Anytime Coalition Structure Generation on Synergy Graphs

Filippo Bistaffa, Alessandro Farinelli
University of Verona
filippo.bistaffa@univr.it
alessandro.farinelli@univr.it

Jesús Cerquides, Juan Rodríguez-Aguilar
IIIA-CSIC
cerquide@iiia.csic.es
jar@iiia.csic.es

Sarvapali D. Ramchurn
University of Southampton, UK
sdr@ecs.soton.ac.uk

ABSTRACT

We consider the coalition structure generation (CSG) problem on synergy graphs, which arises in many practical applications where communication constraints, social or trust relationships must be taken into account when forming coalitions. We propose a novel representation of this problem based on the concept of *edge contraction*, and an innovative branch and bound approach (CFSS), which is particularly efficient when applied to a general class of characteristic functions. This new model provides a non-redundant partition of the search space, hence allowing an effective parallelisation. We evaluate CFSS on two benchmark functions, the *edge sum with coordination cost* and the *collective energy purchasing* functions, comparing its performance with the best algorithm for CSG on synergy graphs: DyCE. The latter approach is centralised and cannot be efficiently parallelised due to the exponential memory requirements in the number of agents, which limits its scalability (while CFSS memory requirements are only polynomial). Our results show that, when the graphs are very sparse, CFSS is 4 orders of magnitude faster than DyCE. Moreover, CFSS is the first approach to provide anytime approximate solutions with quality guarantees for very large systems (i.e., with more than 2700 agents).

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms

Keywords

Coalition Formation, Networks, Graphs, Collective Energy Purchasing

1. INTRODUCTION

Coalition formation is one of the fundamental approaches for establishing collaborations among agents, each with individual capabilities and properties [12]. In particular, coalition structure generation (CSG) represents a key computational task in this scenario [10]. Now, in many real-world applications, sparse synergies between the agents may constrain the formation of some coalitions [13, 14].

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.*

These constraints may be due to communication infrastructures (e.g., non-overlapping communication loci or energy limitations for sending messages across a network), social or trust relationships (e.g., energy consumers who prefer to group with their friends and relatives in forming energy cooperatives), or physical constraints (e.g., emergency responders that have enough fuel to join only specific teams).

Recently, several approaches have been proposed to model these constraints as graphs, where nodes represent agents and edges enable the connected agents to form a coalition [13, 14]. Hence, a coalition is considered feasible only if its members represent the vertices of a connected subgraph of the constraint graph.

In this context, the most prominent solution techniques for coalition structure generation are presented in [13, 14], and while they both represent significant contributions to the state of the art, there are some drawbacks that hinder their applicability. On the one hand, [13] requires that the valuation function fulfils the independence of disconnected members (IDM) property,[1] which may not hold in many real world applications, considering that the addition of every new agent to a coalition typically has a cost that depends on the specific application domain [12]. On the other hand, the memory requirements of [14] grow exponentially in the number of agents, hence limiting the scalability of such technique. Moreover, both algorithms are serial and none of them can provide anytime approximate solutions.

Providing anytime approximate solutions is an important topic in CSG [11, 9], and currently the work of [9] sets the state-of-the-art for anytime algorithms. Nonetheless, as discussed in [14], the behaviour of such algorithms is undefined for cases where only some coalitions are feasible, as simply assigning artificially low values (such as $-\infty$) to unfeasible coalitions would not be suitable for assessing valid bounds.

Against this background, here we propose the first anytime algorithm for coalition structure generation on synergy graphs, called CFSS (Coalition Formation with Sparse Synergies). Unlike previous approaches, CFSS provides anytime solutions with quality guarantees for large scale systems (i.e., more that 2700 agents). This tremendous speedup is possible because we focus on a general class of characteristic functions, the so called $m + a$ (monotonic-antimonotonic) functions, for which we can efficiently provide tight bounds. Moreover, thanks to the novel representation of the search space, CFSS can be effectively parallelised, thus fully exploiting modern multi-core architectures.

In more detail, this paper makes the following contributions to the state-of-the-art:

[1]The IDM property requires that, given two disconnected agents i and j, the presence of agent i does not affect the marginal contribution of agent j to a coalition.

