Near-Optimal Anytime Coalition Structure Generation

Talal Rahwan, Sarvapali D. Ramchurn, Viet Dung Dang, and Nicholas R. Jennings
IAM Group, School of Electronics and Computer Science,
University of Southampton, SO17 1BJ, UK
{tr03r,sdr,vdd,nrj}@ecs.soton.ac.uk

Abstract

Forming effective coalitions is a major research challenge in the field of multi-agent systems. Central to this endeavour is the problem of determining the best set of agents that should participate in a given team. To this end, in this paper, we present a novel, anytime algorithm for coalition structure generation that is faster than previous anytime algorithms designed for this purpose. Our algorithm can generate solutions that either have a tight bound from the optimal or are optimal (depending on the objective) and works by partitioning the space in terms of a small set of elements that represent structures which contain coalitions of particular sizes. It then performs an online heuristic search that prunes the space and only considers valid and non-redundant coalition structures. We empirically show that we are able to find solutions that are, in the worst case, 99% efficient in 0.0043% of the time to find the optimal value by the state of the art dynamic programming (DP) algorithm (for 20 agents), using 66% less memory.

1 Introduction

Coalition formation (CF) is the coming together of distinct, autonomous agents in order to act as a coherent grouping. It has long been studied in cooperative game theory [Osborne and Rubinstein, 1994] and has recently become an important topic in multi-agent systems where a team of agents often need to maximise their individual or their collective efficiency. For example, agents often have to form efficient groups to buy goods in bulk or sensors have to decide on how to group together to monitor a given area [Dang *et al.*, 2006]. Given a set of agents $1, 2, .., i, .., a \in A$, the CF process involves three computationally challenging stages:

1. *Coalition value calculation*: for each subset or coalition $C \subseteq A$, calculate a value $v(C)$ indicating how beneficial that coalition would be if it was formed. Note here that this requires processing 2^a possible coalitions.
2. *Coalition structure generation*: is the equivalent of the complete set partitioning problem [Yeh, 1986]. This means computing the optimal set of coalitions $CS^* = \arg\max_{CS \in \mathcal{CS}} V(CS)$, where a coalition structure $CS \in \mathcal{CS}$ is a partition of A into disjoint exhaustive coalitions, $\mathcal{CS}$ is the set of all such partitions (i.e. each agent belongs to exactly one coalition), and $V(CS) = \sum_{C \in CS} v(C)$. The search space here is $O(a^a)$ and $\omega(a^{\frac{a}{2}})$.
3. *Payment calculation*: compute the transfers between the agents such that they are incentivised to stay in the coalition to which they are assigned. These payments will depend on the stability concept used (e.g. Bargaining set, Kernel, or the Core) and finding these is usually NP-Complete.

In this paper, we focus on the coalition structure generation problem. As in common practice in the literatrue [Sandholm *et al.*, 1999; Dang and Jennings, 2004], we focus on *characteristic function games (CFGs)*, where the value of every coalition C is given using a characteristic function $v(C)$. Up to now, the most widely used algorithm to solve this problem is due to [Yeh, 1986; Rothkopf *et al.*, 1998]. Their algorithm (which runs in $\Theta(3^a)$) is guaranteed to find the optimal solution and is based on dynamic programming (DP). However, the DP approach becomes impractical for agents with limited computational power (e.g. computing the optimal CS for 20 agents requires around 3.4×10^9 operations). Moreover, in the dynamic environments that we consider, agents do not typically have sufficient time to perform such calculations and, in many cases, an approach that gives a good approximation, in a reasonable time, is more valuable.

Against this background, this paper describes a novel anytime search algorithm that uses heuristics to to generate the optimal or near-optimal (with a very tight bound) coalition structure. In more detail, the algorithm works by grouping the coalition structures according to the sizes of coalitions they contain (which we here term a *configuration*). For example, coalition structures $\{\{1\}, \{2, 3\}\}$ and $\{\{3\}, \{1, 2\}\}$ both follow the configuration $\{1, 2\}$. Hence, the space of all coalition structures is partitioned into smaller subsets where every element of a given subset have the same configuration. This is different from previous representations used by other anytime algorithms which looked at the space of interconnected coalition structures [Sandholm *et al.*, 1999; Dang and Jennings, 2004] (which necessitates searching a much bigger portion of the space than our method in order to find only integral worst case guarantees from the optimal solution). Now, using the list of configurations of coalition structures and by estimating the average and upper bound of the solutions that exist within each configuration in this list, we are able to zoom in on the best configurations after searching a relatively minute portion of the search space (typically $3 \times 2^{a-1}$ coalition structures). Moreover, by refining the upper bound of every other configuration after searching the coalition structures of one configuration, we are able to reduce the time to find the optimal configuration still further by discarding those configurations that have a lower upper bound than the best value found so far.

This paper advances the state of the art in the following ways. First, we provide an anytime algorithm to compute the optimal coalition structure that is faster than any previous (anytime) algorithm designed for this purpose. Second, we provide a novel representation for the search space based on coalition structure configurations. This approach permits the selection of a solution based either on the selection of coalition structures of particular configurations or on the time available to find the solution. Third, our algorithm can provide non-integral worst case guarantees on the quality of any computed solution since it can estimate an upper bound for the optimal solution (and improve this estimate as it searches the space). Finally, our algorithm is empirically shown to give solutions which are, at worst, 99% of the optimal value in 0.0043% of the time (in seconds) it takes the DP approach to find the optimal value (for 20 agents).

