Anytime Optimal Coalition Structure Generation

Talal Rahwan[1] and **Sarvapali D. Ramchurn**[1] **Viet D. Dang**[3] **Andrea Giovannucci**[2] **Nicholas R. Jennings**[1]
[1]School of Electronics and Computer Science, University of Southampton, Southampton, UK
{tr,sdr,nrj}@ecs.soton.ac.uk
[2]Institute of Artificial Intelligence Research, CSIC, Bellaterra, Spain
andrea@iiia.csic.es
[3]Lost Wax Media Ltd, Richmond, Surrey, TW9 2JY, UK
v.d.dang@lostwax.com

Abstract

A key problem when forming effective coalitions of autonomous agents is determining the best groupings, or the optimal coalition structure, to select to achieve some goal. To this end, we present a novel, anytime algorithm for this task that is significantly faster than current solutions. Specifically, we empirically show that we are able to find solutions that are optimal in 0.082% of the time taken by the state of the art dynamic programming algorithm (for 27 agents), using much less memory ($O(2^n)$ instead of $O(3^n)$ for n agents). Moreover, our algorithm is the first to be able to find solutions for more than 17 agents in reasonable time (less than 90 minutes for 27 agents, as opposed to around 2 months for the best previous solution).

Introduction

Coalition formation (CF) is a fundamental form of interaction that allows the creation of coherent groupings of distinct, autonomous agents in order to efficiently achieve their individual or collective goals. Now, in many cases the environment within which the agents operate is dynamic and time dependent in that the gains from achieving the goals decrease the further into the future they are met. Moreover, such agents are usually resource limited and so it is critical that the CF process should be as fast as possible and have minimal memory and computational requirements. Furthermore, in cases where the search for the best solution may be lengthy, it is important that solutions are available at any time and that they get incrementally better as time passes.

One of the main bottlenecks in the CF process is that of coalition structure generation (CSG). This involves selecting the best coalitions from the set of all possible coalitions such that each agent joins exactly one coalition. The search space generated for n agents grows exponentially in $O(n^n)$ and $\omega(n^{\frac{n}{2}})$ (Sandholm *et al.* 1999). Moreover, the CSG problem has been shown to be NP-complete and existing algorithms cannot generate solutions within a reasonable time for even moderate numbers of agents (typically more than 17).

Previous work has adopted three principal approaches to solving this problem. First, some anytime algorithms have been devised, but the worst case bounds they provide are usually very low (up to 50% of the optimal in the best case) and they always have to search the whole space to guarantee an optimal solution (Sandholm *et al.* 1999; Dang & Jennings 2004). Anytime algorithms based on Integer programming do manage to prune the space searched by employing linear programming relaxations of the problem, but these quickly run out of memory with small numbers of agents (typically around 17). Second, faster dynamic programming (DP) solutions have been developed that avoid searching the whole space to guarantee an optimal solution, but they cannot generate solutions anytime (Yun Yeh 1986, Rothkopf *et al.* 1998). Moreover, the DP approach quickly becomes impractical for moderate numbers of agents (e.g., about 2 months for 27 agents on a typical modern PC). Third, yet other approaches have tried to limit the size of coalitions that can be formed in an attempt to reduce the complexity of the problem (Shehory & Kraus 1998). However, such constraints can greatly reduce the efficiency of the possible solutions since the solution chosen in this case can be arbitrarily far from the optimal (e.g., if the coalitions of the chosen size have a value of zero). Recently, however, Rahwan *et al.* (2007) proposed an anytime CSG algorithm that could generate near-optimal solutions (i.e. $\geq 95\%$ efficient) faster than DP. However, their algorithm is still slower than DP in finding the optimal solution, provides no theoretical bounds, and was only tested on uniform distributions of coalition values.

Against this background, we develop and evaluate a novel CSG algorithm that is anytime and that finds optimal solutions much faster than any previous algorithm designed for this purpose. The strength of our approach is founded upon three main components. First, we build upon (Rahwan *et al.* 2007) to develop a more precise representation of the search space and prove the bounds that can be computed for disjoint parts of the search space. Second, we propose new search strategies to select the most promising subspaces and allow agents to trade-off solution quality against their computational capability. Also, our search strategies allow us to provide worst case bounds on the quality of the solutions generated. Third, we develop a branch-and-bound algorithm to find the optimal coalition structure without having to gather the values of all coalitions within each possible coalition structure.

