Distributing Coalition Value Calculations to Self-Interested Agents

(Extended Abstract)

Luke Riley
Dept. of Computer Science
University of Liverpool, UK
L.J.Riley@Liverpool.ac.uk

Terry R. Payne
Dept. of Computer Science
University of Liverpool, UK
T.R.Payne@Liverpool.ac.uk

Trevor Bench-Capon
Dept. of Computer Science
University of Liverpool, UK
tbc@csc.liv.ac.uk

Katie Atkinson
Dept. of Computer Science
University of Liverpool, UK
K.M.Atkinson@Liverpool.ac.uk

ABSTRACT

In characteristic function games, an agent can potentially join many different coalitions, and so must choose which coalition to join. To compare each potential coalition, the agents need to calculate a value for each coalition. As the number of coalitions grows exponentially with the number of agents, the burden of requiring every agent to compute all of the values is high. This can be reduced by sharing the computational load, with each agent calculating only a subset of coalition values. Previous state-of-the-art methods assume that agents are cooperative. This paper outlines an algorithm that distributes value calculations to self-interested agents.

Categories and Subject Descriptors

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

Keywords

Coalition Formation; Distributed Algorithm Design

1. INTRODUCTION

As the set of possible coalitions is exponential on the number of agents, it is advantageous from a computational cost viewpoint to distribute the coalition value calculations in an approximately equal, non-overlapping manner by coordinating the agents without using any communication [1]. This is the approach of the current state-of-the-art method of [1], which presents the distributed coalition value calculation algorithm (DCVC). However that work assumes a *cooperative agent* domain (where agents may not act in their own best interest) and thus DCVC cannot *in its current form* be applied to the *self-interested agent* domain (where agents seek to maximise their own utility).

Our contribution is to address this issue by outlining an alternative method to order and distribute the potential coalitions, giving a distributed coalition value calculation algorithm for *self-interested* agents to use. This algorithm is referred to as SICVC (i.e the *Self-Interested Coalition Value Calculation* Algorithm). The SICVC algorithm: requires no communication between the agents; divides the coalition value calculations in a full and approximately equal manner; divides equally the computational burden of computing which coalition values to calculate; motivates each self-interested agent to compute all of its share; and prevents deceit from occurring later in the coalition formation process by introducing a minimal overlap of coalition value calculations between the agents.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

2. THE PROPOSED SOLUTION

To *motivate* the agents to calculate their coalition values, all the coalitions in the share assigned to agent i by SICVC, denoted CV_i, will *include* agent i. Agent i is motivated to calculate each $C \in CV_i$ because if i was to report $v^r(C) < v^t(C)$, where $v^r(C)$ is the reported value and $v^t(C)$ is the truthful value, then i would not be guaranteed to maximise its utility when completing the coalition formation process using the reported values.

To make sure the agents report $v^r(C) = v^t(C)$ and not $v^r(C) > v^t(C)$, each agent j is assigned an additional minimal share of coalition value calculations *to verify*, denoted DV_j, where the coalitions of DV_j *do not include* j. Each agent j is motivated to stop a deceitful coalition value being reported for any $D \in DV_j$, as j would not be guaranteed to maximise its utility without verifying every $D \in DV_j$ that is reported in the coalition formation process.

The SICVC algorithm makes sure each coalition only occurs in one CV_x and one DV_y by: firstly, like [1], dividing the coalitions into lists L_s where s is the coalition's size ($s \in \{1,...,n\}$); and secondly, unlike [1], dividing each list L_s into 2-dimensional lists $L_{s,t}$ where t is based on the patterns of the integer IDs used to represent the agents of each coalition. Like [1], all agents are given a minimum of $\lfloor \frac{|L_s|}{n} \rfloor$ coalitions to calculate from each list L_s. For lists that do not divide equally by n, then each agent should maintain an α pointer indicating the agent who should calculate the next additional coalition above $n \times \lfloor \frac{|L_s|}{n} \rfloor$, where α is incremented for each further additional coalition (if α goes above n, it will be reset to 1). The same α is used for all lists from size 1 to n so that the maximum difference between the number of coalitions computed by the agents will never be more than one.