- We provide a new representation for CSG on synergy graphs proving that, by performing a series of edge contraction operations on the constraint graph, it is possible to efficiently build a search tree where each node corresponds to a feasible coalition structure, while avoiding redundancy (i.e., a coalition structure will appear only once in our search tree). This new model allows a non-redundant partition of the search space, enabling an efficient parallel solving algorithm (with a speedup higher than 7× on a 12 cores machine).
- We propose a branch and bound strategy that, when applied to $m + a$ functions, can efficiently solve the CSG problem providing anytime approximate solutions with tight bounds. Moreover, we study two particular cases of $m + a$ functions which were previously proposed in the CSG literature: i) *edge sum with coordination cost* [13] and ii) *collective energy purchasing* [5], detailing the computation of the bounds.
- We compare CFSS with DyCE, the state-of-the-art algorithm for CSG on synergy graphs. Our results show that CFSS is 4 orders of magnitude faster than DyCE for the *edge sum with coordination cost* function and, for very sparse graphs, our algorithm can solve problems with more than 100 agents, far beyond the current limit of DyCE. Furthermore, we evaluate the performance and optimality guarantees that our algorithm yields when scaling to very large systems (i.e., more than 2700 agents). Our results show that CFSS provides a tight approximation ratio,[2] which is always lower than 1.12, thus producing solutions that are, in the worst case, at least 88% of the optimal.

The rest of the paper is organised as follows: Section 2 illustrates the background on CSG on synergy graphs and on the DyCE algorithm. Section 3 gives a method to generate our search space while Section 4 details our branch and bound approach. Section 5 discusses our empirical evaluation and Section 6 concludes the paper.

2. PROBLEM AND BACKGROUND

The purpose of this section is twofold. First, in Section 2.1 we define the CSG problem on synergy graphs based on the definition introduced in [14]. Second, in Section 2.2 we provide some background on the DyCE algorithm.

2.1 Problem definition

Consider a set of agents $\mathcal{A} = \{a_1, \ldots, a_N\}$, where N is the total number of agents. Consider also a connected[3] graph $G = (\mathcal{A}, E)$, where $E \subseteq \mathcal{A} \times \mathcal{A}$ is a set of edges between agents, representing the relationships between them. We say that a coalition of agents $C \subseteq \mathcal{A}$ is *feasible* if and only if there exists a connected subgraph $G' = (C, E')$ whose vertices are the agents in C, and whose edges are a subset of those in G, namely $E' \subseteq E$. Thus, the set of feasible coalitions is *restricted* by the relationships represented by the graph G. Henceforth, we refer to the set of all feasible coalitions in G as $\mathcal{FC}(G)$. For example, assume the graph in Figure 2(a) represents a social network between users that want to buy energy together at reduced tariffs. Edges here represent the *friend* relationship and hence two agents that are not direct friends (e.g., D and F) cannot form a coalition on their own (i.e., $\{D, F\} \notin \mathcal{FC}(G)$). However, the same agents can form a coalition if a common friend joins such coalition: $\{A, D, F\} \in \mathcal{FC}(G)$.

[2]The ratio between the upper bound on the optimal solution and the value of the solution returned by our approach.

[3]If the graph is not connected we decompose the problem into smaller independent subproblems, each with a connected graph.

To value a feasible coalition $C \in \mathcal{FC}(G)$, we assume that there is a function $v : \mathcal{FC}(G) \rightarrow \Re$. Although function v may be arbitrarily defined, we will assume that the value of a coalition is independent of any other coalitions that may exist. For example, in the *collective energy purchasing* scenario, the value of a coalition is the total cost to the agents if they buy energy together, and this cost does not depend on which coalitions other agents will form. Given the coalitional values, the CSG problem on synergy graphs involves finding the optimal *coalition structure* (i.e., a partition of the set of agents) represented by the graph. Hence, the optimal solution is represented by the best set of disjoint feasible coalitions that collectively cover all agents. To find the optimal coalition structure, we must consider the set of all feasible coalition structures, $\mathcal{FCS}(G)$, namely the set of coalition structures that only contain feasible coalitions. Then, solving the CSG problem on a synergy graph G amounts to find the coalition structure CS^*:

$$CS^* = \arg\max_{CS \in \mathcal{FCS}(G)} \underbrace{\sum_{C \in CS} v(C)}_{f(CS)} \quad (1)$$

where the function $f : \mathcal{FCS}(G) \rightarrow \Re$ is obtained by adding the coalitional values of the single coalitions. Given this objective, we next describe the current best solution approach, namely the DyCE algorithm (which we benchmark against in Section 5).