The rest of the paper is structured as follows. Section 2 describes related work, work and section 3 details the formal model. Section 4 details the algorithm and section 5 empirically evaluates it. Section 6 concludes.

2 Related Work

Yun Yeh's DP algorithm (later rediscovered by Rothkopf et al. for combinatorial auctions) is widely regarded as the fastest algorithm for coalition structure generation and so we use it to benchmark our algorithm. However, for the reasons outlined earlier, a number of researchers have sought to develop heuristics or anytime algorithms. In particular, [Shehory and Kraus, 1998] devised an algorithm to find coalition structures constrained by certain coalition sizes. However, their algorithm does not guarantee to return the optimal value at any point, nor does it provide any means of measuring the efficiency of the coalition structure chosen. Another approach is to assume that only $V(CS)$ is known (as opposed to $v(C)$ in our case). In this case, improving on [Sandholm *et al.*, 1999], Dang and Jennings [2004] devised an algorithm that provides guarantees of the worst case bound from the optimal. In such algorithms, the search space consists of *all* coalition structures (see section 1) even if they are given the values of $v(C)$. Moreover, they only guarantee *integral* bounds up to 3, which means they can guarantee, in the best case, to produce a bound which is only 33% of the optimal value. Furthermore, as opposed to ours, their approach cannot avoid searching the whole space, in all cases, in order to guarantee the optimal solution. Recently, our work [Rahwan and Jennings, 2005] reduced the time taken to cycle through these values without having to maintain the list of coalitions in memory. Here, we build upon this approach to compute coalition structure values as well.

3 Basic Definitions

We now define the basic constructs used by our algorithm. Let $CL_s \in 2^A$ be the list of coalitions of size $s \in \{1, 2, ..., a\}$ such that $\mathcal{CL}$ is the set of coalition lists CL_s (see figure 1 for the configurations for 4 agents). Moreover, let $G_1, G_2, ..., G_{|\mathcal{G}_{CS}|} \in \mathcal{G}_{CS}$ be the set of all possible unique configurations (i.e. there are no two elements with the same coalition sizes). Then, let $F : \mathcal{G}_{CS} \rightarrow 2^{CS}$ be a function that takes a particular configuration and returns all the coalition structures of that configuration. We will denote $\mathcal{N}$ as a list where each element is a *set* of coalition structures of the same configuration. Formally, $\mathcal{N}$ is defined as follows: $\mathcal{N} = \{F(G_1), F(G_2), ..., F(G_{|\mathcal{G}_{CS}|})\}$. Where appropriate we will use $N_1, N_2, ..., N_{|\mathcal{G}_{CS}|} \in \mathcal{N}$ to note every element of $\mathcal{N}$ (see figure 1 for a graphical representation of $\mathcal{N}$ for 4 agents). Finally, the optimal coalition structure is noted as CS^*. In the next section, we further expand on the above notation and describe how our algorithm searches the space.

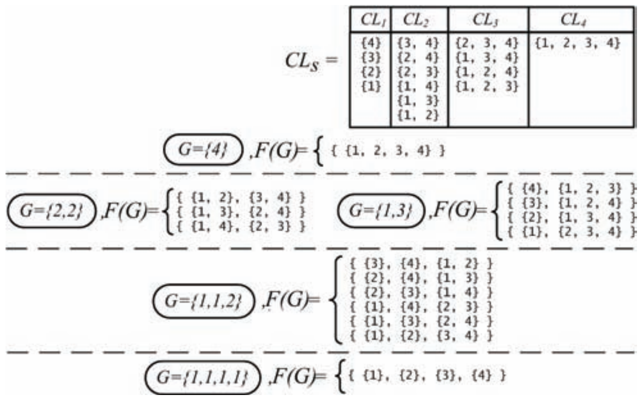
CL_s =
CL_1: {4} {3} {2} {1}
CL_2: {3, 4} {2, 4} {2, 3} {1, 4} {1, 3} {1, 2}
CL_3: {2, 3, 4} {1, 3, 4} {1, 2, 4} {1, 2, 3}
CL_4: {1, 2, 3, 4}
G={4} ,F(G)= { { {1, 2, 3, 4} }
G={2,2} ,F(G)= { { {1, 2}, {3, 4} } { {1, 3}, {2, 4} } { {1, 4}, {2, 3} }
G={1,3} ,F(G)= { { {4}, {1, 2, 3} } { {3}, {1, 2, 4} } { {2}, {1, 3, 4} } { {1}, {2, 3, 4} }
G={1,1,2} ,F(G)= { { {3}, {4}, {1, 2} } { {2}, {4}, {1, 3} } { {2}, {3}, {1, 4} } { {1}, {4}, {2, 3} } { {1}, {3}, {2, 4} } { {1}, {2}, {3, 4} }
G={1,1,1,1} ,F(G)= { { {1}, {2}, {3}, {4} }

Figure 1: $\mathcal{N}$ for 4 agents with the elements $F(G)$ expanded.

