times y_i < t_i \times x_i \Rightarrow \big(\exists j \in \{1,\ldots,n\}\big)\colon t_j \times y_j > t_j \times x_j \geqslant t_i \times x_i \tag{7}$$

That is, decreasing some component x_i must be at the expense of increasing some other component x_j such that $t_j \times y_j > t_i \times x_i$. Now since we have $\sum_{i=1}^{n} x_i = N_s$, then (7) implies that:

$$t_1 \times x_1 = t_2 \times x_2 = \cdots = t_n \times x_n$$

By solving this equation, we find that $x_i = N_s/(t_i \times \sum_{j=1}^{n} 1/t_j)$. This means that by making $N_{s,i} = N_s/(t_i \times \sum_{j=1}^{n} 1/t_j)$ for every $i \in \{1,\ldots,n\}$, we minimize the overall time of calculations. □

Note that the agent's actual share is $\lfloor N_s/(t_i \times \sum_{j=1}^{n} 1/t_j)\rfloor$ with a possibility of one additional calculation. This difference, however, is very small and is therefore not considered in the proof.

4. Generalizing DCVC to deal with subsets of agents

So far, we assumed that every agent is able to join any coalition. However, as discussed earlier, there are cases where this assumption does not always hold. Now let $A^* \subset A$ be the set of agents that are currently able to join any coalition, and let $n^* = |A^*|$. Similarly, let $\bar{A}^* = \{\bar{a}_1^*, \bar{a}_2^*, \ldots, \bar{a}_{\bar{n}^*}^*\}$ be the set of agents that are not able to join any coalition, where $\bar{n}^* = |\bar{A}^*|$ (this means that: $\bar{A}^* = A \backslash A^*$, and $\bar{n}^* = n - n^*$). Finally, let P denote the set of all potential coalitions that are of any size $s \in S$, and let P^* denote the subset of P in which every coalition contains only members of A^*. Based on this, every time the agents need to form a coalition, they only need to consider the coalitions that belong to P^*. Note that A^* is continuously changing due to the coalitions that are being formed. Also note that any change in A^* corresponds to a change in P^*, and whenever $A^* = A$, we have: $P^* = P$.

As mentioned earlier, after a particular coalition is formed, the coalitional values might need to be re-calculated before the agents can form another coalition. Now in order to perform this re-calculation process in a distributed manner, the set P^* must first be distributed among the agents. This can be done using either of the following methods:

- Searching through P.
- Repeating the entire distribution process.

We will first discuss the first method since this is the way that the SK algorithm handles the problem, and then we will show that simply repeating the distribution process (which DCVC can do, but SK cannot) is faster and more efficient.

4.1. Searching through P

In this method, the set P is initially distributed among the agents, and each agent a_i maintains its share of P (denoted by P_i). Then, whenever the coalitional values need to be re-calculated, each agent a_i searches through P_i, and finds the coalitions that belong to P^*. These coalitions are then the agent's share of P^*, for which it calculates the coalitional values. In more detail, based on the memory that is available to the agent, this can be done using one of the following approaches:

(a) Each agent maintains its share of P, but does not maintain its share of P^*. Therefore, whenever the coalitional values need to be re-calculated, each agent a_i has to search through P_i, and find the coalitions that belong to P^* (and that is even if P^* remains unchanged). Specifically, finding the coalitions that belong to P^* is done by finding those that contain only members of A^*. Note that we assume every coalition is written in memory using n bits, where each bit indicates whether an agent is a member of the coalition.[13] Therefore, finding whether a coalition belongs to P^* is done by first checking the bit that represents $\bar{a}_1^*$, and if it is set to 1, then the coalition contains a member of $\bar{A}^*$, and therefore does not belong to P^*. On the other hand, if it is set to 0, then the agent must check the bit that represents $\bar{a}_2^*$, and so on. This is repeated until a member of $\bar{A}^*$ is found in the coalition, or until all the members of $\bar{A}^*$ are checked.

(b) Each agent a_i does not only maintain P_i, but also maintains its share of P^* in a temporary list (denoted by $temp_i$). Obviously, this approach requires allocating a larger memory space (for example, if there are no limitations on the coalitional sizes, then this approach would require allocating 50% more memory space). However, using this approach requires performing fewer operations (see Section 5 for details).
To show how $temp_i$ can be used, let us first consider $\bar{A}^*_{prev}$ to be the previous value of $\bar{A}^*$ (i.e., $\bar{A}^*_{prev}$ is the set of agents that were not able to join other coalitions *during the previous re-calculation process*, while $\bar{A}^*$ is the set of agents that are *currently* not able to join other coalitions). Also, consider $\bar{A}^*_{removed}$ to be the set of agents that

[13] For example, given 8 agents, the coalition $\{a_1, a_2, a_4, a_7, a_8\}$ can be written in memory as: 11010011 instead of: 1, 2, 4, 7, 8. This way, writing a coalition in memory requires less space, and finding whether an agent belongs to the coalition requires checking a single value instead of searching the entire coalition.

belong to $\bar{A}^*_{prev}$, but do not belong to $\bar{A}^*$, and consider $\bar{A}^*_{added}$ to be the set of agents that belong to $\bar{A}^*$, but do not belong to $\bar{A}^*_{prev}$. By this, we have:

$$\bar{A}^* = \bar{A}^*_{prev} \backslash \bar{A}^*_{removed} \cup \bar{A}^*_{added}: \ \bar{A}^*_{removed} \cap \bar{A}^*_{added} = \phi \tag{8}$$

Now, let $\bar{n}^*_{prev}$ be the number of agents in $\bar{A}^*_{prev}$ (i.e., $\bar{n}^*_{prev} = |\bar{A}^*_{prev}|$), and let $\bar{n}^*_{removed}, \bar{n}^*_{added}$ be the number of agents in $\bar{A}^*_{removed}$ and $\bar{A}^*_{added}$ respectively. By this, we have:

$$\bar{n}^* = \bar{n}^*_{prev} - \bar{n}^*_{removed} + \bar{n}^*_{added} \tag{9}$$

Finally, let P^*_{prev} be the previous value of P^*, let A^*_{prev} be the previous value of A^*, and let $n^*_{prev} = |A^*_{prev}|$. Now, whenever the coalitional values need to be re-calculated, the agents would have one of the following possible cases:

(i) $\bar{A}^*_{removed} = \phi$, and $\bar{A}^*_{added} \neq \phi$:
Here, we have: $\bar{A}^* = \bar{A}^*_{prev} \cup \bar{A}^*_{added}$, and in this case: $P^* \subseteq P^*_{prev}$. Thus, in order to find the coalitions that belong to P^*, it is sufficient to search through P^*_{prev} (in other words, it is sufficient for each agent a_i to search through $temp_i$[14]). Now since every coalition in $temp_i$ does not contain members of $\bar{A}^*_{prev}$, and since we have: $\bar{A}^* = \bar{A}^*_{prev} \cup \bar{A}^*_{added}$, then every coalition in $temp_i$ that does not contain members of $\bar{A}^*_{added}$ would be a coalition that does not contain members of $\bar{A}^*$, and therefore belongs to P^*. Based on this, each agent a_i should search through $temp_i$, and copy the coalitions that do not contain members of $\bar{A}^*_{added}$ to a new list, then free the memory allocated to $temp_i$. This new list would then be the new $temp_i$.

(ii) $\bar{A}^*_{removed} = \phi$, and $\bar{A}^*_{added} = \phi$:
In this case, we have: $A^* = A^*_{prev}$, which implies that: $P^* = P^*_{prev}$. Since the agents already have their shares of P^*_{prev} maintained in memory, then no search is required.

(iii) $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} = \phi$:
Here, we have: $\bar{A}^*_{prev} = \bar{A}^* \backslash \bar{A}^*_{removed}$, and this implies that: $P^*_{prev} \subseteq P^*$. In this case, finding the coalitions that belong to P^* can be done using two different methods:

- Since $temp_i$ contains the coalitions in P_i that do not contain members of $\bar{A}^*_{prev}$, and since we have: $\bar{A}^*_{prev} = \bar{A}^* \cup \bar{A}^*_{removed}$, then agent a_i already knows the coalitions in P_i that do not contain members of $\bar{A}^* \cup \bar{A}^*_{removed}$. In this case, the agent must search through P_i, find the coalitions that do not contain members of $\bar{A}^*$, but contain members of $\bar{A}^*_{removed}$, and then add them to $temp_i$. In more detail, for every coalition in P_i, the agent searches for members of $\bar{A}^*_{removed}$, and if it finds any, then it searches for members of $\bar{A}^*$, and if it does not find any, then it adds the coalition to $temp_i$.
- This method involves finding the coalitions that belong to P^*, without taking into consideration the fact that each agent maintains its share of P^*_{prev}. In more detail, each agent a_i starts by emptying $temp_i$. After that, the agent searches through P_i, finds the coalitions that do not contain members of $\bar{A}^*$, and copies them to $temp_i$.