2.2 The DyCE algorithm

DyCE [14] uses dynamic programming to find the optimal coalition structure by progressively splitting the current solution into its best partition. This modus operandi is similar to IDP [8], with the fundamental addition of an initial preprocessing phase that enumerates all feasible coalitions. Then, DyCE considers only such coalitions significantly reducing the run time of the overall approach. However, DyCE requires an exponential amount of memory in the number of agents (i.e., $O(2^n)$) and is not an anytime approach. Hence, the scalability of such approach is limited to systems consisting of tens of agents (e.g., around 30). Moreover, DyCE is a serial algorithm and cannot be efficiently parallelised due to the exponential memory requirements in the number of agents. To overcome these limitations, we propose a new way of representing the CSG problem, and in particular its search space, by means of edge contractions, as detailed in the next section.

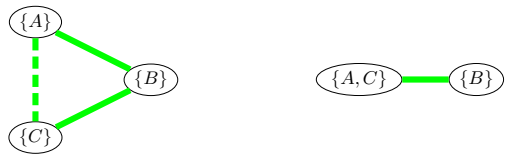
3. REPRESENTING THE SEARCH SPACE

In this section we show that all feasible coalition structures induced by G can be easily modelled as a search tree in which each feasible coalition structure is represented only once. Specifically, we first detail how we can use edge contractions to represent the CSG problem and then we provide an algorithm to build the search space.

3.1 Representing CSG via edge contractions

The main idea to generate our search space is that each feasible coalition structure can be represented as the contraction of a set of edges of G, and that each contraction of a set of edges of G represents a feasible coalition structure. In more detail, let us define an *edge contraction* as follows:

DEFINITION 1. *Given a graph $G = (V, E)$ and an edge $e = (u, v)$, where $e \in E$ and $u, v \in V$, the result of the contraction of edge $e = (u, v)$ is a graph G' obtained by removing edge e and adding a new vertex w obtained by merging u and v (i.e., labelling w with the union of their labels). Moreover, each edge incident to either u or v in G will become incident on w in G', merging the parallel edges[4] that may result from the aforementioned operations.*



(a) Before contraction (b) After contraction

Figure 1: Example of an edge contraction in a triangle graph (the dashed edge is contracted to give the graph on the right)

Intuitively, a contraction of an edge represents the merging of the coalitions associated to its incident vertices. Figure 1 shows the contraction of the edge $(\{A\},\{C\})$, which results in a new vertex $\{A,C\}$ connected to vertex $\{B\}$. Notice that edge contraction is a commutative operation (i.e., first contracting e and then e' results in the same graph as first contracting e' and then e). Hence, we can define the contraction of a set of edges as the result of contracting each of the edges of the set in any given order. We can now show that the following propositions holds:

PROPOSITION 1. *The graph resulting from the contraction of any set of edges represents a feasible coalition structure, where coalitions correspond to the labels of the vertices of the resulting graph.*

PROOF. It is easy to see that an edge contraction in a graph representing a feasible coalition structure will generate a coalition structure still feasible, without any loss of information: in fact, only redundant constraints (i.e., the parallel edges resulting from the contraction) are removed, while any other edge in the original graph is preserved. Hence, by induction starting from the coalition structure of all singletons, a sequence of edge contractions will result in a feasible coalition structure. □

PROPOSITION 2. *Any feasible coalition structure CS can be generated by contracting a set of edges of G.*

PROOF. Each coalition $C_i \in CS$ induces a connected subgraph of G, namely $SG_i = (V_i, E_i)$, in which V_i is the set of agents members of C_i and E_i is the set of edges among them in G. Every C_i can be generated by contracting the set of edges of any possible spanning tree of the corresponding subgraph SG_i, and repeating this operation for all $C_i \in CS$ we can generate CS. Notice that the contraction of the edges of SG_i will not affect the edges of a different subgraph SG_j, since only redundant edges are removed. □

Thus, a possible way of listing all feasible coalition structures is to list the contraction of every subset of edges of the initial graph. However, notice that the number of subsets of edges is larger than the number of feasible coalition structures over the graph. For example, in the triangle graph in Figure 1(a), contracting any two edges leads to the grand coalition $\mathcal{A} = \{A,B,C\}$. Thus, we need a way to avoid listing feasible coalition structures more than once. To avoid such redundancies, we mark each edge of the graph so to keep track of the edges that have been contracted so far. Notice that there are only two different alternative actions for each edge: either we contract it, or we do not. If we decide to contract an edge, such edge will be removed from the graph in all the subtree rooted in the current node, but if we decide not to contract it we have to mark such edge to make sure that we do not contract it in the future operations of the search space generation. To represent such marking, we will use the notion of *2-coloured graph*:

[4]Two edges are parallel if they are incident on the same two vertices.