4 Anytime Coalition Structure Generation

In this section we describe our algorithm for coalition structure generation. Its input is the set $\mathcal{CL}$ for which $v(C)$ for all $C \in CL_s$ and all $CL_s \in \mathcal{CL}$ is given, as well as the *acceptable bound* $\beta \in [0, 1]$ from the optimal value (if $\beta = 1$, we stop when the optimal solution is found and if $\beta = 0$ any known CS is returned). Given this, our algorithm proceeds in three main stages. The first is a pre-processing stage that involves scanning the input to obtain the maximum and average values of every $CL_s \in \mathcal{CL}$. These values are then used in the second stage which involves generating the list of unique $G \in \mathcal{G}_{CS}$ and selecting the element $F(G)$ to search for CS^*. The third and most computationally costly stage involves determining the values of the $CS \in F(G)$ to find CS^*. We detail each of these stages in the following subsections. Note that, within the first stage, a solution (the best found so far) can be returned and its quality is guaranteed to be $|A|/2$ of the optimal (i.e. 10% in the case of 20 agents).[1] In either of

[1]This bound was proven by [Sandholm *et al.*, 1999] as the procedure in our stage 1 equates to searching the second layer in their

the last two stages the optimal (i.e. $\beta = 1$) or a good enough (i.e. $\beta < 1$) solution may be found and guarantees about the quality of the solution can be given in cases where the second stage completes.

4.1 Stage 1: Pre-processing

We will denote a coalition at index c in the list CL as CL^c where $c \in 1, ..., |CL|$ and an agent in a coalition at index k as $C[k]$. Also, the vectors containing the maximum and average value of each CL_s are noted as max_s and avg_s. The pre-processing stage (detailed in algorithm 1) calculates the maximum and average value of every $CL_s \in \mathcal{CL}$. However, in so doing, it is also possible to search some elements of $\mathcal{N}$. To start with, it is possible to get the value of the CS containing 1 coalition (i.e. the grand coalition $\{1, 2, \ldots, a\}$) from CL_a and the coalitions containing a coalitions (i.e. $\{\{1\}, \{2\}, \ldots, \{a\}\}$ by summing up the values of all coalitions in CL_1. These are depicted as levels 1 and 4 in figure 1). We therefore note the best of these two structures as CS' since this is the most efficient CS found so far.

Then, for each CL_s, we go through every element and record the maximum $max_s = \max_{C \in CL_s}(v(C))$, as well as the average $avg_s = \frac{1}{|CL_s|}\sum_{C \in CL_s} v(C)$. Moreover, it is possible to cycle through CL_s and CL_{a-s} at the same time since a pair of elements from each of these lists forms a coalition structure (see CL_3 and CL_1 in figure 1 and line 18 in algorithm 1). However, in so doing, we must ensure that only pairs of disjoint coalitions (i.e. not containing the same agent) are chosen to form a CS. We ensure this property is upheld by indexing CL_s as per [Rahwan and Jennings, 2005] (see figure 1).[2] This indexing mechanism actually results in each element $C \in CL_s$ at index $c \in [1, ..., s]$ being exactly matched to an element $C' \in CL_{a-s}$ at index $|CL_s| - c + 1$.

Moreover, at the same time, it is possible to compute, the value of all CS that follow a configuration G such that $G = \{1, ..., 1, i\}$ This is achieved by looking into each coalition in the pair when cycling through CL_s and CL_{a-s}. For each such coalition, we retrieve the value of the coalitions of size 1 for each element in the coalition (i.e. the coalition is split into its constituent parts), sum these up and add them to the value of the other coalition and vice versa (see lines 18 and 19 in algorithm 1).

In searching coalition structures in the two ways shown above (i.e. evaluating complementary sizes and coalitions containing 1s), we can actually search a part of the space (which gets smaller as the a increases). This is illustrated by figure 2 for 8 agents where the shaded configurations represent those that will be fully searched after scanning the input. Moreover we will have already covered all coalition structures in case $|A| \leq 4$ after the pre-processing stage. Since the size of the input is of order $O(2^a)$, and the algorithm scans all of it, then it is bound to perform at least this many operations. The operations performed within the two loops (at lines 1 and 10 in algorithm 1) are all linear in time. Hence, the complexity of the above algorithm is $O(2^a)$. Note that we cannot yet guarantee that the best CS found so far (i.e. CS' returned by algorithm 1) is CS^* since we have not yet ensured that all other unsearched configurations (i.e. other than the ones with two elements and those with repeated coalitions of size 1) have a value lower than $V(CS')$. Having obtained the values of the maximum and average of every coalition list, as well as the best possible solution CS' found so far, we next describe how to construct $\mathcal{G}_{\mathcal{CS}}$ and how to choose which element in $\mathcal{N}$ to search.

search space.

[2]In [Rahwan and Jennings, 2005], we use heuristics that guarantee that C as well as CL_s are in ascending order lexicographically.