This paper advances the state of the art in the following ways. First, we develop a novel anytime optimal coali-

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

tion structure generation algorithm that is significantly faster than any previous algorithm designed for this purpose (e.g., it takes less than 90 minutes (on average) to solve the 27 agents CSG problem, as opposed to more than 2 months for the state of the art DP approach). Second, we improve upon Rahwan *et al.*'s representation of the search space and describe its algebraic properties. Using this representation, which partitions the search space into smaller sub-spaces, we propose new search strategies that allow agents to select to search certain sub-spaces according to their constraints and goals. For instance, we can choose, on the fly, whether coalition structures with larger or smaller coalitions should be selected (since each sub-space contains coalition structures composed of coalitions of pre-determined sizes). Also, we can analyse the trade-off between the size (i.e. the number of coalition structures within) of a sub-space and the improvement it may bring to the actual solution by virtue of its lower bound. Hence, rather than constraining the solution to fixed sizes, as per the third strand of previous work discussed above, agents using our representation can make an informed decision about the sizes of coalitions to choose. Third, we prove how bounds on sub-spaces can be obtained and show how these allow us to reach the optimal solution faster than other anytime approaches. Finally, our algorithm can provide very high worst case guarantees on the quality of any computed solution very quickly (e.g, more than 95% of the optimal within 0.5 seconds in the worst case for 21 agents) since it can rapidly (after scanning the input) tighten the upper and lower bounds of the optimal solution.

The rest of this paper is structured as follows. First, we present the basic definitions of the CSG problem and describe our novel representation for the search space. Then, we detail our CSG algorithm and show how we use this representation to prune the search space and find the optimal coalition structure using a branch-and-bound technique. Finally, we provide an empirical evaluation of the algorithm, before concluding.

Basic Definitions

The CSG problem is equivalent to a *complete set partitioning problem* (Yun Yeh 1986) where *all* disjoint subsets (i.e. coalitions) of the set of agents need to be evaluated in order to find the optimal partition (i.e. a coalition structure). The weight associated to a subset is given by the value of the coalition. More formally, we denote the set of agents as $A = \{a_1, a_2, \dots, a_i, \dots, a_n\}$, where n is the number of agents, and a subset (or coalition) as $C \subseteq A$. Each coalition has a value given by a characteristic function $v(C) \in \mathbb{R}^+ \cup \{0\}$. A partition, or coalition structure, is noted as $CS \subseteq 2^A$ and the set of all partitions of A is noted as $\mathcal{P}(A) = \{CS \in 2^A | \cup_{C \in CS} C = A \wedge \forall C, C' \in CS\ C \cap C' = \emptyset\}$ (i.e. the set of unique disjoint coalition structures). For example, if $A = \{a_1, a_2, a_3, a_4\}$, possible coalition structures are $\{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}$ or $\{\{a_1, a_3\}, \{a_2, a_4\}\}$. The value of a coalition structure is then given by the function $V(CS) = \sum_{C \in CS} v(C)$. Our goal is to find the optimal coalition structure, noted as $CS^* = \arg\max_{CS \in \mathcal{P}(A)}(V(CS))$, given $v(C), \forall C \subseteq A$.

Search Space Representation

The search space representation employed by most existing state of the art anytime algorithms is an undirected graph in which the vertices represent coalition structures (Sandholm *et al.* 1999; Dang & Jennings 2004). However, this representation forces all valid solutions to be explored to guarantee that the optimal has been found.[1] In contrast, the DP approach employs a more efficient representation, where precomputed components of the final solution are reused, which allows it to prune the space (Yun Yeh 1986). However, the DP approach does not select incrementally better solutions, which makes it unsuitable when there is insufficient time to wait for an optimal solution. Given the above, we believe an ideal representation for the search space should allow the computation of solutions anytime, while establishing bounds on their quality and permit the pruning of the space to speed up the search. With this objective in mind, in this section we describe just such a representation. In particular, it supports an efficient search for the following reasons. First, it partitions the space into smaller independent sub-spaces, for which we can identify upper and lower bounds, which allow us to compute a bound on solutions found during the search. Second, we can prune most of these sub-spaces since we can identify the ones that cannot contain a solution better than the one found so far (hence we can produce solutions anytime). Third, since the representation pre-determines the size of coalitions present in each sub-space, agents can balance their preference for certain coalition sizes against the cost of computing the solution for these sub-spaces. Note that our approach builds upon and improves that of (Rahwan *et al.* 2007) by redefining the representation of the search space formally and describing its algebraic properties. Also, we additionally prove how valid upper and lower bounds can be computed for such sub-spaces. Later on, we also describe worst case bounds on the quality of the solution that our representation allows us to generate.

Partitioning the Search Space

We partition the search space $\mathcal{P}(A)$ by defining sub-spaces that contain coalition structures that are similar according to some criterion. The particular criterion we specify here is based on the integer partitions of the number of agents $|A|$.[2] These integer partitions, of an integer n, are the sets of nonzero integers that add up to exactly n (Skiena 1998). For example, the five distinct partitions of the number 4 are $\{4\}, \{3, 1\}, \{2, 2\}, \{2, 1, 1\}$, and $\{1, 1, 1, 1\}$. It can easily be shown that the different ways in which a set of 4 elements can be partitioned can be directly mapped to the integer partitions of the num-

[1] Within such a representation it is comparatively easy to find *integral* worst case bounds from the optimal. Such bounds can be useful in giving an indication as to the quality of the solution found by searching parts of the space (rather than obtaining a solution that could be infinitely far from the optimal).