By using the first method (instead of the second), the coalitions that contain members of $\bar{A}^*_{removed}$ would always require more operations,[15] while the coalitions that do not contain members of $\bar{A}^*_{removed}$ might or might not require fewer operations.[16] However, even if they do require fewer operations, the number of these coalitions is often much smaller than the number of coalitions that contain members of $\bar{A}^*_{removed}$. Therefore, agent a_i performs fewer operations by not taking $temp_i$ into consideration.

(iv) $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} \neq \phi$:
Here, we have: $\bar{A}^* = (\bar{A}^*_{prev} \backslash \bar{A}^*_{removed}) \cup \bar{A}^*_{added}$. In this case, finding the coalitions that belong to P^* can also be done using two different methods:

[14] This is because whenever the coalitional values need to be re-calculated, the set $temp_i$ would initially contain the coalitions in P_i that belong to P^*_{prev} (because each agent a_i would initially have in memory the set $temp_i$ from the previous re-calculation process).

[15] Because instead of searching for members of $\bar{A}^*$, the agent first has to search for members of $\bar{A}^*_{removed}$, and then search for members of $\bar{A}^*$.

[16] Because instead of searching for members of $\bar{A}^*$, the agent has to search for members of $\bar{A}^*_{removed}$, and the number of these members might or might not be less than the number of the members of $\bar{A}^*$.

- Each agent a_i should first search through $temp_i$, find the coalitions that do not contain members of $\bar{A}^*_{added}$, and then copy them to a new list. This new list would then contain the coalitions in P_i that do not contain members of $(\bar{A}^* \cup \bar{A}^*_{removed})$.[17] After that, a_i should search through P_i, find the coalitions that do not contain members of $\bar{A}^*$, but contain members of $\bar{A}^*_{removed}$, and then add them to the new list. Finally, a_i should free the memory allocated to $temp_i$, and the new list would then be the new $temp_i$.
- This method finds the coalitions that belong to P^*, without taking into consideration the fact that each agent maintains its share of P^*_{prev}.

Here, using the first method (instead of the second) requires even more operations (compared to the case where: $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} = \phi$). This is because the agent must also search through $temp_i$, and find the coalitions that do not contain members of $\bar{A}^*_{added}$. Therefore, agent a_i performs less operations by not taking $temp_i$ into consideration.

Now that we have discussed the different approaches of the first method (i.e., searching through P), we will discuss the other method (i.e., repeating the entire distribution process) and then make a comparison between these two methods.

4.2. Repeating the entire distribution process

In this method, the agents initially distribute P^* (instead of P), and then repeat the entire distribution process of P^* whenever A^* is changed. Note that when using other distribution algorithms (e.g. [15]), this method is considered inapplicable, due to the communication that is required every time the distribution process is repeated, as well as the time required for each agent to re-build its entire share in memory. However, when using DCVC, the distribution process can be repeated without any communications between the agents. Moreover, the agents do not have to re-build any lists in memory, thus, reducing the required time (for more details, see Section 6). This makes repeating the distribution process a feasible possibility. Note that in order to distribute P^* (instead of P), DCVC should be modified as follows:

- *Replace N_s with N^*_s in Fig.* 1. This would be sufficient for calculating the number of coalitions in each agent's share of P^* (and not P), as well as calculating the index of the last coalition in the share, and the number of additional values.
- *Replace n with n^* in Fig.* 2. This would be sufficient for setting M to the last coalition in each agent's share.
- *Replace n with n^* when calculating C_1.* By this, $C_1 = \{n^* - s + 1, \ldots, n^* - 1, n^*\}$. This would be sufficient for setting M to the coalition before it (this is because searching for β is done by comparing values of M with values of C_1).

4.3. Comparing the distribution efficiency

We will now compare both methods (i.e., searching through P and repeating the entire distribution process) in terms of distribution efficiency (a comparison of both methods, in terms of computational complexity can be seen in Sections 5):

- When each agent a_i searches for the coalitions in P_i that belong to P^*, some agents might find significantly more coalitions than others, and thus end up with larger shares of P^*. In fact, using this method results in a random distribution of P^*. Clearly, this is not optimal since P^* needs to be distributed in a way that minimizes the calculation time. On the other hand, repeating the distribution process results in an optimal distribution of P^*. Although we have advanced qualitative reasons for the relative advantages of repeating the entire distribution process, we need to provide quantitative results to back this up. To this end, we tested both methods for the

[17] This is because $temp_i$ contains the coalitions in P_i that do not contain members of A^*_{prev}. Then, by finding the coalitions in $temp_i$ that do not contain members of $\bar{A}^*_{added}$, the agent actually finds the coalitions in P_i that do not contain members of $\bar{A}^*_{prev} \cup \bar{A}^*_{added}$. And from Eq. (8), we know that $(\bar{A}^*_{prev} \cup \bar{A}^*_{added}) = (\bar{A}^* \cup \bar{A}^*_{removed})$.

Table 2
For the case of 30 agents, the table shows the difference between the agent that had the biggest share of calculations and the one that had the smallest, given different values of $\bar{n}^*$

$\bar{n}^*$	Repeating the distribution	Searching each agent's share
1	1	4,445,182
2	1	5,105,232
3	1	3,808,067
4	1	1,482,807
5	1	1,120,504
6	1	679,789
7	1	382,358
8	1	213,289
9	1	116,533
10	1	62,869
11	1	31,964
12	1	16,638
13	1	8,290
14	1	4,431
15	1	2,627
16	1	1,402
17	1	681
18	1	350
19	1	197
20	1	106
21	1	59
22	1	29
23	1	16
24	1	9
25	1	4
26	1	3
27	1	2
28	1	1

case of 30 agents.[18] Here, we assume that $S = \{1, \ldots, 30\}$, and that the agents have equal computational speeds. Initially, each agent was given an equal share of P. Then, both of the methods were tested given different sizes of $\bar{A}^*$ (i.e., given different values of $\bar{n}^*$, ranging from 0 to 28[19]); the results were taken as an average of running a statistically significant number of times, and in every case, the members of $\bar{A}^*$ were randomly selected from A. Table 2 shows the difference between the agent that had the biggest share of the calculations and the one that had the smallest. As shown in the table, by having each agent a_i search for the coalitions in P_i that belong to P^*, the agents ended up with unequal shares. On the other hand, when repeating the distribution process, the difference between the agents was as small as possible.

- By having each agent a_i search through P_i, the set P^* will always be distributed among *all* of the agents. Note, however, that some agents might have already joined other coalitions, and these agents might be busy performing the tasks they were assigned. Therefore, it might be more efficient if they did not take part in the re-calculation process. Otherwise, if all the agents were always busy performing the search process as well as the value calculation process, then the members of the formed coalitions might not be able to focus on their assigned tasks long enough to actually perform them on time. On the other hand, by repeating the distribution process, the set P^* would not necessarily be distributed among all of the agents. Instead, it can be distributed among any subset of A. For example, it can be distributed among those that are not members of any coalition.

[18] Tests with different numbers of agents were carried out and gave broadly similar results and so they are not shown here.

[19] This is because having $\bar{n}^* = 29$ means that A^* contains only one agent, which implies that there is only one potential coalition. In other words, no distribution is required in this case.

Each agent a_i should first perform the following once:

- Sort the set of agents based on the agents' UID.
- Set: $\alpha = 1$.
- If (*equal_computational_speeds* = *false*) then:
  1. Calculate t_i, and send it to every other agent.