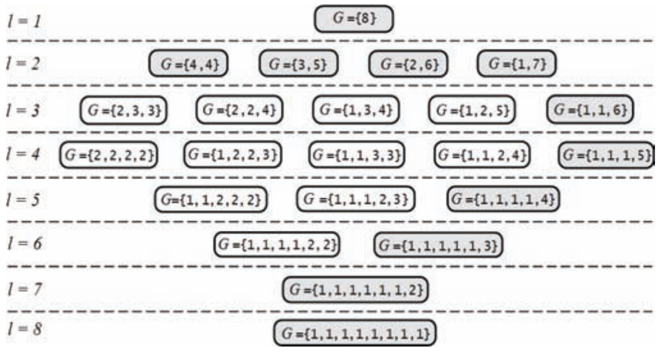

Figure 2: $\mathcal{G}_{\mathcal{CS}}$ with 8 agents where $F(G)$ for the shaded Gs have been searched after scanning the input.

```
Require: $\mathcal{CL}$, $v(C)$ for all $C \in CL$ and all $CL \in \mathcal{CL}$, $CS'$
 1: for $s = 1$ to $\lfloor a/2 \rfloor$ do
 2:   let $s' = a - s$
 3:   Initialise $max_s, avg_s, max_{s'}, avg_{s'}$ to 0, where $max_s, max_{s'} \in$ max
      and $avg_s, avg_{s'} \in$ avg
 4:   Initialise $\nu_{\max}$ to $V(CS')$
 5:   if ($a \mod 2 = 0$) & ($s = a/2$) then
 6:     $c_{end} \leftarrow |CL_s|/2$ % Because the complement of s is actually s
 7:   else
 8:     $c_{end} \leftarrow |CL_s|$
 9:   end if
10:   for $c = 1$ to $c_{end}$ do
11:     let $c' \leftarrow (|CL_s| - c + 1), C \leftarrow CL_s^c$
12:     if $v(C) > max_s$ then
13:       $max_s \leftarrow v(C)$
14:     end if
15:     $sum_s \leftarrow sum_s + v(C)$
16:     Repeat lines 12 to 15, after replacing $C, c, s$ with $C', c', s'$ to calculate
        $max_{s'}, sum_{s'}$
        % Calculate the values of some coalition structures in lines
        17,18, and 19 below.
17:     let $\nu_1 \leftarrow v(C) + v(C')$
18:     let $\nu_2 \leftarrow v(C) + \sum_{k=1}^{s'} v(CL_1^{a-C'[k]})$
19:     let $\nu_3 \leftarrow v(C') + \sum_{k=1}^{s} v(CL_1^{a-C[k]})$
20:     if $\max(\nu_1, \nu_2, \nu_3) > \nu_{\max}$ then
21:       $\nu_{\max} \leftarrow \max(\nu_1, \nu_2, \nu_3)$, $s_{\max} \leftarrow s$, and $c_{\max} \leftarrow c$
22:     end if
23:   end for
24:   $avg_s \leftarrow \frac{sum_s}{|CL_s|}$ and $avg_{s'} \leftarrow \frac{sum_{s'}}{|CL_{s'}|}$
25: end for
26: $s'_{\max} \leftarrow (a - s_{\max}), c'_{\max} \leftarrow (|CL|_{s_{\max}} - c_{\max} + 1)$
27: $C_{\max} \leftarrow (CL_{s_{\max}}^{c_{\max}}), C'_{\max} \leftarrow (CL_{s'_{\max}}^{c'_{\max}})$
28: $CS_1 \leftarrow \{C_{\max}, C'_{\max}\}$
29: $CS_2 \leftarrow \{C_{\max} \cup \bigcup_{i \in C'_{\max}} \{\{i\}\}\}$
30: $CS_3 \leftarrow \{C'_{\max}, \bigcup_{i \in C_{\max}} \{\{i\}\}\}$
31: return max, avg and $CS' = \arg\max_{CS \in \{CS_1, CS_2, CS_3, CS'\}} V(CS)$
```

Algorithm 1: The pre-processing stage.

4.2 Stage 2: Constructing and Choosing Configurations

It is crucial that the algorithm finds the $F(G)$ which contains the CS we seek (which is not necessarily optimal) as quickly as possible (since $\mathcal{N}$ is large). There are two possible ways of doing this. First, we can choose those configurations to search which are most likely to contain the optimal value and, at the same time, allow us to prune the search space in case the optimal is not found. Second, by updating the maximum value of every other G' with more precise information gathered from previously searched G, it is possible to further prune the search space. Before carrying out these operations, however, it is necessary that we first identify all possible configurations in $\mathcal{G}_{\mathcal{CS}}$.

Constructing Unique Configurations in $\mathcal{G}_{\mathcal{CS}}$. The set of all configurations is equal to all the integer partitions that can be generated for the number of agents $|A|$ [Skiena, 1998]. For example, for the number 5, the possible integer partitions or configurations $\mathcal{G}$ is $\{5\}, \{4,1\}, \{3,2\}, \{3,1,1\}, \{2,2,1\}, \{2,1,1,1\}$, and $\{1,1,1,1\}$. The growth rate of the set of configurations is known to be $\Theta(e^{\pi\sqrt{2a/3}}/a)$ (which is much lower than that of the input i.e. $O(2^a)$) where a is the number to be partitioned. Fast recursive algorithms to generate such partitions already exist and we chose a standard algorithm from [Kreher and Stinson, 1999] to do so.

Having obtained all possible configurations and given that we have computed all the maximum and average values of each coalition size in the pre-processing stage, it is now possible to calculate the upper bound (UB_G), and average value (AVG_G) of every G corresponding to $F(G) \in \mathcal{N}$ as well. This can be achieved by cycling through $F(G)$ and summing up the maximum and average values for every coalition size in the configuration. For example, for a configuration $G = \{1,2,4\}$, we obtain $UB_G = max_1 + max_2 + max_4$ and $AVG_G = avg_1 + avg_2 + avg_4$ (which is actually the minimum[3] value that a coalition structure with that configuration can take). Moreover, for those sizes that are repeated, we can use the second best, third best, and so on until the end of the list. For example, for a configuration $\{2,2,1,1,1\}$, we would sum the maximum of CL_2 with the second best of CL_2 and the maximum of CL_1 with the second and third best of CL_1. This procedure gives us a more precise UB for the element, but requires searching the coalition lists for these values (which are not ordered in the input) which for size s is $\frac{a!}{(a-s)!s!}$. The latter grows very quickly with s and since most elements of $\mathcal{G}_{\mathcal{CS}}$ will contain more of the smaller elements (such as 1, 2 or 3), meaning we get more precise updates for most of the elements. We stop at $s = 3$ to limit the cost of searching values lower than max_s.