[2] Other criteria could be developed to further partition the space into smaller sub-spaces, but the one we develop here allows us to choose coalition structures with certain properties as we show later.

ber 4. For instance, partitions (or coalition structures) of the set of four agents, $\{\{a_1, a_2\}, \{a_3\}, \{a_4\}\} \in \mathcal{P}(A)$, and $\{\{a_4, a_1\}, \{a_2\}, \{a_3\}\} \in \mathcal{P}(A)$ are associated with the integer partition $\{2, 1, 1\}$, where the parts (or elements) of the integer partition correspond to the cardinality of the elements (i.e. the size of the coalitions) of the set partition (i.e. the coalition structure). For example, for the coalition structure $\{\{a_4, a_1\}, \{a_2\}, \{a_3\}\} \in \mathcal{P}(A)$, the elements of its configuration can be obtained as follows: $|\{a_4, a_1\}| = 2, |\{a_2\}| = 1$, and $|\{a_3\}| = 1$.

Here we precisely define this mapping by the function $F : \mathcal{P}(A) \rightarrow \mathcal{G}$, where $\mathcal{G}$ is the set of integer partitions of $|A|$. Thus, F defines an equivalence relation $\sim$ on $\mathcal{P}(A)$ such that $CS \sim CS'$ iff $F(CS) = F(CS')$ (i.e. the cardinality of the elements of CS are the same as those of CS'). Given this, in the rest of the paper we will refer to an integer partition as a coalition structure *configuration*. Then, the pre-image[3] of a configuration G, noted as $F^{-1}[\{G\}]$, contains all coalition structures with the same configuration G.

Selecting Bounds for the Sub-Spaces

For each sub-space $F^{-1}[\{G\}]$, it is possible to compute an upper and a lower bound. To this end, we define the set of coalitions of the same size (or coalition lists) as $CL_s = \{C \subseteq A | \ |C| = s\}$, where $s \in \{1, \ldots, |A|\}$. We note as max_s, min_s, and avg_s, the maximum, minimum, and average value of coalitions of a given size s. Now, given a configuration G, we define a set $S_G = \prod_{s \in G} CL_s^{G(s)}$ which is the cartesian product of the coalition lists CL_s, where $G(s)$ returns the multiplicity of s in G. We will note as $g_n \in G$ (instead of s) the elements of G where the index of the element matters. Notice that the set S_G contains many combinations of coalitions that are invalid coalition structures since they may overlap with each other (i.e., two coalition structures may contain coalitions which contain the same agents). For example, a coalition structure of $\{\{a_1, a_2\}, \{a_1\}, \{a_3\}\}$ is invalid since agent a_1 appears in two coalitions.

Now, consider the value UB_G obtained by summing the maximum value of each coalition list involved in a set S_G; that is, $UB_G = \sum_{s \in G} G(s) \cdot max_s$. As UB_G defines the maximum value of the elements in S_G, it is easy to demonstrate that UB_G is an upper bound for the coalition structures contained in $F^{-1}[\{G\}]$ which contains only valid coalition structures. In a similar way, it is possible to compute a lower bound. Intuitively, one would expect to select $\sum_{s \in G} G(s) \cdot min_s$ (i.e., the minimum value of an element in S_G) as a lower bound for all coalition structures (including invalid overlapping ones). However, a better lower bound for the optimal value is actually the *average* of the coalition structures in $F^{-1}[\{G\}]$ since there is bound to be a coalition structure that has a higher value than the average value and this average value is very likely to be much greater than the minimum one (depending on the distribution of values). The key point to note is that this average can be obtained without having to go through *any* coalition structure (as noted, but not proved, by (Rahwan *et al.* 2007)). This is because the average value of a sub-space can be proven (see appendix) to be the sum of the averages of the coalition lists (i.e., $AVG_G = \sum_{g_i \in G} avg_i$) and these can be computed with very little cost by only scanning the input which is much smaller than the space of coalition structures.

Theorem 1 *Let G be a configuration, $G = \{g_1, \ldots, g_i, ..., g_{|G|}\}$. Let AVG_G be the average of all coalition structures in $F^{-1}[\{G\}]$ and avg_{g_i} be the average of all coalitions in CL_{g_i}, for every $1 \leq i \leq |G|$. Then the following holds:*

$$AVG_G = \sum_{g_i \in G} avg_{g_i}$$

The Anytime Optimal Algorithm

Having described our representation, we now focus on how to search it. Here, we describe the two main steps of the algorithm. First, we obtain the bounds UB_G and AVG_G for every $F^{-1}[\{G\}]$. Within the same process, we are able to compute an optimal solution for some sub-spaces (at a very small cost). Moreover, we can establish a worst case bound on the quality of the solution found so far and prune parts of the space after this step. Second, we describe how the algorithm searches the elements of $F^{-1}[\{G\}]$ using a branch-and-bound technique to reduce the number of coalition structures that we need to go through, while further pruning the space.