The 2-Dimensional Ordering

In every coalition there exists *integer increment values* that separate each agent ID of the coalition from the other agent IDs of the coalition. For example, the integer increment value separating agents 2 and 6 is 4, as the agent ID 2 needs to be incremented by 4 to be equal to agent ID 6. There are many different possible integer increment values when comparing every agent ID of a coalition to each other. Yet if the agents of the coalition are arranged into some fixed array order (i.e. not in the standard set notation of a coalition) named the *coalition array*, then the integer increment values can also be given a fixed order, named the *additional increment array* (AIA). The AIA notes the integers needed to be added to each agent ID of position i of the coalition array, to find the agent ID of position $i+1$ of the coalition array. Analysis of these AIAs has shown that many coalition arrays share the same AIAs. The AIAs, denoted t, are the key to the 2-dimensional representation of each list $L_{s,t}$.

The AIAs make use of integer partitions of y, denoted $\mathscr{I}(y)$. Integer partitions are the sets of positive integers that add up to exactly y, e.g. the integer partitions of 3 are: $\{3\}$, $\{2,1\}$, $\{1,1,1\}$. The integer partitions of $I \in \mathscr{I}(n-s)$ where $|I| \leq s$, are the main component used to generate each t to split each L_s into 2-D lists $L_{s,t}$. The equation $(n-s)$ is used because this gives the number of agents missing from each coalition array. Each I that solves $|I| \leq s$ describes the different ways the missing agent's IDs can be spaced out in the coalitions, meaning each I will be used in at least one t.

Each t of a list $L_{s,t}$ is therefore an array of length s that includes an integer partition of $I \in \mathscr{I}(n-s)$ (if $|I| \leq s$). If the integer partition $I < s$ then additional zeros are used to fill up the empty spaces of t. Where the zeros are placed in the array depends on the length of the integer partition used and the previous AIAs generated.

Each t is based on how much additional increment is needed from the current agent ID (maintained using the value denoted ω) to find the next agent ID of the coalition over the minimum increment of δ = 1, since we assume that agent IDs are unique. Therefore, for an agent i to generate a coalition C assigned to itself using an *additional increment array*, denoted t, the coalition will initially include i (thus $\omega \leftarrow i$) to *motivate* i to compute the coalition's value, then the second agent ID j will equal $\omega + t_0 + \delta$ (then ω is set to: $\omega \leftarrow j$), the third agent ID k will equal $\omega + t_1 + \delta$ (with ω set to: $\omega \leftarrow k$) and so on, until the coalition's size limit has been reached. Note that if the generated agent ID j ever becomes greater than n, then j will be *corrected* by the following equation: $j \leftarrow j - n$.

For example, for a four agent coalition ($s = 4$) in a six agent game ($n = 6$) then an AIA that could be generated is $t = [2,0,0,0]$. The AIA $[2,0,0,0]$ is possible as $\{2\}$ is a possible integer partition of $\mathscr{I}(6-4)$ that satisfies $|\{2\}| \leq 4$, and so the integer in $\{2\}$ is placed in the AIA. As $|\{2\}| \neq 4$, additional zeros are needed to fill up the AIA to make the AIA the required size s. If agent 1 was to use t to find a coalition array, agent 1 would generate $C^1 = [1,4,5,6]$, agent 2 would generate $C^2 = [2,5,6,1]$, agent 3 would generate $C^3 = [3,6,1,2]$, etc.

The idea is that as each agent starts generating the coalition at a different position (which they do, as all the agents start with themselves as the first member), then if all the agents follow the same AIA pattern, the coalitions generated will not overlap as long as the AIAs are only used $\leq n$ times. The AIA is used n times if there is no repeating subsection of t. The AIA is used $< n$ times if there is a repeating subsection of t. In the next section we discuss the number of times to use t in this case (albeit briefly due to space).