Having obtained the UB_G and AVG_G of all elements in $\mathcal{N}$, we compare the value $V(CS')$ found so far, with the UB'_G (i.e. excluding previously searched configurations). If $V(CS') > UB_{G'} \times \beta$ for all G', it means we have found a suitable solution. Otherwise, we proceed according to the steps described below.

Choosing a Configuration. Having computed all possible configurations and knowing that it is possible to search each element of $\mathcal{N}$ (using the search procedure described in section 4.3), we now consider the procedure to choose the element of $\mathcal{N}$ to search. First, if it is found that the AVG_G for the element(s) with the highest upper bound (i.e. UB^*_G) is such that $\frac{AVG_G}{UB^*_G} \geq \beta$ we choose the smallest of these elements. Otherwise, we need to choose $F(G)$ that will prune most of the search space and, at the same time, be more likely to contain the required solution. To do this, we need to compute the expected gain (the amount of space that can be avoided after searching the element) and cost in searching each $F(G) \in \mathcal{N}$ (one time). To estimate such parameters, we first need to estimate $\arg\max_{CS \in F(G)}(V(CS))$. This equates to the value lying between UB_G and AVG_G of that element under absolute uncertainty.[4] Then, we need to compute, given estimates of other UBs, the size of the space that will be pruned as a result of the value obtained being greater than the UBs of other elements. This is performed for each $G' \in \mathcal{G}_{\mathcal{CS}}/G$ and the one with the maximum gain is chosen to be searched. The details are outlined in algorithm 2. Next we consider how to improve our estimates of the UB which should allow us to further reduce the space to be searched online.

```
Require: $\mathcal{G}_{\mathcal{CS}}$,$UB_G$ and $AVG_G$ for all $G \in \mathcal{G}_{\mathcal{CS}}$.
 1: if $\mathcal{G}_\beta = \{G | \frac{AVG_G}{UB^*_G} \geq \beta\} \neq \emptyset$ where $UB^*_G = \max_{G \in \mathcal{G}_{\mathcal{CS}}}(UB_G)$ then
    % some configurations satisfy the bound
 2:     $G \rightarrow \arg\min_{G \in \mathcal{G}_\beta}(|F(G)|)$
 3:     return $G$
 4: end if
 5: Initialise $MAX_{gain}$ to 0
 6: for all $G \in \mathcal{G}_{\mathcal{CS}}$ do
 7:     let $E_{\max} = \frac{AVG_G + UB_G}{2}$ % the expected value.
 8:     let $E_{gain} = \left(\sum_{G' \in \mathcal{G}_{\mathcal{CS}}/G, UB_{G'} < E_{\max}} |F(G')|\right) - |F(G)|$ % the
        expected gain in choosing this element.
 9:     if $E_{gain} > MAX_{gain}$ then
10:         $MAX_{gain} \leftarrow E_{gain}$ and $G^* \leftarrow G$
11:     end if
12: end for
13: return $G^*$
```

Algorithm 2: Selecting an element of $\mathcal{G}_{\mathcal{CS}}$ to be searched.

Updating UB. As each element of $\mathcal{N}$ is being searched for CS^* (detailed in section 4.3), it is possible to compute the maximum value of certain combinations of sizes or splits ($split_G \in 2^G$). These can be used then to update the UB of elements of $\mathcal{N}$ in a similar way to the procedure described in section 4.1. For example, while searching an element with configuration $G = \{1,1,1,1,2,3,4\}$, it is possible to find the maximum value of combinations such as $\{1,1,3\}$ or $\{1,2,3,4\}$ which can then be used to update the UB of $\{1,1,1,3,3,4\}$ (and adding $max_1 + max_3 + max_4$) and $\{1,2,3,3,4\}$ (and adding max_3) respectively. If the UB of

[3]This is because every element in CL appears the same number of times in all possible coalition structures of a given configuration. In the case of repeated coalitions of size 1, while the coalitions are not repeated an equal number of times at the same position in the structure, they do repeat equally when considering all positions where they could occur in a coalition structure.

[4]However, as the estimates of UB_G become more precise, the value is more likely to tend towards the upper bound of the element.

that configuration is lowered when using the split, it is updated accordingly.

Since the number of splits can be as large as $2^{|G|}$ (including repeated splits), and computing the maximum value of each of them every time we compute $V(CS)$ can be costly. Therefore, we only compute maximum values for those splits whose value are not known so far. When the search starts, every G is initialised to be split in half. Then as $F(G)$ is searched, the value of the splits to be tracked are computed as $V(CS)$ (see section 4.3). After the search is complete, all $G' \in \mathcal{G}_{CS}/G$ are updated with the newly tracked split. Moreover, all updated G' add one more element from their own configuration to that split to create a new split to be tracked and remove the previous split from the list to be tracked in case it was there. These procedures gradually increase each G's list of splits to be tracked when searched, but also ensure all splits are never computed twice. Next we describe how $F(G)$ is searched and how the value of the coalition structures and the splits are computed.

4.3 Stage 3: Searching an Element of $\mathcal{N}$

This stage is the main potential bottleneck since it could involves searching a space that could possibly equal the whole space of coalition structures (if all $\mathcal{N}$ is searched). However, using the heuristics described in section 4.2, we reduce the possibility of having to search all this space (see section 5 for the impact of this reduction). Moreover, we provide heuristics here that allow us to minimise any redundant computations when cycling through this space. The latter procedure builds upon our previous work [Rahwan and Jennings, 2005] and allows us to construct coalition structures that are unique and fit a particular G, and at the same time, calculate the values of these structures. However, the main problem in constructing coalition structures is that if the procedure is performed naïvely, repetitions will occur and cause the formation of redundant, as well as invalid, coalition structures. Now, there are two main types of repetitions that can occur. First, a structure could contain coalitions that have the same agent(s) in different coalitions (i.e. overlapping coalitions) and this would render the CS invalid. Second, two structures could actually contain the same coalitions. This can happen when the configuration requires multiple coalitions of the same size such that the same coalition appears at different positions (since elements in G are in ascending order). Our procedure, described below, avoids both types.