For every coalition that needs to be formed, each agent a_i should perform the following:

- For every $(s \in S, s \leqslant n)$ do the following:
  1. If $(N_s^* \geqslant n)$ then:
  1.1. If (*equal_computational_speeds*):
    - Calculate the size of your share: $N_{s,i} = \lfloor N_s^*/n \rfloor$
    - Calculate the size of both sub-lists: $N^1_{s,i} = \lfloor N_{s,i} \times 0.4 \rfloor$, $N^2_{s,i} = \lceil N_{s,i} \times 0.6 \rceil$
    - Calculate the number of additional values that need to be calculated: $N' = N_s^* - n \times N_{s,i}$
    - Calculate the index of the last coalition of each sub-list: $index^1_{s,i} = i \times N^1_{s,i}$
      $index^2_{s,i} = N_s^* - N' - ((i-1) \times N^2_{s,i})$

    Otherwise:
    - Calculate the size of every agent's share: $N_{s,j} = \lfloor \frac{N_s^*}{t_j \times \sum_{k=1}^{n} 1/t_k} \rfloor$: $j = 1, \ldots, n$
    - Calculate the size of both sub-lists: $N^1_{s,j} = \lfloor N_{s,j} \times 0.4 \rfloor$, $N^2_{s,j} = \lceil N_{s,j} \times 0.6 \rceil$: $j = 1, \ldots, n$
    - Calculate the number of additional values that need to be calculated: $N' = N_s^* - \sum_{j=1}^{n} N_{s,j}$
    - Calculate the index of the last coalition of each sub-list: $index^1_{s,i} = \sum_{j=1}^{i} N^1_{s,j}$
      $index^2_{s,i} = N_s^* - N' - \sum_{j=1}^{i-1} N^2_{s,j}$

  1.2. Calculate the values of each coalition in your share.
  Otherwise:
  1.3. Calculate the number of additional values that need to be calculated: $N' = N_s^*$
  2. If $(N' > 0)$ then:
  2.1. 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 $(\alpha + N' - 1 \leqslant n)$ then: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_{\alpha+N'-1})$
      else: $A' = (a_\alpha, a_{\alpha+1}, \ldots, a_n, a_1, \ldots, a_{(\alpha+N'-1)-n})$
    - If $(a_i \in A')$ then calculate one of the additional values based on your position in A'
    - If $(\alpha + N' \leqslant n)$ then: $\alpha = \alpha + N'$, else: $\alpha = \alpha + N' - n$

Fig. 8. The DCVC algorithm (final version).

To sum up, Fig. 8 shows the final version of DCVC, including the modifications mentioned in Sections 3.2, 3.3, and 4.2 (henceforth, all references to DCVC refer to this version unless stated otherwise). Here, we assume that the required calculations are distributed among the entire set of agents (A). However, if the distribution is required to be among a subset of A, then two changes need to be made to the algorithm. The first is to replace n with the number of agents that are required to take part in the re-calculation process. The second is to initialize α every time the distribution process is carried out (otherwise, the agents that did not take part in a previous distribution will not be able to know the current value of α).

5. Computational complexity

As discussed earlier, by repeating the entire distribution process, we obtain an optimal distribution of P^*. However, we need to show whether this comes at the expense of an increase in the number of operations required per agent. Therefore, we calculate the computational complexity for each of these methods. Here, we consider the computational complexity to be the number of operations required, given different values of $\bar{n}^*$. Note that instead of using the big-O notation (which only gives an idea of how the number of operations grows with $\bar{n}^*$), we calculate the complexity using equations that give the exact number of operations required.

5.1. Searching through P

Here, each agent a_i searches through P_i in order to find the coalitions that belong to the set P^*. As mentioned earlier, this can be done using two different approaches:

(a) If the agents do not maintain their shares of P^*, then whenever the coalitional values need to be re-calculated, every agent a_i must search through P_i, and find the coalitions that do not contain members of $\bar{A}^*$. In this case, the total number of operations is calculated as follows. For every coalition in P, we calculate the number of comparisons required to determine whether it contains members of $\bar{A}^*$. In more detail, for every size $s \in S$, we have:
- The number of coalitions that require 1 comparison is equivalent to the number of coalitions in which $\bar{a}_1^*$ is a member, and that is: C_{s-1}^{n-1}.
- The number of coalitions that require 2 comparisons is equivalent to the number of coalitions in which $\bar{a}_1^*$ is not a member, and $\bar{a}_2^*$ is a member, and that is: $C_{s-1}^{(n-1)-1} = C_{s-1}^{n-2}$.
- Similarly, for every i $(1 \leqslant i < \bar{n}^*)$, the number of coalitions that require i comparisons is given as follows: C_{s-1}^{n-i}.
- Finally, the number of coalitions that require $\bar{n}^*$ comparisons is equivalent to the number of coalitions in which $\bar{a}_1^*, \ldots, \bar{a}_{\bar{n}^*-1}^*$ are not members, and $\bar{a}_{\bar{n}^*}^*$ is a member (and that is: $C_{s-1}^{n-\bar{n}^*}$), plus the number of coalitions in which $\bar{a}_1^*, \ldots, \bar{a}_{\bar{n}^*}^*$ are not members (and that is: $C_s^{n-\bar{n}^*}$).

Note, however, that there are a number of issues that also need to be considered when calculating the number of comparisons that need to be performed:
- If $\bar{n}^* = 0$ (i.e., if all the agents were able to join other coalitions), then we have $P = P^*$, in which case no search is required.
- If $0 < \bar{n}^* < n$, then the agents would only need to search through the coalitions that are of size: $s \in S, s \leqslant n - \bar{n}^*$; this is because A^* contains only $n - \bar{n}^*$ agents (i.e., the agents in A^* cannot form a coalition that contains more than $n - \bar{n}^*$ members).
- If $\bar{n}^* = n$, then no coalition can be formed, and therefore no search is required.

Based on this, the number of required operations (denoted by $op(\bar{n}^*)$) is given in the following equation:

$$op(\bar{n}^*) = \begin{cases} \sum_{s \in S, s \leqslant n-\bar{n}^*} \left(\left(\sum_{i=1}^{\bar{n}^*} i \times C_{s-1}^{n-i} \right) + (\bar{n}^* \times C_s^{n-\bar{n}^*}) \right) & \text{if } 0 < \bar{n}^* < n \\ 0 & \text{otherwise} \end{cases}$$

Note that the agents search through the coalitions of size: $s \leqslant n - \bar{n}^*$, and perform a maximum of $\bar{n}^*$ operations per coalition. Therefore, given a larger value of $\bar{n}^*$, the search space would be smaller, but the average number of operations per coalition would be larger. Fig. 9 shows the total number of operations required, given different values of $\bar{n}^*$, and that is for the case of 30 agents.

As shown in the figure, as long as $\bar{n}^* < 10$, the number of operations increases given larger values of $\bar{n}^*$. This is because the search space becomes slightly smaller, while the average number of operations required per coalition becomes larger. However, when $\bar{n}^* \geqslant 10$, the number of operations decreases given larger values of $\bar{n}^*$, and that is because the search space becomes significantly smaller.

(b) If each agent maintains its share of P^*, then whenever the coalitional values need to be re-calculated, the agents would have one of the following possible cases:

(i) $\bar{A}_{removed}^* = \phi$, and $\bar{A}_{added}^* \neq \phi$:
Here, we have: $\bar{A}^* = \bar{A}_{prev}^* \cup \bar{A}_{added}^*$. In this case, for every coalition in $temp_i$, agent a_i must determine whether it contains members of $\bar{A}_{added}^*$, and, if not, then the agent must copy it to a new list. Note that $temp_i$ contains the agent's share of P_{prev}^*. Therefore, the total number of operations is calculated as follows. First, for every coalition in P_{prev}^*, we calculate the number of comparisons required to determine whether it contains members of $\bar{A}_{added}^*$. In more detail, for every size $s \in S$, we have:
- The number of coalitions that require 1 comparison is equivalent to the number of coalitions in P_{prev}^* in which $\bar{a}_1^*$ is a member, and that is: $C_{s-1}^{n_{prev}^*-1}$.

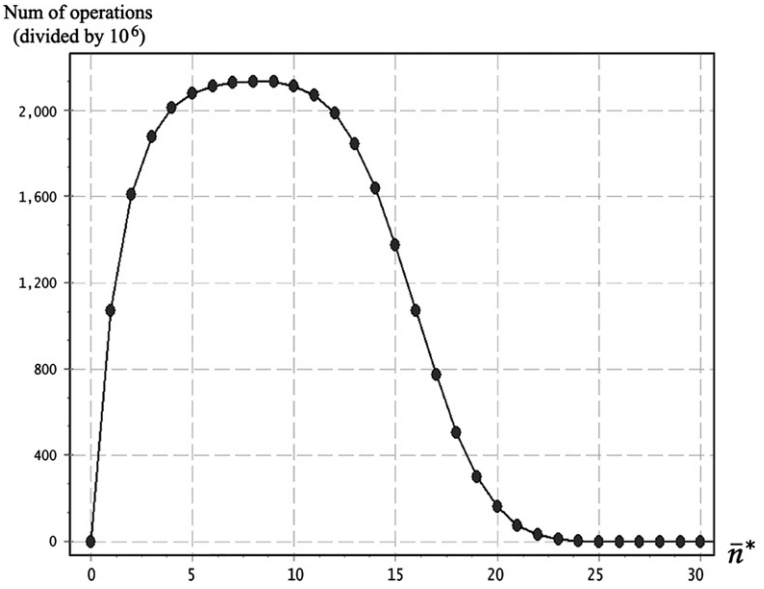

Fig. 9. The total number of operations required for distributing P^*, and that is given 30 agents, where each agent a_i searches through P_i without maintaining its share of P^*.