		L_3			
		$L_{3,[3,0,0]}$	$L_{3,[2,1,0]}$	$L_{3,[2,0,1]}$	$L_{3,[1,1,1]}$
CV	CV_1	1,5,6	1,4,6	1,4,5	
	CV_2	2,6,1	2,5,1	2,5,6	
	CV_3	3,1,2	3,6,2	3,6,1	
	CV_4	4,2,3	4,1,3	4,1,2	4,6,2
	CV_5	5,3,4	5,2,4	5,2,3	5,1,3
	CV_6	6,4,5	6,3,5	6,3,4	

Table 1: Displays the CV assignments for all 3 agent coalitions in a 6 agent coalition-game, with α initially set to 4.

SICVC Example

SICVC works by distributing coalitions of each size s to the agents to calculate or verify. The following SICVC example is visualised in Table 1 for the CV partitions. If an agent i is assigned $C \in CV_i$, then i is assigned $D \in DV_i$ where $D = N \backslash C$. To generate 3 agent coalitions in CV for a 6 agent coalition-game where $\alpha = 4$ initially, the integer partitions of $\mathscr{I}(6-3)$ will be used. The first integer partition generated will be $I^1 = \{3\}$, I^1 will be placed in an additional increment array (AIA) t^1 and the excess space will be filled with zeros, like so: $t^1 = [3,0,0]$. As there are no repeating subsections of t^1, each agent will know it should use t^1 to generate a coalition in its share. For instance, agent 1 uses t^1 to generate $\{1,5,6\}$, agent 2 generates $\{2,6,1\}$, etc. Once an agent has finished using I^1 to generate its coalition, the next integer partition will be generated (because $|I^1| == 1$), which will be $I^2 = \{2,1\}$. Then I^2 will be placed in an AIA t^2 and the excess space will be filled with zeros, i.e: $t^2 = [2,1,0]$. Again, as there are no repeating subsections of t^2, each agent will use t^2 to generate a coalition in its share. For instance, agent 1 uses t^2 to generate $\{1,4,6\}$, agent 2 generates $\{2,5,1\}$, etc. Now as $|I^2| > 1$, then all possible non-repeating permutations of $t^2_1,...,t^2_{s-1}$ need to be used where the first and last number of the new t are not equal. In this case, there is only one other possible non-repeating permutation, that results in the array: $t^3 = [2,0,1]$. Again, no repeating patterns are found in t^3 so all agents use t^3 to generate coalitions: agent 1 generates $\{1,4,5\}$, agent 2 generates $\{2,5,6\}$, etc.

No more permutations of $t^3_1,...,t^3_{s-1}$ can be developed that have not occurred in a previous AIA. Therefore the last integer partition $I^3 = \{1,1,1\}$ is generated which converts directly into $t^4 = [1,1,1]$. As t^4 has a repeating subsection (t^4_0 is repeated in all the following positions), then the agents need to work out how many coalition values need to be calculated from t^4 and who needs to use t^4. They do this by finding the value denoted λ. For instance, agent 1 finds that the repeating subsection of t^4 has length 1. The coalition agent 1 generates is $\{1,3,5\}$ and so after sorting the coalition into lexicographical order λ becomes: $\lambda = 5 - 3 = 2$. As agent 1 is not one of the next λ agents of the α pointer (initially set $\alpha = 4$), then agent 1 does not have to calculate the coalition's value. Agent 4 on the other hand, uses t^4 to generate the coalition $\{4,6,2\}$ and finds λ to be: $\lambda = 6 - 4 = 2$. As agent 4 is the next agent to calculate an excess coalition value according to the α pointer, agent 4 should calculate the value of the coalition $\{4,6,2\}$. Finally, as all possible permutations of $t^4_1,...,t^4_{s-1}$ find AIAs that have already been used and there are no more unique integer partitions of $\mathscr{I}(6-3)$ left (where $|I| \leq 3$), then SICVC terminates.

3. REFERENCES

[1] T. Rahwan and N. R. Jennings. An algorithm for distributing coalition value calculations among cooperating agents. *Artificial Intelligence*, pages 535–567, 2007.