Let $C_k \in G$ be a coalition located at position $1 \leq k \leq |G|$ of G, and let $\vec{A}_k$ be a vector of agents $\vec{A}_k$ (where $\vec{A}_k^i \leq \vec{A}_k^{i+1}$) from which the members of C_k can be selected. Finally, let $M_k : |M_k| = |C_k|$ be a temporary array that will be used to cycle through all possible instances of C_k. The use of M_k here is mainly to show that we do not need to keep in memory all the coalition structures that can be constructed (although we do need to cycle through all of the coalition structures in N). Instead, we just need to maintain in memory one structure at a time. Since we do not maintain the complete history of computed solutions, our algorithm has a memory requirement of $O(2^a)$ (i.e. the size of the input) as opposed to the DP algorithm's $O(2^{alog_2 3})$ (see [Yeh, 1986; Sandholm *et al.*, 1999]) and hence our algorithm would take only 66% less memory than used by DP.

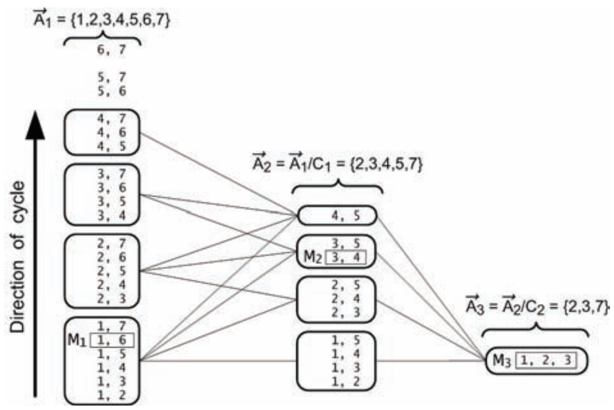

Figure 3: Composing a coalition structure for 7 agents with configuration $G = \{2, 2, 3\}$.

Avoiding Overlapping Coalitions. Basically, for C_1 we use M_1 to cycle through all possible instances of C_1 (i.e., through the set of combinations of size $|C_1|$ taken from the set $\vec{A}_1$) and for *each* coalition in M_1 we use M_2 to cycle through all possible instances of C_2 (i.e., through the coalitions of size $|C_2|$ from the vector $\vec{A}_2 = \vec{A}_1/C_1$), and for each of these possible coalitions, we use M_3 to cycle through all the possible instances of C_3 (i.e., through the coalitions of size $|C_3|$ out of the set: $\vec{A}_3 = \vec{A}_2/C_2$), and so on, for every $k = 1, ..., |G|$. In so doing, we avoid repeating the agents (i.e., we avoid having coalition structures such as: $\{\{1, 2\}, \{2, 3\}\}$) in any coalition fitting the configuration G. Moreover, we use the fast indexing techniques provided by our previous work [Rahwan and Jennings, 2005] to search all instances of C_k.[5]

Avoiding Redundant Coalition Structures. Redundant coalition structures, such as $\{\{1, 2\}, \{3, 4\}\}$ and $\{\{3, 4\}, \{1, 2\}\}$, are only possible when we have repeated coalition sizes in the configuration (e.g $G = \{1, 2, 2, 3\}$ or $G = \{1, 4, 4, 4, 6\}$). Formally, this can only happen when $|C_k| = |C_j|$ where $k \neq j$. Now, since $\vec{A}_k$ is ordered in an ascending fashion (see figure 3), and since we use M_k to cycle through combinations of $\{1, 2, ..., |G|\}$ (which ensures that the combinations are always in an ascending order), we can use the smallest element in the coalition (i.e., C_k^1) as a key to identify the coalitions that we have already gone through. In more detail, as in [Rahwan and Jennings, 2005], M_k will initially go through the coalitions that contain 1 (see bottom of left-most column in figure 3), and this points to the first (smallest) element in $\vec{A}_k$. Since $\vec{A}_{k+1} = \vec{A}_k/C_k$, then the smallest element in C_{k+1} must be greater than the smallest element in C_k (i.e. C_k^1), and this ensures that C_{k+1} can never be equal to a previous instance of C_k (particularly in the case where $|C_{k+1}| = |C_k|$).

After M_k has cycled through the coalitions that contain 1, we move on to coalitions that do not contain 1, but con-

[5]Here, the difference is that $M_k^i \in M_k$ represents an index that points to the required agent in $\vec{A}_i$ (i.e., the required agent is $\vec{A}_i^{M_k^i}$), while in [Rahwan and Jennings, 2005], M_k^i represents the actual agent.

tain 2 (see figure 3 where the arrow moves from elements starting with one and goes up). This means the smallest element in C_k is the second smallest element in $\vec{A}^k$ and, since $\vec{A}_{k+1} = \vec{A}_k / C_k$, the smallest element in C_{k+1} might actually be the smallest element of $\vec{A}_k$. This would, in turn, certainly lead to a repeated coalition structure, because we have already been through the coalition structures that contain two coalitions of size $|C_k|$, where one of them contains $\vec{A}_k^1$, and the other does not. In order to avoid this repetition, M_{k+1} needs to cycle through the combinations starting with 2, instead of those starting with 1 (i.e. see that only the top three boxes in the second column in figure 3 are connected to the second box, from the bottom, in the first column). Thus, when M_k cycles through the combinations that do not contain elements $1, ..., (j-1)$, but contain element j, M_{k+1} should cycle through the combinations starting with j and higher values. This is repeated until we reach the combinations that do not contain $\left|\vec{A}_{k+1}\right| - |C_{k+1}| + 1$, and that is because M_{k+1} cannot contain a combination in which the smallest value is greater than this value. Similarly, if we have more than two coalitions of the same size in a coalition structure (i.e., if we have $C_i = C_{i+1} = ... = C_{i+x}$), then we repeat the same procedure until we reach the combinations that do not contain $\left|\vec{A}_{i+x}\right| - |C_{i+x}| + 1$.