Step 1: Computing Bounds

The input to the coalition structure generation problem is the value associated to each coalition (i.e. $v(C)$ for all $C \in 2^A$) provided in ordered lists as in (Rahwan *et al.* 2007).

Now, let $\mathcal{G}^2 = \{G \in \mathcal{G} | \ |G| = 2\}$ denote the set of configurations where the number of elements in a given G is equal to 2. Given this, we can compute the value of all coalition structures with configurations $G \in \mathcal{G}^2$ by simply summing the values of the coalitions while scanning the lists CL_s and $CL_{|A|-s}$, starting at different extremities for each list. The details of how to do this are in (Rahwan *et al.* 2007). In so doing, we can also record max_s and avg_s (and $max_{|A|-s}$ and $avg_{|A|-s}$). Note that this process is $O(m)$ where $m = 2^{|A|} - 1$ is the size of the input.

Having computed max_s and avg_s for each sub-space we can now compute their upper (UB_G) and lower bounds (AVG_G). Then, we assign the lower bound of the optimal to $LB = max(AVG_G^*, V(CS'))$, where $AVG_G^* = \arg\max_{G \in \mathcal{G}}(AVG_G)$ is the highest lower bound of the sub-spaces and $V(CS')$ is the value of the best coalition structure CS' obtained by scanning the input as above. Hence, all sub-spaces with $UB_G < LB$ can be pruned straightaway. Note that after scanning the input (i.e. searching sub-spaces corresponding to configurations in $\mathcal{G}^2$), we can establish a worst case bound equal to $|A|/2$ on the value found so far according to (Sandholm *et al.* 1999). However, unlike (Rahwan *et al.* 2007) who do not bound their solution, we can also specify a worst case bound equal to $\frac{UB^{max}}{AVG^*_{\mathcal{G}^2}}$, where $AVG^*_{\mathcal{G}^2} = \arg\max_{G \in \mathcal{G}^2}(AVG_G)$ and

[3] Recall that the pre-image or inverse image of a set $G \subseteq \mathcal{G}$ under $F : \mathcal{P}(A) \rightarrow \mathcal{G}$ is the subset of $\mathcal{P}(A)$ defined by $F^{-1}[G] = \{CS \in \mathcal{P}(A) | F(CS) = G\}$.

$UB^{max} = \max_{G \in \mathcal{G}}(UB_G)$. This is because the value of LB is at worst equal to $V(CS')$ which, in turn, can be no worse than the average of coalition structures with configurations $G \in \mathcal{G}^2$, that is $AVG^*_{\mathcal{G}^2}$ as obtained above. Thus, while Sandholm *et al.*'s bound is integral and is, at best 2 (i.e. the solution is guaranteed to be 50% of the optimal), after scanning the input ours may not be an integer and could be as low as 1 (i.e. the solution is guaranteed to be 100% of the optimal). Hence, the best of the two measures can be taken as the worst case bound of the solution. Next, we describe how we search through the remaining sub-spaces.

Step 2: Selecting and Searching $F^{-1}[\{G\}]$

Given a set of promising sub-spaces obtained after scanning the input, we need to select the one sub-space $F^{-1}[\{G\}]$ to search and then search for the best coalition structure within it (i.e., CS^*_G). These operations are performed repeatedly, one after the other, until either of the following termination conditions are reached, at which point the optimal solution is obtained (i.e., $CS^* = CS^*_G$):

1. If $V(CS^*_G) = UB^{max}$, this coalition structure is the best that can ever be found.
2. If all nodes have been searched or the remaining sub-spaces have been pruned.

As can be seen, unlike previous CSG algorithms, the speed with which our algorithm reaches the optimal value depends on the closeness of the upper bound to the optimal value. This closeness is determined by the spread of the distribution of the coalition values (e.g., a larger variance means that the upper bound is more representative of the maximum and conversely for a tighter variance). Hence, later in the paper we will evaluate the robustness of our algorithm against a number of distributions. In the next subsection we describe how we select the next sub-space to search.

Selecting $F^{-1}[\{G\}]$. From the previous subsection, we know the optimal solution can only be guaranteed if, either there are no sub-spaces left to search or the maximum upper bound has been reached. Either condition can only be reached if the sub-space with the highest UB_G (i.e. UB^{max}) is searched. A key point to note is that these conditions mean that the cost of searching a sub-space to find the CS^* is equal to the cost of search that sub-space and all sub-spaces with a higher upper bound. Hence, in order to minimise the cost of searching for the optimal we go choose $F^{-1}[\{G\}]$ using the following rule:

$$\textbf{Select } F^{-1}[\{G\}], \textbf{ where } G = \arg\max_{G \in \mathcal{G}}(UB_G)$$

Note that this rule, which implies best-first search, applies only if we are seeking the optimal solution. In case we are after a near-optimal solution where a bound $\beta \in [0, 1]$ is specified (e.g., $\beta = 0.95$ means that the solution sought only needs to be 95% efficient in the worst case), then the selection function will be different since we do not need to search the sub-space with $UB_G = UB^{max}$ in order to return a possible solution at any time. Rather, we need to search sub-spaces that are smaller but could give a value close to $\beta \times UB^{max}$. The point to note is that, given our representation, we are able to specify β in cases where computing the optimal solution would be too costly and, given this, we modify the selection function accordingly to speed up the search.