- The number of coalitions that require i comparisons, where $1 < i < \bar{n}^*_{added}$, is equivalent to the number of coalitions in P^*_{prev} in which $\bar{a}^*_1, \ldots, \bar{a}^*_{i-1}$ are not members, and $\bar{a}^*_i$ is a member (and that is: $C_{s-1}^{n^*_{prev}-i}$).
- The number of coalitions that require $\bar{n}^*_{added}$ comparisons is equivalent to the number of coalitions in P^*_{prev} in which $\bar{a}^*_1, \ldots, \bar{a}^*_{\bar{n}^*_{added}-1}$ are not members, and $\bar{a}^*_{\bar{n}^*_{added}}$ is a member (and that is: $C_{s-1}^{n^*_{prev}-\bar{n}^*_{added}}$), plus the number of coalitions in P^*_{prev} in which $\bar{a}^*_1, \ldots, \bar{a}^*_{\bar{n}^*_{added}}$ are not members (and that is: $C_s^{n^*_{prev}-\bar{n}^*_{added}}$).

Now, we calculate the number of operations required to copy the coalitions that are in P^*_{prev} and do not contain members of $\bar{A}^*_{added}$. Note that every coalition is maintained in memory using n bits, and that the minimum unit of memory that can be allocated is one byte. Therefore, we can say that every coalition is maintained in an array of $\lceil n/8 \rceil$ bytes, and thus, copying a single coalition requires $\lceil n/8 \rceil$ operations.

Finally, since we have: $\bar{A}^* = \bar{A}^*_{prev} \cup \bar{A}^*_{added}$, and since $\bar{A}^*_{added} \neq \phi$, then: $\bar{A}^* \neq \phi$, and this implies that $\bar{n}^* > 0$. Therefore, $\bar{n}^*$ can have one of the following values: $(1, 2, \ldots, n)$, and for any given value of $\bar{n}^*$, the number of required operations is given in the following equation:

$$op(\bar{n}^*) = \begin{cases} 0 & \text{if } \bar{n}^* = n \\ \sum_{s \in S, s \leqslant n-\bar{n}^*} \left(\left(\sum_{i=1}^{\bar{n}^*_{added}} i \times C_{s-1}^{n^*_{prev}-i} \right) + \left((\bar{n}^*_{added} + \lceil n/8 \rceil) \times C_s^{n^*_{prev}-\bar{n}^*_{added}} \right) \right) & \text{if } \bar{n}^* < n \end{cases}$$

Now for the case of 30 agents, Fig. 10 shows the number of operations required, given different values of $\bar{n}^*$. Note that we have: $\bar{n}^* = \bar{n}^*_{prev} + \bar{n}^*_{added}$. Therefore, we calculated $op(\bar{n}^*)$ by taking the average of all possible cases, where $\bar{n}^*_{added} = 1, 2, \ldots, \bar{n}^*$. For example, $op(10)$ was calculated as an average of the cases where $(\bar{n}^*_{prev} = 9, \bar{n}^*_{added} = 1)$, and where $(\bar{n}^*_{prev} = 8, \bar{n}^*_{added} = 2)$, and so on. As shown in the figure, except when $\bar{n}^* < 3$, having each agent maintain its share of P^* requires less operations, especially given large values of $\bar{n}^*$. This is because the number of operations in this case depends mainly on the size of P^*_{prev} (which is often much smaller than P).

(ii) $\bar{A}^*_{removed} = \phi$, and $\bar{A}^*_{added} = \phi$:
In this case, we have: $P^* = P^*_{prev}$. As mentioned earlier, since the agents already have their shares of P^*_{prev} maintained in memory, then no search is required (i.e. the number of required operations is 0):

$$op(\bar{n}^*) = 0$$

(iii) $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} = \phi$:

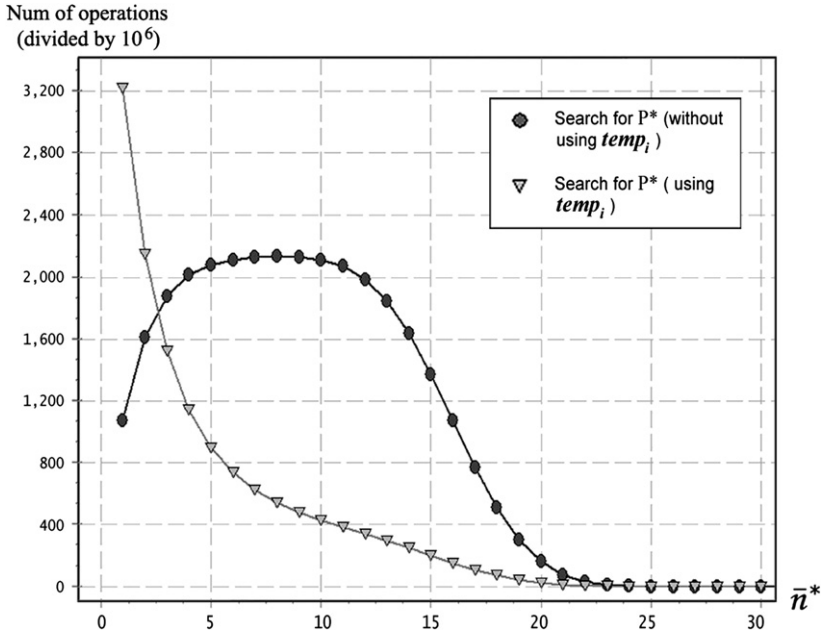

Fig. 10. Given that $\bar{A}^*_{removed} = \phi$, and $\bar{A}^*_{added} \neq \phi$, the figure shows the total number of operations required to distribute P^*, and that is given 30 agents, where each agent maintains its share of P as well as P^*.

As mentioned earlier, finding the coalitions that belong to P^* is done without taking into consideration the fact that each agent maintains its share of P^*_{prev}. Instead, each agent a_i searches through P_i, and copies the coalitions that do not contain members of $\bar{A}^*$ to the list $temp_i$. Based on this, the total number of operations is calculated as follows.[20] First, for every coalition in P, we calculate the number of comparisons required to determine whether it contains members of $\bar{A}^*$. After that, we calculate the number of operations required to copy the coalitions that are in P and do not contain members of $\bar{A}^*$. Note that we have $\bar{A}^* = \bar{A}^*_{prev} \backslash \bar{A}^*_{removed}$, and $\bar{A}^*_{removed} \neq \phi$. This means that $\bar{n}^* < \bar{n}^*_{prev}$, which implies that $\bar{n}^* < n$. Based on this, $\bar{n}^*$ can have one of the following values: $(0, 1, \ldots, n-1)$, and for any given value, the number of operations is given as follows:

$$op(\bar{n}^*) = \begin{cases} \sum_{s \in S,\ s \leqslant n-\bar{n}^*} \left(\left(\sum_{i=1}^{\bar{n}^*} i \times C^{n-i}_{s-1} \right) + \left((\bar{n}^* + \lceil n/8 \rceil) \times C^{n-\bar{n}^*}_{s} \right) \right) & \text{if } 0 < \bar{n}^* \\ 0 & \text{otherwise} \end{cases}$$

Now for the case of 30 agents, Fig. 11 shows the number of operations required, given different values of $\bar{n}^*$. When compared to the case where the agents do not maintain their shares of P^*, we find that unless $\bar{n}^* = 0$, this method would always require more operations.[21] However, as shown in the figure, this difference becomes insignificant when $\bar{n}^* \geqslant 8$.