While cycling through every C using M_k, note that we also retrieve $v(C)$ and add these as we go through increasing values of k. In so doing we can also compute the value of splits that need to be tracked for that configuration (as discussed earlier). This can easily be done by keeping track of the maximum value of the coalition sizes dictated by the split. Moreover, if the algorithm finds a coalition structure that has the required value (i.e. either within the required bound or equal to UB_G) then it stops. Finally, if after searching all elements of a particular G, the value of the best coalition structure found so far is higher than the $UB_{G'}$ $\forall G' \in \mathcal{G}_{\mathcal{CS}}/G$, then we guarantee this is the optimal value. We summarise all the procedures described in section 4 in algorithm 3 and show how this leads to computing the optimal (or required) value.

Sketch of Proof of Correctness. The algorithm generates all unique and valid solutions as described above. The algorithm is guaranteed to terminate with a solution since it gradually removes invalid configurations (in step 14), and searches remaining valid ones for the optimal coalition structure, after which it discards these configurations (in step 6) until the set of configurations is empty (as in step 20).

5 Empirical Evaluation

Having described all the parts of our algorithm, we now seek to evaluate its effectiveness by comparing it against the DP approach outlined earlier. In our experiments, the coalition values (i.e. $v(C)$) are randomly selected and scaled by the size of the coalition (similar to the distribution of values used by [Larson and Sandholm, 2000]), thus representing the case where coalitions of bigger sizes have, *on average*, values that are higher than smaller ones. We recorded the time taken by our algorithm to find the re-

```
Require: $\mathcal{CL}$,$v(C) \forall C \in \mathcal{CL}$, $\beta \in [0,1]$
 1: Initialise $\mathcal{N}$, $UB_G$, $splits = \mathcal{CS}/F(\mathcal{G}_{\mathcal{CS}})$, $CS'$,$UB^* = \max_{G \in \mathcal{G}_{\mathcal{CS}}}(UB_G)$ % see algorithm 1 and section 4.1.
 2: return $CS''$ if $\frac{V(CS')}{UB^*} \geq \beta$
 3: set $v_{max} = V(CS')$ % keep track of the maximum value.
 4: repeat
 5:   Select $G$ according to algorithm 2.
 6:   $G_{\mathcal{CS}} \leftarrow G_{\mathcal{CS}}/G$ % discard this element for the next iteration.
 7:   $CS_G = \arg\max_{CS \in F(G)}(V(CS))$ and update all $v(split_{G'})$ and $UB_{G'}$ where $G' \in \mathcal{G}_{\mathcal{CS}}$ % find the coalition structure in this $G$, update the value of splits, and the upper bound of other elements of $\mathcal{G}_{\mathcal{CS}}$-- see section 4.3.
 8:   if $V(CS_G) > v_{max}$ then
 9:     $CS' \leftarrow CS_G$ % set the best coalition to this one.
10:     if $\frac{V(CS')}{UB^*} \geq \beta$ then % if we have reached $\beta$
11:       return $CS''$
12:     else
13:       $v_{max} \leftarrow V(CS_G)$ % set the new lower bound
14:       $\mathcal{G}_{\mathcal{CS}} \leftarrow \{\mathcal{G}_{\mathcal{CS}}/G' | UB(G') < v_{max}, \forall G' \in G_{\mathcal{CS}}\}$
15:       if $\mathcal{G}_{\mathcal{CS}} = \emptyset$ then
16:         return $CS^* \leftarrow CS'$ % the optimal structure.
17:       end if
18:     end if
19:   end if
20: until $\mathcal{G}_{\mathcal{CS}} = \emptyset$
21: return $CS^* \leftarrow CS'$ % the optimal structure.
```

Algorithm 3: The near-optimal anytime coalition structure generation algorithm.

quired value which, in the worst case, lies between 95% and 100% of the optimal (i.e. $\beta \in [0.95, 1]$). By worst case, we mean that the value selected will be, at worst, equal to $\beta \times V(CS^*)$. Thus, it may happen that we find the optimal solution, while only searching for a less efficient one. We choose to benchmark against the DP approach since the other existing anytime algorithms (i.e. [Dang and Jennings, 2004; Sandholm *et al.*, 1999]) only provide integral bounds and always need to search the whole search space to guarantee optimality. Hence, our algorithm is the first to establish a benchmark for anytime heuristic algorithms that generate non-integral bounds and which do so without always needing to search the whole space of solutions.