Another advantage of being able to control the configuration selected for the search is that agents can choose what type of coalition structures to build according to their computational resources or private preferences (Sandholm *et al.* 1999; Shehory & Kraus 1998). For example, it has been argued that the computation time could be reduced if we could limit the size of the coalitions that could be chosen. However, this is a very costly self-imposed constraint since it possibly means neglecting a number of highly efficient solutions. Instead, using our approach, it is possible to determine, *ex-ante* (before performing the search), which coalition structure configurations are most promising according to their upper and lower bounds. Therefore the computation time can be focused on these configurations and the gains can be traded-off against the computation time since the size of a given sub-space can be exactly computed using the following equation:

$$|F^{-1}[\{G\}]| = \frac{\binom{|A|}{g_1} \times \binom{|A| - g_1}{g_2} \times \ldots \times \binom{|A| - \sum_{i=1}^{k-2} g_i}{g_{k-1}}}{\prod_{i=1, i \in E(G)}^{|A|} G(i)!} \quad (1)$$

where $E(G)$ is the underlying set[4] of elements of $G = \{g_1, \ldots, g_k\}$.

In cases where agents do prefer coalition structures of particular types (e.g., containing bigger or smaller coalitions), they can now, *a priori*, balance such preferences with the quality of the solutions (bounded by AVG_G) that can be obtained from such coalition structures. Indeed, this is because, in our case, it is possible to determine the worst-case bound from the optimal that the search of a given subspace will generate (i.e. $\frac{UB_{max}}{AVG_G}$). We next describe how we search the elements of the chosen sub-space $F^{-1}[\{G\}]$.

Searching within $F^{-1}[\{G\}]$. This stage is one of the main potential bottlenecks since it could ultimately involve searching a space that could possibly equal the whole space of coalition structures (if all $F^{-1}[\{G\}]$ need to be searched). The challenge then lies in avoiding computing redundant, as well as invalid, solutions. Moreover, it is critical that we identify those solutions that are bound to be lower than the current best and avoid computing them. Rahwan *et al.* (2007) first presented a technique for cycling through only valid and unique coalition structures within $F^{-1}[\{G\}]$. However, their procedure does not avoid the problem of having to gather all coalitions of each possible coalition structure. Given this, we describe a novel branch-and-bound approach that avoids retrieving all coalitions of every possible coalition structure by identifying possible solutions that are worse than the current best.

Applying Branch-and-Bound. Each valid coalition structure $CS = \{C_1 \ldots, C_k, C_{k+1}, \ldots, C_{|G|}\}$ of configuration

[4] For example $\{1, 2\}$ is the underlying set of $\{1, 1, 2\}$.

$G = \{g_1, \ldots, g_k, g_{k+1}, \ldots, g_{|G|}\}$ is generated by summing values of all coalitions within it one at a time. When the sum of values of coalitions $C_1, \ldots, C_k$ is computed during the search, it is also possible to compute an upper bound for the other coalitions (i.e. the feasible region) $C_{k+1}, \ldots, C_{|G|}$ that could be added. This upper bound can be computed using max_s for every possible coalition size $s \in 1, 2, \ldots, |A|$ (as above). Let this upper bound be computed as $UB_{\{g_{k+1},\ldots,g_{|G|}\}} = \sum_{i=k+1}^{i=|G|} max_{g_i}$. Also, let LB be the current best solution found so far and $V(C_1, ..., C_k) = \sum_{i=1}^{i=k} v(C_i)$. Then, if $LB > V(C_1, ..., C_k) + UB_{\{g_{k+1},\ldots,g_{|G|}\}}$ we do not need to compute those coalition structures that start with $C_1, ..., C_k$ and end with coalitions of size $g_{k+1}, ..., g_{|G|}$ since they are bound to be lower than the current best solution. Graphically, this is expressed by avoiding the move to rightmost columns (i.e. size 3 or size 4 depending on the difference between the sum of $v(C_x)$ with max_3 or max_4 respectively and the maximum value LB found so far) as in figure 1.