(iv) $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} \neq \phi$:
In this case, finding the coalitions that belong to P^* is also done without taking into consideration the fact that each agent maintains its share of P^*_{prev}. Therefore, $op(\bar{n}^*)$ is calculated just as in the case where $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} = \phi$. Note, however, that we have: $(\bar{A}^* = \bar{A}^*_{prev} \backslash \bar{A}^*_{removed} \cup \bar{A}^*_{added}$: $\bar{A}^*_{removed} \cap \bar{A}^*_{added} \neq \phi)$, and therefore, having: $\bar{A}^*_{added} \neq \phi$, $\bar{A}^*_{removed} \neq \phi$, implies that: $0 < \bar{n}^* < n$. Based on this, the total number of required operations is given as follows:

$$op(\bar{n}^*) = \sum_{s \in S, s \leqslant n-\bar{n}^*} \left(\left(\sum_{i=1}^{\bar{n}^*} i \times C^{n-i}_{s-1} \right) + \left((\bar{n}^* + \lceil n/8 \rceil) \times C^{n-\bar{n}^*}_{s} \right) \right)$$

[20] Note that the agent must first empty $temp_i$. However, this is done using a single operation, and, therefore, is not considered when calculating the total number of operations required.

[21] These additional operations would be the ones required to copy the coalitions that do not contain members of $\bar{A}^*$.

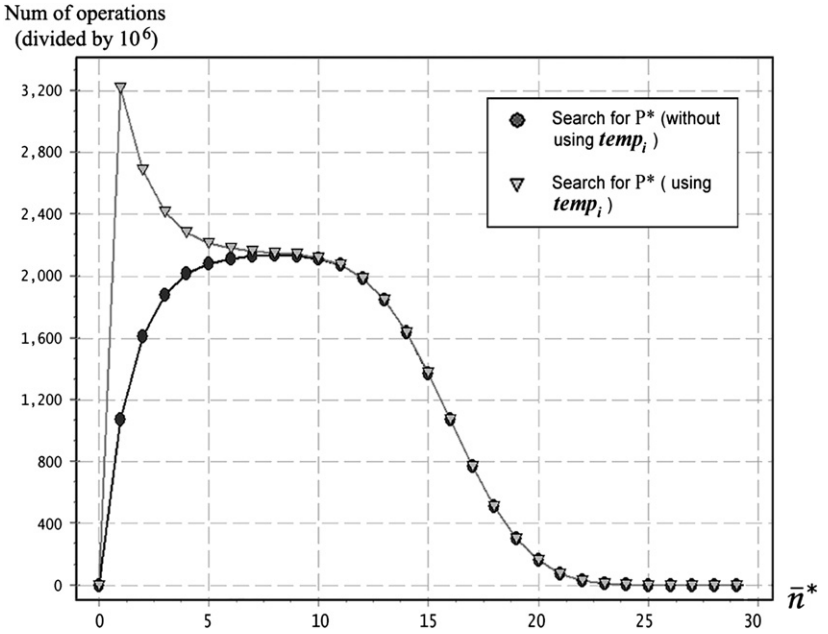

Fig. 11. Given that $\bar{A}^*_{removed} \neq \phi$, and $\bar{A}^*_{added} = \phi$, the figure shows the total number of operations required to distribute P^*, and that is given 30 agents, where each agent maintains its share of P as well as P^*.

5.2. Repeating the entire distribution process

When repeating the distribution process using DCVC, we distinguish between two different cases, based on the memory that is available to the agent:

- In case the agent has sufficient memory space to maintain P_i, then, by maintaining P_i, the agent can avoid repeating the distribution process whenever $\bar{n}^* = 0$, because repeating the distribution process in this case would only result in the same share as the one maintained in memory.[22]
- In case the memory space is not large enough for agent a_i to maintain P_i, then the agent can only repeat the distribution process, even when $\bar{n}^* = 0$.

Next, we calculate the complexity for the second case (the complexity of the first case can then be calculated easily, by replacing the value with 0 whenever $\bar{n}^* = 0$). Specifically, for every size $s \in S$, we calculate the number of operations required to set M from one coalition to the next, throughout the list.[23] As mentioned earlier, changing x values in M requires performing x comparisons, as well as x additions, which gives a total of $(2 \times x)$ operations (see Fig. 5 for an example). Therefore, to calculate the number of operations required, we calculate the number of comparisons, and then multiply this number by 2. In more detail, the number of coalitions that require 1 comparison is equivalent to the number of coalitions in which $a^*_{n^*}$ is a member (i.e., $C^{n^*-1}_{s-1}$), and the number of coalitions that require i comparisons, where $1 \leqslant i \leqslant s$, is equivalent to the number of coalitions in which $a^*_{n^*}, \ldots, a^*_{n^*-(i-1)}$ are not members, and $a^*_{n^*-i}$ is

[22] However, the agent in this case might still need to maintain one coalition (instead of maintaining P_i). This is because the agent, at some point, might need this space to perform other tasks (e.g., the tasks that are assigned to the coalition in which it is a member).

[23] Clearly, these are not the only operations required (e.g. operations are also required to calculate the size of each agent's share, the index at which each share ends, … etc.). However, we will only consider these when calculating the complexity. This is because we are dealing with an exponential number of coalitions, and, therefore, we only consider the operations that are performed for every coalition. In other words, we only count the operations that grow exponentially with the number of agents.

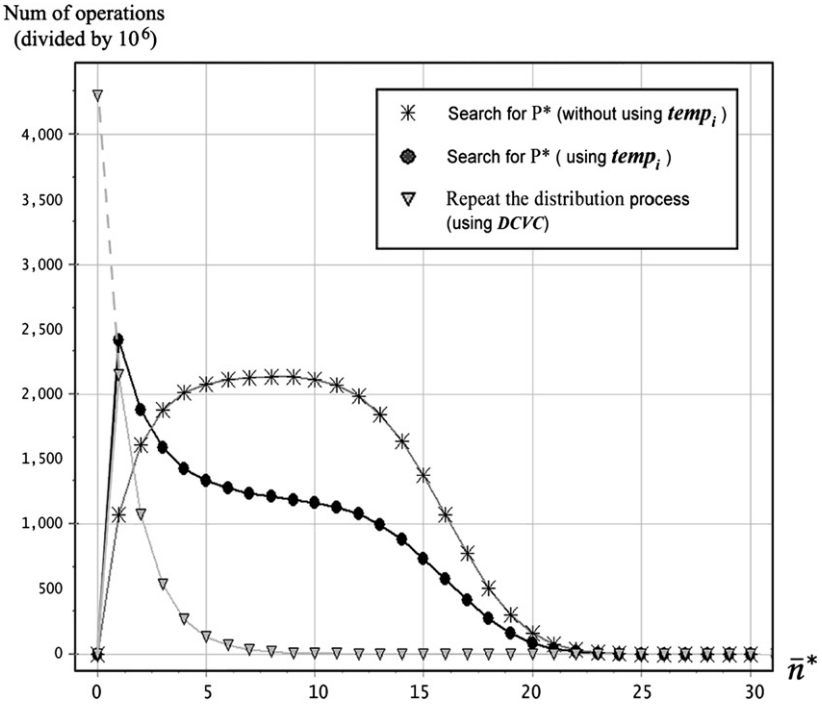

Fig. 12. The number of operations required for distributing P^* given 30 agents, and that is using different distribution methods.

a member (i.e., $C^{n^*-i}_{s-i+1}$). Based on this, the total number of operations required, in order to set M from one coalition to another, is:

$$op(\bar{n}^*) = 2 \times \left(\sum_{s \in S,\ s \leqslant n-\bar{n}^*} \sum_{i=1}^{s} i \times C^{n^*-i}_{s-i+1} \right)$$

We will now compare the number of operations required, given different values of $\bar{n}^*$, and that is when searching through P, and when repeating the distribution process using DCVC. To make this comparison possible, we assume that every agent a_i has sufficient memory space to maintain P_i (otherwise, the agents have no other choice but to repeat the distribution process using DCVC).

Now, given 30 agents, Fig. 12 shows the number of operations required to distribute P^* using any of the methods that were discussed earlier. Note that when repeating the distribution process (using DCVC), the dashed line shows how the number of operations increases when each agent maintains one coalition (instead of maintaining P_i). Also note that when searching through P (using $temp_i$), the figure shows the average number of operations required, and that is for all the different cases of: $\bar{A}^*_{removed}$, $\bar{A}^*_{added}$.

Unlike what we had initially expected, we found that, on average, repeating the entire distribution process (using DCVC) requires fewer operations, and that is even when the agents do not maintain P_i. Specifically, when compared to the case where each agent searches through P_i (without using $temp_i$), we find that if each agent maintains P_i, then, repeating the distribution process requires 13.8% of the operations, otherwise, repeating the distribution process requires 27.6% of the operations. Similarly, when compared to the case where each agent searches through P_i (using $temp_i$), we find that if each agent maintains P_i, then, repeating the distribution process requires 20.3% of the operations, otherwise, it requires 40.7% of the operations.