(a) Before contraction (b) After contraction

Figure 2: Example of a green edge contraction in a 2-coloured graph (the dashed edge is contracted to give the graph on the right)

DEFINITION 2. *A 2-coloured graph $G^c = (V,E,c)$ is composed of a set of vertices V and a set of edges E, as well as a function $c : E \to \{red, green\}$ that assigns a colour (red or $green$) to each edge of the graph.*

In our case, a red edge means that a previous decision not to contract that edge was made. On the other hand, green edges can be still contracted. Figure 2(a) shows an example of a 2-colour graph in which edge $(\{A\},\{D\})$ is coloured in red: hence, in any subsequent step of the algorithm it is impossible to contract it. On the other hand, all other edges in such graph can still be contracted. In a 2-coloured graph, we define a *green edge contraction* as follows:

DEFINITION 3. *Given a 2-coloured graph $G = (V,E,c)$ and a green edge $e = (u,v)$, where $e \in E$ and $u,v \in V$, the result of the contraction of edge $e = (u,v)$ is a graph G' obtained by performing the contraction of the edge e in the graph G. Whenever two parallel edges are merged into a single one, the resulting edge is coloured in red if at least one of them is red-coloured, and it is green-coloured otherwise.*

The rationale behind marking parallel edges in this way is that, whenever we mark an edge $e = (u,v)$ to be red, we want the nodes of that edge to be in separate coalitions, hence whenever we merge some edges with e we must mark the new edge as red to be sure that future edge contractions will not generate a coalition that contains both the agents corresponding to nodes u and v. For example, note that in Figure 2 the red edge $(\{D\},\{A\})$ (dotted in the figure) and the green edge $(\{D\},\{C\})$ are merged as a consequence of the contraction of edge $(\{A\},\{C\})$, resulting in an edge $(\{D\},\{A,C\})$ marked in red. In this way, we enforce that any possible contraction in the new graph will keep agents A and D in separate coalitions.

Having defined how we can use the edge contraction operation to represent coalition structures, we now provide a way to model the search space of the CSG problem on synergy graphs.

3.2 Generating the entire search space

Given the green edge contraction operation defined above, we can generate each feasible coalition structure only once. In more detail, at each point of the generation process, each red edge indicates that it has been discarded for contraction from that point onwards, and hence its vertices cannot be joined. Observe that the way we defined green edge contraction guarantees that the information in red edges is always preserved. Thus, given a 2-coloured graph, its children can be readily assessed as follows: for each edge in the graph, we generate the graph that results from contracting that edge. Moreover, we colour the selected edge in red so that it cannot be contracted again in subsequent edge contractions. As an example, Figure 3 shows the search tree of a square graph. Algorithm 1 shows the procedure that builds our search tree, and it is possible to show that such algorithm visits all feasible coalition structures and each of them is visited only once. In more detail, we can prove the following proposition:

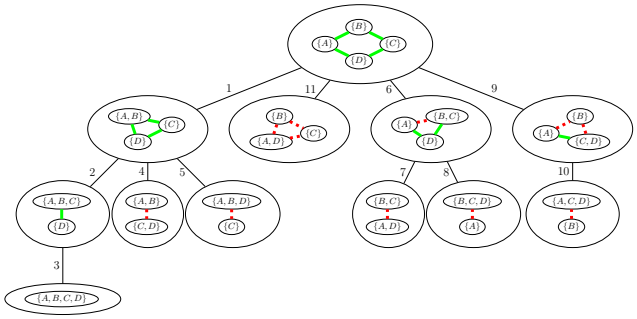


Figure 3: Search tree for a square graph

PROPOSITION 3. *Given a 2-coloured graph G^c, the tree rooted at G^c contains all the coalition structures compatible with G^c, and each of them appears only once.*