Here we focus on two experiments to show that our algorithm finds good enough solutions much faster than the DP approach takes to find the optimal and that the time taken to find near-optimal solutions decreases to a lower gradient as the number of agents increases. In order to test this, we recorded the time taken for various degrees of optimality sought (from 0.95 to 1) using our algorithm and compared them to the corresponding DP performance (see figure 4).[6] As can be seen, the time taken by our algorithm in the case where it searches for the optimal (i.e. efficiency 1) increases exponentially faster by a samll constant factor greater than the DP algorithm and even performs better for small numbers of agents (less than 14). Moreover, when the algorithm seeks an efficiency below 1, it finds the solution much faster than the DP algorithm and this difference grows exponentially with increasing numbers of agents (since it searches a much smaller space than 3^a). This implies that the trade-off

[6]We computed the average over 100 runs on a 3GHz Intel Xeon CPU with 2GB of RAM and computed the 95% confidence intervals which are plotted as error bars on the graph. For 16 and more agents, we extrapolated the graph only for the case where $\beta = 1$.

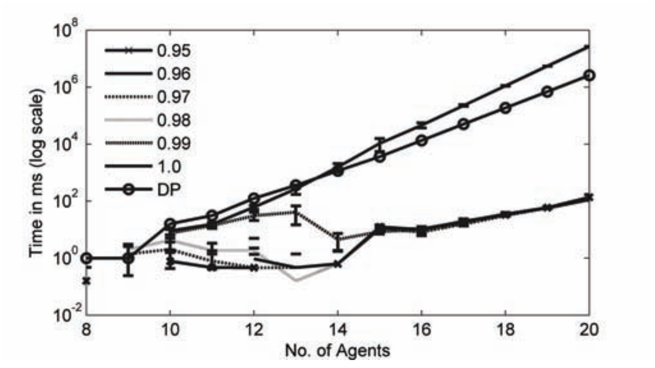

Figure 4: Average time taken for $\beta \in [0.95, 1]$.

between the quality of the solution and the time taken gradually decreases for increasing numbers of agents (112.5ms for $a = 20$,$\beta = 0.99$ v/s 2594942ms for DP i.e. 0.0043% of the time taken by DP). To understand why this happens, we recorded the percentage of the space that is searched in the worst case for increasing numbers of agents.

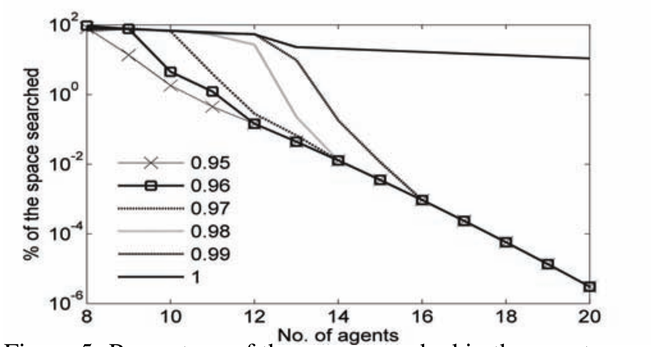

Figure 5: Percentage of the space searched in the worst case.

As figure 5 shows, the percentage of the space searched in the worst case to find the optimal solution as well as for $\beta < 1$, decreases very rapidly as the number of agents increases. This shows that we actually find the solution mostly after scanning the input (see section 4.1) and that has a computational complexity of $O(2^a)$! A possible reason for this result is the distribution of coalition values. Since we use a uniform distribution to generate such values, we expect coalition structure values within each $F(G) \in \mathcal{N}$ to follow a similar distribution (piece-wise linear in most cases) such that the upper bound calculated for each configuration is relatively close to the actual maximum value that can be found within an $F(G)$. Hence, it is possible to find an approximately optimal value fairly quickly by choosing a $F(G)$ with a very high UB_G (which our algorithm does). We will further investigate this result in future work and test this hypothesis by trying out other distributions and using probabilistic techniques.

6 Conclusions

We have shown that it is possible to come up with near-optimal solutions for extremely large search spaces of coalition structures by examining only a minute portion of the space ($3 \times 2^{a-1}$). Our solution relies on a number of techniques to achieve this. First, we use a novel representation of the search space based on configurations of coalition structures. Second, we are able to cycle through the list of coalition structures without creating any redundant or invalid ones. Altogether, these techniques have enabled us to reach a 99% efficient solution in 0.0043% of the time to compute the optimal coalition structure using the DP approach for 20 agents (for larger numbers of agents, this improvement will be exponentially bigger). Moreover, our approach also uses 66% less memory than the DP approach. Future work will look at different coalition value distributions and identifying the theoretical bound established by searching elements of $\mathcal{N}$ using our technique.

Acknowledgements

This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project and is jointly funded by a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership. We also thank the anonymous reviewers for their valuable comments.

References

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

[Dang *et al.*, 2006] V. D. Dang, R. K. Dash, A. Rogers, and N. R. Jennings. Overlapping coalition formation for efficient data fusion in multi-sensor networks. In *21st National Conference on AI (AAAI*, pages 635–640, 2006.

[Kreher and Stinson, 1999] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, Boca Raton, 1999.

[Larson and Sandholm, 2000] K. Larson and T. Sandholm. Anytime coalition structure generation: an average case study. *J. Exp. Theor. Artif. Intell.*, 12(1):23–42, 2000.

[Osborne and Rubinstein, 1994] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge MA, USA, 1994.

[Rahwan and Jennings, 2005] T. Rahwan and N. R. Jennings. Distributing coalitional value calculations among cooperative agents. In *AAAI*, pages 152–159, 2005.

[Rothkopf *et al.*, 1998] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 1998.

[Sandholm *et al.*, 1999] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé. Coalition structure generation with worst case guarantees. *Artif. Intelligence*, 111(1-2):209–238, 1999.

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

[Skiena, 1998] S. S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, New York, 1998.

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