6. Performance evaluation

Having calculated the computational complexity, we now present empirical results against SK (for the reasons outlined in Section 2). In more detail, Fig. 13 shows how SK operates. Note that the figure only shows the steps that are required for each agent to know its share of calculations (i.e. the figure does not show the steps that are performed

Each agent a_i should perform the following:

- Put in S_i the set of potential coalitions that include up to k agents including a_i.
- While S_i is not empty do:
  - Contact an agent a_j that is a member of a potential coalition in S_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 S_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 P_i.
  - For each agent a_k that has contacted you, subtract from S_i the set S_{ki} of the potential coalitions it had committed to calculate values for.
  - Calculate the values for the coalitions you have committed to (S_{ij}).
  - Repeat contacting other agents until $S_i = a_i$ (i.e., no more agents to contact).

Fig. 13. The SK algorithm.

to calculate the coalitional values themselves), because we are only interested in the distribution process. Also note that Shehory and Kraus assume that the coalitions are only allowed to contain up to k agents. Finally, note that in SK, each agent a_i maintains P_i, and whenever the coalitional values need to be re-calculated, the agent finds the coalitions that belong to P^* by searching through P_i using $temp_i$ (in SK, $temp_i$ is equivalent to L_i^{cr}).

As mentioned earlier in Section 4, without repeating the entire distribution process, P^* would be distributed among all the agents in A. However, by using SK, every coalition in P_i would be a coalition in which a_i is a member, and in this case, P^* would be distributed among the members of A^* instead.[24] Note that the agents in $\bar{A}^*$ might not always be too busy to take part in the re-calculation process. In other words, it would be more efficient if P^* can also be distributed among A whenever necessary (as in DCVC). However, since SK only distributes P^* among A^*, then, when comparing the performance of both DCVC and SK, we set DCVC to distribute P^* only among A^*.

We tested the performance[25] of DCVC and SK given different numbers of agents, ranging from 10 to 25 (however, given any number of agents outside this range, the ratio between the performance of DCVC and SK remains broadly similar). Note that we have 25 agents as a limit, rather than 30 as per the previous sections, because SK requires each agent to maintain a list of all the potential coalitions in which it is a member, and for 30 agents, this list would require more memory space than is actually available to the agent in the simulation. In other words, the agent would not be able to run SK, given $N = 30$.

Given the desiderata mentioned in Section 1, we compare the performance of both algorithms based on the following metrics:

- Distribution time.
- Communication between the agents.
- Redundant calculations performed.
- Memory requirements.
- Equality of the agents' shares.

As for SK, note that all the results (except the memory requirements) were empirically evaluated rather than theoretically proven. This is because they depend heavily on the order in which the agents contact each other, and there are an exponential number of possible contact sequences, which makes the results non-deterministic and not amenable to a theoretical analysis. As for the time requirements, we deliberately chose empirical evaluation based on clock time; this is because the large memory requirements for SK affect the computer's performance speed, and this effect will not appear in a theoretical analysis that takes into account only the number of operations performed.

[24] This is because whenever $a_i \in \bar{A}^*$, every coalition in which a_i is a member can no longer be formed, including all the coalitions in P_i. In other words, even if a_i was able to take part in the re-calculation process, it will not find any coalitions in P_i that belong to P^*.

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

Table 3
The time required (in seconds) for the distribution process

Number of Agents	DCVC	DCVC (maintain P_i)	SK (99% confidence)
10	< 0.01	< 0.01	$0.18 \pm 1\%$
11	< 0.01	< 0.01	$1.11 \pm 1.2\%$
12	< 0.01	< 0.01	$1.54 \pm 0.9\%$
13	< 0.01	< 0.01	$0.16 \pm 0.5\%$
14	< 0.01	< 0.01	$1.73 \pm 1.4\%$
15	< 0.01	< 0.01	$1.83 \pm 1\%$
16	< 0.01	< 0.01	$0.36 \pm 0.5\%$
17	< 0.01	< 0.01	$0.32 \pm 2.2\%$
18	< 0.01	0.01	$0.61 \pm 0.3\%$
19	< 0.01	0.02	$1.68 \pm 2\%$
20	< 0.01	0.04	$2.44 \pm 1.1\%$
21	< 0.01	0.08	$4.81 \pm 1.8\%$
22	< 0.01	0.16	$10.64 \pm 1.9\%$
23	< 0.01	0.33	$21.41 \pm 4.8\%$
24	0.01	0.67	$48.99 \pm 1.9\%$
25	0.02	1.36	$108.72 \pm 3.1\%$

As for DCVC, note that when calculating the distribution time, as well as the memory requirements, we distinguish between the case where each agent maintains one coalition, and the case where each agent maintains P_i. This is because the issue of maintaining P_i affects both the distribution time and the memory requirements.

In our simulation, the agents initially distribute P among themselves, and after that, given different values of $\bar{n}^*$, they distribute P^*. The results presented below are for the case where each agent has the same computational speed, and coalitions of any size are allowed to be formed (which means in our terms $S = \{1, \ldots, n\}$, and in SK's terms: $k = n$).[26] Section 6.1 shows the results for distributing P, while Section 6.2 shows the results for distributing P^*, given different values of $\bar{n}^*$.

6.1. Distributing P

Here, given different numbers of agents (ranging from 10 to 25), we show the results for distributing P among the agents.

6.1.1. Distribution time

The time required to distribute P among the agents is shown in Table 3.[27]

As can be seen, by using DCVC, the agents performed significantly faster, even when each agent maintains its share of P. The reason for this is that when using DCVC, each agent can start processing its share of coalitions immediately, while in SK each agent had to build 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.

Note that in our simulation, the set P^* was distributed among A^*. However, by using DCVC, the set P can also be distributed among A whenever applicable. By having more agents take part in the distribution process, the required time would be even less (for example, given that $n = 25$ and $|A^*| = 10$, distributing P among all the agents would only take 40% of the time required to distribute it among A^*).

[26] Note that if there are any limitations on the coalitional sizes, then P would contain a smaller number of coalitions. However, the ratio between the performance of DCVC and SK remains broadly the same.

[27] Here, we calculated the *standard error of the mean*, as well as the 99% *confidence intervals*. Thus, showing the results in the form: $x \pm y$, means that we are 99% confident that the true *mean* (i.e. average) lies within the range of values: $x - y$ to $x + y$. For more details on how to calculate the standard error of the mean, as well as the confidence intervals, see [1].

Table 4
The total number of bytes that had to be sent between the agents

Number of agents	DCVC	SK (99% confidence)
10	0	$8{,}799 \pm 0\%$
11	0	$20{,}447 \pm 0\%$
12	0	$45{,}076 \pm 0\%$
13	0	$99{,}538 \pm 0\%$
14	0	$217{,}080 \pm 0\%$
15	0	$469{,}173 \pm 0\%$
16	0	$1{,}011{,}217 \pm 0\%$
17	0	$3{,}242{,}544 \pm 0\%$
18	0	$6{,}888{,}787 \pm 0\%$
19	0	$14{,}644{,}832 \pm 0\%$
20	0	$30{,}913{,}264 \pm 0\%$
21	0	$65{,}114{,}817 \pm 0\%$
22	0	$136{,}877{,}925 \pm 0\%$
23	0	$286{,}712{,}976 \pm 0\%$
24	0	$573{,}494{,}824 \pm 0\%$
25	0	$1{,}146{,}989{,}648 \pm 0\%$

6.1.2. Communications between the agents

Table 4 shows the total number of bytes that had to be sent between the agents, in order for each one of them to know its share of the calculations. As shown in the table, SK requires sending an exponentially large number of bytes between the agents; this is mainly because if an agent a_i contacts another agent a_j, and commits to a set of coalitions S_{ij}, 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 communication between the agents because each of them knows its share by using the provided equations, and not by negotiating with other agents.

6.1.3. Redundant calculations performed

Here by redundant we mean having the value of the same coalition calculated by more than one agent, while it was sufficient for only one agent to calculate it. Table 5 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 undertaken with very limited knowledge about the other agent's 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 commit to calculating coalition values without knowing that other agents have already committed to calculating them.

6.1.4. Memory requirements

As mentioned earlier, each coalition is maintained in memory using $\lceil n/8 \rceil$ bytes. Given this, Table 6 shows the number of bytes required per agent to maintain the necessary coalitions.[28] As can be seen, the memory requirements grow exponentially for SK. This is because SK cannot be applied without having each agent start with a list of all the potential coalitions in which it is a member (line 2 in Fig. 13), and the number of such coalitions is (2^{n-1}).[29] In contrast, 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).