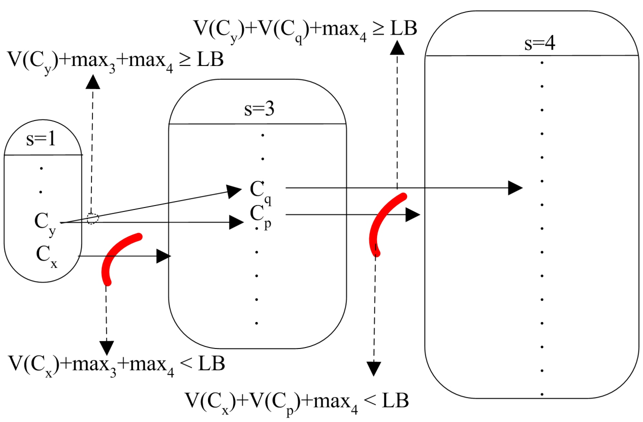

Figure 1: Applying branch-and-bound on a sub-space with $G = \{1, 3, 4\}$. Coalitions starting with C_x or $\{C_y, C_p\}$ have a lower upper bound than LB and therefore are not searched (denoted by red arcs) , while coalitions with $\{C_y, C_q\}$ could be better than LB and are searched.

Experimental Evaluation

In this section we empirically evaluate and benchmark our algorithm. The general hypothesis is that it will perform better than current approaches. However, a potential criticism that can be levelled against our algorithm is that, contrary to the other approaches, it is dependent on computing upper and lower bounds that are relatively close to the actual optimal value in order to prune large parts of the space and so guarantee that the optimal value has been found. Since this closeness to the optimal is determined by the spread of the distribution of the values of the coalitions, it is crucial that we test our algorithm against different distributions of input values and show that it is robust to all of them. However, we also aim to determine which types of inputs allow us to clearly delineate the most promising sub-spaces very quickly.

Experimental Setup

We test our algorithm with the four value distributions used and defined by (Larson & Sandholm 2000): Normal, Uniform, Sub-additive, and Super-additive.

Using the same input, we tested the other state-of-the art algorithms, namely DP and Integer Programming (using ILOG's CPLEX). We do not experiment with the other anytime algorithms since they need to search the whole space to find the optimal value and this is not feasible within reasonable time for more than 8 agents.

Results

Given the above setup, we ran DP, CPLEX and our algorithm 20 times for $|A| \in \{15, 16, \ldots, 26, 27\}$ and recorded the clock time[5] taken to find the optimal value. The DP algorithm has a deterministic running time since it always performs the same operations which grow in $O(3^{|A|})$. Hence, we computed the results for DP up to 20 agents and extrapolated the rest of the points (since the DP algorithm takes an unreasonable amount of time and runs out of memory for higher values). For each point, we computed the 95% confidence interval which are plotted as error bars on the graphs.

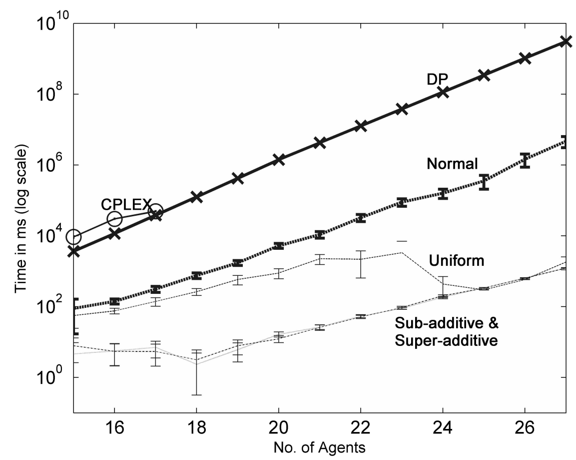

Figure 2: Running times for CSG algorithms for 15 to 27 agents (log scale).

As can be seen from figure 2 (in log scale), our algorithm always finds the optimal value for all distributions faster than the other algorithms. In the worst case, our algorithm finds the solution for 27 agents in 4.69×10^3 seconds (i.e. 1.3 hours), while the DP algorithm takes 5.67×10^6 seconds (i.e around 2 months), which means our algorithm takes 0.082% of the time taken by DP (an improvement that gets exponentially better with increasing numbers of agents). Moreover, CPLEX is found to be slower than DP and runs out of memory when there are more than 17 agents. Our algorithm performs worst, comparatively speaking, when the

[5]The experiments were carried out on a Xeon dual-core PC with 2GB of RAM. The algorithms were implemented in Java 1.5.