[28] Clearly, this is not the only memory space that is required per agent. For example, one could take into consideration the memory required to save the program that actually performs the algorithm, along with all the variables that are required for this program, such as: n, S, ... etc. However, these do not grow exponentially with the number of agents involved, and thus, are considered insignificant.

[29] This is because in the simulation, we assume no limitations on the coalitional sizes. However, if there are limitations, then the number of such coalitions becomes: $\sum_{s \in S} C_{s-1}^{n-1}$.

Table 5
The total number of redundant values that were calculated

Number of agents	DCVC	SK (99% confidence)
10	0	$3{,}381 \pm 0\%$
11	0	$8{,}182 \pm 0\%$
12	0	$18{,}449 \pm 0\%$
13	0	$41{,}584 \pm 0\%$
14	0	$92{,}164 \pm 0\%$
15	0	$201{,}827 \pm 0\%$
16	0	$440{,}081 \pm 0\%$
17	0	$949{,}783 \pm 0\%$
18	0	$2{,}034{,}125 \pm 0\%$
19	0	$4{,}357{,}330 \pm 0\%$
20	0	$9{,}255{,}853 \pm 0\%$
21	0	$19{,}607{,}795 \pm 0\%$
22	0	$41{,}431{,}679 \pm 0\%$
23	0	$87{,}182{,}393 \pm 0\%$
24	0	$182{,}993{,}734 \pm 0\%$
25	0	$383{,}229{,}848 \pm 0\%$

Table 6
The minimum number of bytes required per agent to save the necessary coalitions

Number of agents	DCVC	DCVC (maintain P_i)	SK
10	2	206	1,024
11	2	374	2,048
12	2	684	4,096
13	2	1,262	8,192
14	2	2,342	16,384
15	2	4,370	32,768
16	2	8,192	65,536
17	3	23,133	196,608
18	3	43,692	393,216
19	3	82,785	786,432
20	3	157,287	1,572,864
21	3	299,595	3,145,728
22	3	571,953	6,291,456
23	3	1,094,169	12,582,912
24	3	2,097,153	25,165,824
25	4	5,368,712	67,108,864

As mentioned earlier, given sufficient memory space, each agent using DCVC can also maintain its share of P, and that is to avoid repeating the distribution process whenever $P^* = P$. In this case, the agent would maintain $2^n/n$ coalitions in memory.[30] Note that DCVC would still require allocating a smaller memory space, compared to SK, and that is given any number of agents $n > 2$ (for example, given 25 agents, the memory required by DCVC would only be 8% of that required by SK).

6.1.5. Equality of the agents' shares

Since the agents in our simulation are assumed to have equal computational speeds, then the agents' shares should be as equal as possible. Table 7 shows the difference between the agent that had the biggest share of the calculations and the one that had the smallest. As can be seen, when using DCVC, the maximum difference is only 1, and that is only because the total number of values was not divisible by the given numbers of agents. However with SK, the

[30] Given any limitations on the coalitional sizes, this number becomes: $\sum_{s \in S} C_{s-1}^{n-1}/n$.

Table 7
The difference between the agent that had the biggest share of calculations and the one that had the smallest

Number of agents	DCVC	SK (99% confidence)
10	1	$51 \pm 0.4\%$
11	1	$76 \pm 0.4\%$
12	1	$136 \pm 0.4\%$
13	1	$215 \pm 0.4\%$
14	1	$358 \pm 0.5\%$
15	1	$636 \pm 0.5\%$
16	1	$962 \pm 0.7\%$
17	1	$1{,}537 \pm 1\%$
18	1	$2{,}882 \pm 1.3\%$
19	1	$4{,}441 \pm 1.8\%$
20	1	$7{,}717 \pm 2.8\%$
21	1	$12{,}094 \pm 3.9\%$
22	1	$18{,}243 \pm 6\%$
23	1	$33{,}568 \pm 5.2\%$
24	1	$54{,}544 \pm 5.7\%$
25	1	$85{,}817 \pm 4\%$

difference grows exponentially with the number of agents. This is because the agents' shares are 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.

6.2. Distributing P^*

Here, for the case of 25 agents, we show the results for distributing P^*, given different values of $\bar{n}^*$.

6.2.1. Distribution time

Here, given different values of $\bar{n}^*$, Fig. 14 shows the time required to distribute P^* among A^*. As for DCVC, the dashed line shows how this time increases when each agent maintains one coalition (instead of maintaining P_i). As for SK, the figure shows the average of all the different cases of: $\bar{A}^*_{removed}$ and $\bar{A}^*_{added}$. As shown in the figure, using DCVC requires significantly less time, compared to SK. Specifically, by calculating the average for all the different values of $\bar{n}^*$, we find that if each agent maintains P_i, then DCVC requires 0.4% of the time, otherwise DCVC requires 0.8% of the time.

Note that in Section 5, when calculating the number of operations performed, using each of the methods (i.e., repeating the distribution process using DCVC, and searching through P using $temp_i$ (as in SK)), we found that DCVC requires either 20.3% or 40.7% of the operations (depending on whether the agents maintain P_i). In other words, the difference between both methods was smaller than what we have here. The main reason for this is that when calculating the number of operations required to search through P, we assumed that the agents are searching through exactly $|P|$ coalitions. However, when using SK, the total number of coalitions through which the agents had to search was much larger than $|P|$ (due to the redundancy in the agents' shares). Another reason for this is that when using SK, the agents were dealing with an exponentially large space of memory (while in DCVC, the agents deal with an extremely small space of memory), and this affects the performance speed. As we mentioned earlier, this effect does not appear when theoretically calculating the number of operations required. Moreover, when using SK, the required operations were not distributed equally among the agents, unlike in DCVC (see Section 6.2.5 for details). Note that having unequal shares does not affect the total number of operations performed. However, it does affect the required time.

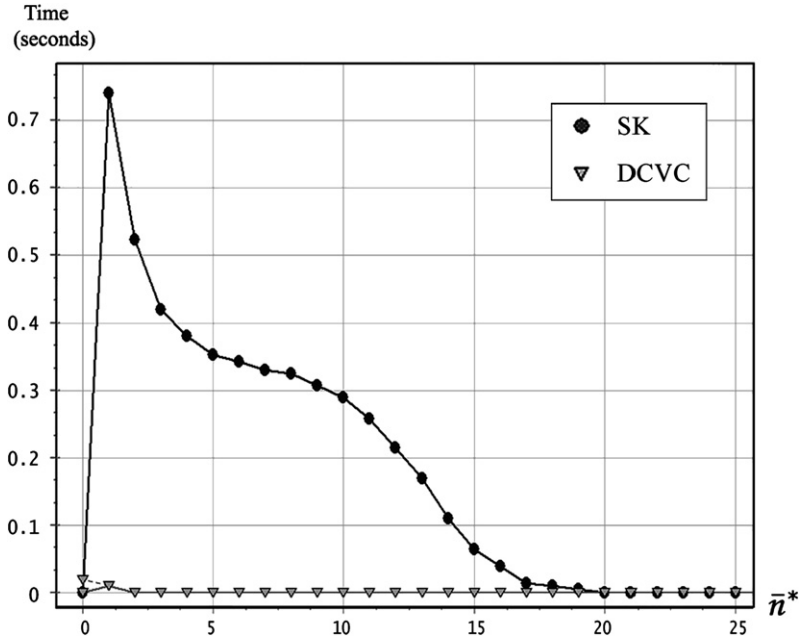

Fig. 14. For the case of 25 agents, the figure shows the time required to distribute P^* among A^*, given different values of $\bar{n}^*$.

6.2.2. Communications between the agents

Here, note that distributing P^*, using any of the two algorithms, is done without any communication between the agents.

6.2.3. Redundant calculations performed

Table 8 shows the total number of redundant values that were calculated, and that is given different values of $\bar{n}^*$.

As shown in the table, 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, when using SK, the number of redundant calculations becomes exponentially large (because the agents' shares of P^* are subsets of their shares of P, and these are not disjoint). Note, however, that for smaller values of $\bar{n}^*$, the redundancy becomes smaller, since P^* becomes smaller.

6.2.4. Memory requirements

Note that by memory requirements, we mean the minimum memory space that must be available to the agent in order for it to perform the distribution algorithm. As for DCVC, distributing P^* (instead of P) does not change the fact that the agent still needs to maintain one coalition at a time (and that the agent might also maintain P_i to avoid repeating the distribution process whenever $P^* = P$). As for SK, when searching for the coalitions that belong to P^*, the agent will not be using a memory space that is sufficient to maintain every potential coalition in which it is a member (as when initially distributing P). However, this does not change the fact that without having this memory space available, the agent will not be able to use SK. Based on this, the memory requirements remain as in Section 6.1.4.

6.2.5. Equality of the agents' shares

Table 9 shows the difference between the agent that had the biggest share of the calculations and the one that had the smallest, and that is given different values of $\bar{n}^*$. As can be seen, when using DCVC, the maximum difference is only 1 (and that is only because the total number of values was not divisible by $\bar{n}^*$). On the other hand, when using SK, the difference becomes much larger. This is because when each agent a_i searches for the coalitions in P_i that belong to P^*, some agents find more coalitions than others, and thus end up with larger shares of P^*. Note that for smaller values of $\bar{n}^*$, the difference between the agents becomes smaller, because P^* becomes smaller.

Table 8
For the case of 25 agents, the total number of redundant values that were calculated, given different values of $\bar{n}^*$

$\bar{n}^*$	DCVC	SK (99% confidence)
0	0	383,229,848 ± 0%
1	0	183,341,930 ± 0%
2	0	87,514,975 ± 0%
3	0	41,671,477 ± 0%
4	0	19,792,110 ± 0%
5	0	9,374,041 ± 0%
6	0	4,424,080 ± 0%
7	0	2,080,256 ± 0%
8	0	974,435 ± 0%
9	0	454,271 ± 0%
10	0	210,717 ± 0%
11	0	97,156 ± 0%
12	0	44,459 ± 0%
13	0	20,093 ± 0%
14	0	8,987 ± 0.1%
15	0	3,960 ± 0.1%
16	0	1,713 ± 0.2%
17	0	725 ± 0.2%
18	0	297 ± 0.4%
19	0	113 ± 0.6%
20	0	40 ± 1.4%
21	0	13 ± 2.5%
22	0	4 ± 4.8%

Table 9
For the case of 25 agents, the table shows the difference between the agent that had the biggest share of the calculations and the one that had the smallest, given different values of $\bar{n}^*$

$\bar{n}^*$	DCVC	SK (99% confidence)
0	1	85,817 ± 4%
1	1	58,485 ± 5.1%
2	1	35,973 ± 4.8%
3	1	20,738 ± 4.4%
4	1	11,580 ± 4.1%
5	1	6,237 ± 3.5%
6	1	3,301 ± 3%
7	1	1,814 ± 3.2%
8	1	936 ± 3.9%
9	1	498 ± 4%
10	1	259 ± 3.9%
11	1	132 ± 3.8%
12	1	67 ± 4.8%
13	1	45 ± 4.9%
14	1	28 ± 5.7%
15	1	19 ± 6.5%
16	1	13 ± 6.7%
17	1	8 ± 6%
18	1	6 ± 6.6%
19	1	5 ± 6.8%
20	1	3 ± 7.4%
21	1	2 ± 9%
22	1	1 ± 11.8%

7. Conclusions and future work

In this paper, we have developed a novel algorithm for distributing the coalitional value calculations among cooperative agents. We have shown how the algorithm can be modified to reflect variations in the agents' computational speed, analyzed the case in which only a subset of agents can form a coalition, calculated the computational complexity of the algorithm, and benchmarked its performance against the only available one in the literature. This comparison showed that our algorithm is significantly faster, requires significantly less memory space, and requires infinitely less communication. These improvements stem from the fact that our algorithm performs no redundant calculations and distributes the calculations equally among the agents.[31] Thus, our algorithm can be seen to represent a significant advance in the state of the art. For future work, we will concentrate on the following:

- Develop an 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.
- Many of the existing coalition formation algorithms involve calculating the values of coalition structures in order to form the most profitable one. We believe this calculation can be distributed using techniques that are similar to those used in the DCVC algorithm. Therefore, we intend to develop an algorithm for distributing the set of possible coalition structures among the agents so that each agent calculates the values of the coalition structures it is assigned. We also need to study the available coalition formation algorithms that search only a subset of the coalition structure graph (e.g., [3,12]); this is necessary since the distribution algorithm should also be efficiently applicable given any subset.
- We would like to relax the assumption of having a fixed number of agents (N) in the system. In particular, we would like to specify how the agents should react to events such as the appearance or disappearance of agents in the agent society. For example, if an agent enters the society during the value calculation process, then the agents should be able to decide whether to restart the whole distribution and calculation process to take into consideration the arrival of the new agent or continue in the ongoing process and have the new agent perform some of the remaining calculations.

Acknowledgements

This article is a significantly revised and extended version of [10]. We are grateful to the participants of AAAI-05, as well as EUMAS-05, for giving us much useful feedback. We would also like to thank Onn Shehory, Gal A. Kaminka, Alex Rogers, and Sarvapali Ramchurn for their helpful comments. Finally, we would like to thank the anonymous reviewers for their help. This research was funded by the DIF-DTC project (8.6) on Agent-Based Control.

References

[1] D.G. Altman, D. Machin, T.N. Bryant, M.J. Gardner, Statistics with Confidence: Confidence Intervals and Statistical Guidelines, BMJ Publishing Group, London, ISBN 0-7279-1375-1, 2000.
[2] J.H. Conway, R.K. Guy, The Book of Numbers, Springer, New York, ISBN 0-387-97993-X, 1996.
[3] V.D. Dang, N.R. Jennings, Generating coalition structures with finite bound from the optimal guarantees, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2004, pp. 564–571.
[4] V.D. Dang, R.K. Dash, A. Rogers, N.R. Jennings, Overlapping coalition formation for efficient data fusion in multi-sensor networks, in: Proceedings of The Twenty First National Conference on Artificial Intelligence (AAAI-06), 2006, pp. 635–640.
[5] J. Kahan, A. Rapoport, Theories of Coalition Formation, Lawrence Erlbaum Associates Publishers, New Jersey, ISBN 0898592984, 1984.
[6] M. Klusch, O. Shehory, A polynomial kernel-oriented coalition formation algorithm for rational information agents, in: Proceedings of International Conference on Multi-Agent Systems (ICMAS-96), 1996, pp. 157–164.
[7] C. Li, K.P. Sycara, Algorithm for combinatorial coalition formation and payoff division in an electronic marketplace, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2002, pp. 120–127.

[31] In cases where there are differences in the agents' computational speeds, the equality refers to the time taken for the calculations, rather than the number of calculations performed.

[8] T.J. Norman, A.D. Preece, S. Chalmers, N.R. Jennings, M. Luck, V.D. Dang, T.D. Nguyen, V. Deora, J. Shao, W.A. Gray, N.J. Fiddian, Agent-based formation of virtual organisations, International Journal of Knowledge Based Systems 17 (2–4) (2004) 103–111.
[9] M.J. Osborne, A. Rubinstein, A Course in Game Theory, The MIT Press, ISBN 0262650401, 1994.
[10] T. Rahwan N.R. Jennings, Distributing coalitional value calculations among cooperating agents, in: Proceedings of The Twentieth National Conference on Artificial Intelligence (AAAI-05), 2005, pp. 152–157.
[11] T.W. Sandholm, V.R. Lesser, Coalitions among computationally bounded agents, Artificial Intelligence 94 (1) (1997) 99–137.
[12] T.W. Sandholm, K. Larson, M. Andersson, O. Shehory, F. Tohmé, Coalition structure generation with worst case guarantees, Artificial Intelligence 111 (1–2) (1999) 209–238.
[13] O. Shehory, S. Kraus, Task allocation via coalition formation among autonomous agents, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), 1995, pp. 655–661.
[14] O. Shehory, S. Kraus, Formation of overlapping coalitions for precedence-ordered task-execution among autonomous agents, in: Proceedings of International Conference on Multi-Agent Systems (ICMAS-96), 1996, pp. 330–337.
[15] O. Shehory, S. Kraus, Methods for task allocation via agent coalition formation, Artificial Intelligence 101 (1–2) (1998) 165–200.
[16] M. Tsvetovat, K.P. Sycara, Y. Chen, J. Ying, Customer coalitions in the electronic marketplace, in: Proceedings of the Fourth International Conference on Autonomous Agents, 2000, pp. 263–264.
[17] G. Zlotkin, J.S. Rosenschein, Coalition, cryptography and stability: Mechanisms for coalition formation in task oriented domains, in: Proceedings of the Twelfth National Conference on Artificial Intelligence, 1994, pp. 432–437.