input is a normal distribution of values. This corroborates our initial expectations about the relationship between the spread of the distribution and the time it takes to find the optimal. Indeed, compared to the uniform distribution (against which our algorithm has a slowly increasing running time), the normal distribution concentrates most values around the mean. This means that there are very few values at the upper tail of the distribution that will fit into a valid coalition structure. It can also be noted that the sub-additive and super-additive distributions are solved nearly instantaneously (right after scanning the input; that is, after 1.241 seconds for 27 agents). This means that, *in the best case*, our algorithm takes $2.2 \times 10^{-5}\%$ of the time of the DP algorithm. In the sub and super-additive case, it is easy to verify that our algorithm, by virtue of its computation of upper and lower bounds, identifies the optimal solution straight after scanning the input since the upper bound of the sub-spaces in these cases (without knowing whether the input is super or sub-additive) are always lower than the grand coalition (in the super additive case) or the coalitions of single agents (in the sub additive case). For the uniform distribution, it is noted that the optimal value is found much quicker than the normal distribution and, as the number of agents grows beyond 24, the optimal value is found as fast as in the sub or super-additive case. This can only happen if the optimal is found just after scanning the input and is explained by the fact that as the number of agents increases, there is an increased likelihood that the optimal solution will be found in the combination of coalitions of big sizes (and these are usually found in sub-spaces with configurations in $G \in \mathcal{G}^2$). Moreover, in the uniform case, we can expect most of the optimal coalition structures within a sub-space to have values close to the upper bound. This results in either the most promising sub-space being indentified with a relatively high degree of accuracy or in the sub-space being pruned right after scanning the input.

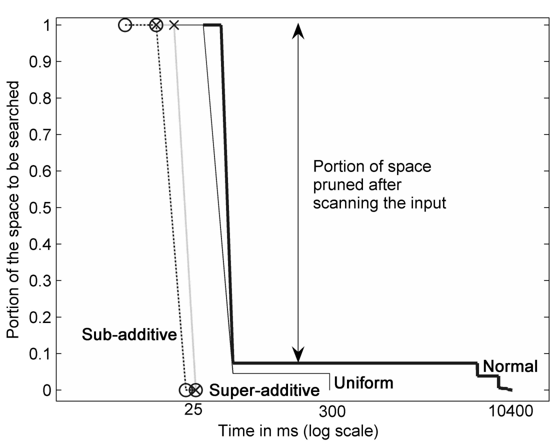

Figure 3: Space pruned for each distribution type (for 21 agents).

To further support our claim regarding the relationship between the distribution type and the pruning of the search space, we studied the space remaining to be searched, as well as the quality of the solution found during the search (see figure 3 for the 21 agents case, other values gave similar patterns). To this end, we recorded the percentage of the space remaining at each pruning attempt, as well as the value of the ratio of the best solution found to the optimal value during the search. As can be seen from figure 3, the major drops in the space left to be searched indicate that large sub-spaces are being pruned, while when the graph is flat, branch-and-bound is being applied within sub-spaces to reduce the solving time. In more detail, our algorithm tends to be less able to prune the space in the case of the normal distribution. In fact, in such cases most of the time is spent searching extremely small portions of the space (since the graph is flat most of the time) for a long time until the optimal value can be confirmed. During this search, the solution does not improve as much, as can be seen from figure 4. In the case of the sub and super additive distributions, the solution is found nearly instantaneously right after scanning the input. For the uniform case, we are able to prune most of the space right from the beginning and then the algorithm takes some time to find the optimal. From figure 4 it can also be seen that intermediate solutions found during the search becomes near-optimal very rapidly ($> 95\%$ of the optimal). This shows that our algorithm rapidly zooms in on the most promising sub-spaces and finds good solutions quickly within these.

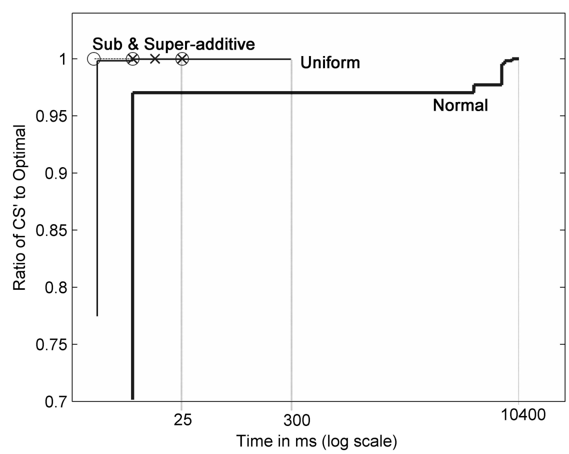

Figure 4: Quality of the solution obtained during the search (for 21 agents).

Conclusions

We have devised an anytime algorithm that can compute optimal coalition structures. Moreover, we have shown that it is significantly faster than the current state of the art. This efficiency is based on (i) a novel representation of the search space and efficient search strategies that can exploit the representation and (ii) a branch-and-bound technique that can rapidly identify the best coalition structures during the search and therefore prune bigger portions of the space than has hitherto been possible.

Future work will seek to investigate the process of distributing the search procedure among multiple agents so as to speed up the search still further. We believe this is possible since our representation easily allows us to assign each agent an independent portion of the space to search.

Acknowledgements

We wish to thank Prof. Tuomas Sandholm for his comments on the algorithm. 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 (EP/C548051/1). Andrea Giovannucci was supported by a CSIC I3P scholarship.

References

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

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

Rahwan, T.; Ramchurn, S. D.; Dang, V. D.; and Jennings, N. R. 2007. Near-optimal anytime coalition structure generation. In *IJCAI*, 2365–2371.

Rothkopf, M. H.; Pekec, A.; and Harstad, R. M. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147.

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

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

Skiena, S. S. 1998. *The Algorithm Design Manual*. Springer.

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

Appendix: Proof of Theorem 1

Let $\bar{G} = (g_1, g_2, ..., g_k)$. That is, $\bar{G}$ contains elements of G with a lexicographic ordering on them. Then let $\bar{F}^{-1}[\{\bar{G}\}]$ return all *ordered coalition structures* $(C_1, C_2, \ldots, C_k)$, $C_i \in CL_{g_i}$. That is, the lexicographic ordering of the elements C_i of each coalition structure is taken into consideration. For example, with $a = 4$ and $G = \{1, 1, 2\}$, $\bar{G} = \{1, 1, 2\}$; then considering ordered coalition structures in $\bar{F}^{-1}[\{\bar{G}\}]$, we have two possibilities: $(\{a_1\}, \{a_2\}, \{a_3, a_4\})$ and $(\{a_2\}, \{a_1\}, \{a_3, a_4\})$ that correspond to one coalition structure $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$ in $F^{-1}[\{G\}]$. Moreover, as the number of repetitions of different coalition structures of $F^{-1}[\{G\}]$ in $\bar{F}^{-1}[\{\bar{G}\}]$ is always the same (e.g., in the above example with $\bar{G} = \{1, 1, 2\}$, all coalition structures in $F^{-1}[\{G\}]$ will appear twice in $\bar{F}^{-1}[\{\bar{G}\}]$), we have:

$$AVG_G = AVG_{\bar{G}} \tag{2}$$

where $AVG_{\bar{G}}$ is the average of coalition structures in $\bar{F}^{-1}[\{\bar{G}\}]$.

Then, let $N_n(g_1, g_2, ..., g_k)$ (with $n = \sum_{g_i \in G} g_i$) be the number of ordered coalition structures in $\bar{F}^{-1}[\{\bar{G}\}]$ and $K_n(g_i)$ be the number of coalitions of size g_i in a system consisting of n agents. Clearly, $K_n(g_i) = \frac{n!}{g_i!(n-g_i)!}$.

Now for each coalition $C_i \in CL_{g_i}$, there are $N_{n-g_i}(g_1, g_2,$ $..., g_{i-1}, g_{i+1}, ..., g_k)$ ordered coalition structures that contains it, so we have:

$$N_n(g_1, g_2, ..., g_k) = K_n(g_i) N_{n-g_i}(g_1, ..., g_{i-1}, g_{i+1}, ..., g_k) \tag{3}$$

Also, we have:

$$\begin{aligned} AVG_{\bar{G}} &= \frac{1}{N_n(g_1, g_2, ..., g_k)} \sum_{CS \in \bar{F}^{-1}[\{\bar{G}\}]} V(CS) \\ &= \frac{1}{N_n(g_1, g_2, ..., g_k)} \sum_{CS \in \bar{F}^{-1}[\{\bar{G}\}]} \sum_{i=1, C_i \in CS}^{k} v(C_i) \end{aligned}$$

Given this, we next compute $AVG_{\bar{G}}$ as follows. Let CL'_{g_i} be the set of all coalitions with size g_i and being the $i-th$ coalition in an ordered coalition structure $CS \in \bar{F}^{-1}[\{\bar{G}\}]$. Now for any $C_i \in CL'_{g_i}$, the number of times that $v(C_i)$ occurs in the sum of all coalition values in $F^{-1}[\{\bar{G}\}]$ is $N_{n-g_i}(g_1, ..., g_{i-1}, g_{i+1}, ..., g_k)$. Thus:

$$\begin{aligned} AVG_{\bar{G}} &= \frac{1}{N_n(g_1, g_2, ..., g_k)} \\ &\quad \sum_{i=1}^{k} \sum_{C_i \in CL'_{g_i}} N_{n-g_i}(g_1, ..., g_{i-1}, g_{i+1}, ..., s_k) v(C_i) \\ &= \sum_{i=1}^{k} \sum_{C_i \in CL'_{g_i}} \frac{N_{n-g_i}(g_1, ..., g_{i-1}, g_{i+1}, ..., g_k)}{N_n(g_1, g_2, ..., g_k)} v(C_i) \\ &= \sum_{i=1}^{k} \sum_{C_i \in CL'_{g_i}} \frac{1}{K_n(g_i)} v(C_i) \text{ (following equation (3))} \\ &= \sum_{i=1}^{k} \left(\frac{1}{K_n(g_i)} \sum_{C_i \in CL'_{g_i}} v(C_i) \right) \\ &= \sum_{i=1}^{k} avg_{g_i} \end{aligned}$$

As $AVG_G = AVG_{\bar{G}}$ (equation (2)), we have:

$$AVG_G = \sum_{i=1}^{k} avg_{g_i}$$

□