PROOF. See Appendix. □

Then, as a consequence of the previous proposition, we can prove the following corollary:

COROLLARY 1. *There is a bijection between $\mathcal{FCS}(G)$ and the nodes visited by Algorithm 1.*

PROOF. By direct application of Proposition 3 to the initial graph G with all edges coloured green. □

Corollary 1 ensures that by visiting the entire search tree and evaluating the value of each coalition structure, we can always find the optimal solution. Moreover, each feasible coalition is represented only once in the tree, hence if we search different branches in parallel (i.e., assigning different iterations of the *for all* statement at line 2 to different threads/cores) we will not have redundant computations.[5]

Algorithm 1 VISITALLCOALITIONSTRUCTURES(G^c)

```
1: VISIT(G^c)
2: for all G ∈ CHILDREN(G^c) do
3:     VISITALLCOALITIONSTRUCTURES(G)
4:
5: function CHILDREN(G^c)
6:     G' ← G^c                          ▷ Initialise graph G' with G^c
7:     Ch ← ∅                            ▷ Initialise the set of children
8:     for all e ∈ G^c : c(e) = green do  ▷ For all green edges
9:         Ch ← Ch ∪ {GREENEDGECONTRACTION(G', e)}
10:        Mark edge e with colour red in G'
11:    return Ch                         ▷ Return the set of children
```

Nonetheless, it is possible that, even for sparse graphs, the number of feasible coalition structures is very large, making their visit not affordable. Hence, in the next section we propose a branch and bound technique that prunes significant parts of the search space.

4. THE CFSS ALGORITHM

In this section we describe CFSS, our branch and bound approach to CSG on synergy graphs. In particular, we will focus on a general class of characteristic functions (called $m+a$ functions). Next, we show that two specific functions, previously used in the literature, are $m+a$ functions, detailing how we can provide bounds for such functions so to drive the branch and bound strategy.

[5]This is important when several agents with heterogeneous capabilities need to share the computation of the solution.

4.1 A general branch and bound algorithm for $m+a$ functions

As shown in Equation 1, our objective is to maximise a function whose domain is the set of feasible coalition structures $\mathcal{FCS}(G)$. On the other hand, a strategy to prune significant portions of the search space is needed to have a computationally affordable solution technique. In what follows, we detail some properties of our reference domain that our branch and bound approach exploits:

PROPERTY 1. *$\mathcal{FCS}(G)$ is a lattice, i.e., a partially ordered set, in which every two elements CS_i and CS_j have a supremum ($CS_i \vee CS_j$) and an infimum ($CS_i \wedge CS_j$).*

In particular, we define the following partial order over partitions:

DEFINITION 4. *Given any two partitions CS_i and CS_j, we say that $CS_i \preceq CS_j$ if every element of CS_i is a subset of some element of CS_j.*

As an example, $\{A,B\}\{C\} \preceq \{A,B,C\}$, but the order between $\{A,B\}\{C\}$ and $\{A\}\{B,C\}$ is not defined. It is well known that with this partial order the set of partitions forms a complete lattice (see Section V.4 in [6]), called the partition lattice or equivalence lattice. It is easy to see that our domain of interest is the sublattice generated by the set of feasible coalition structures and thus it is a lattice. Furthermore, in our scenario, the grand coalition represents a supremum of any two elements, while the coalition structure of all singletons represents an infimum.

PROPERTY 2. *The elements of $\mathcal{FCS}(G)$ can be arranged in an order-preserving tree: whenever CS_j is a descendant of CS_i in the tree, then $CS_j \succeq CS_i$. Thus, CS_i is the infimum of the subtree rooted at CS_i:*

$$CS_i = \bigwedge ST(CS_i) = \inf ST(CS_i)$$

In the search tree defined in Section 3 each child is the result of contracting an edge in the parent. As a consequence of the contraction, two of the coalitions in the parent are merged, making the child coalition coarser than that of the parent. Hence, by direct application of Property 1, the above statement holds.

PROPERTY 3. *Given a node CS_i, there is a computationally efficient procedure to assess an element $\overline{CS_i}$ that is bigger than any of the elements of the subtree:*

$$\overline{CS_i} \geq \bigvee ST(CS_i) = \sup ST(CS_i)$$

It is easy to see that $\overline{CS_i}$ can be found by removing all red edges from the 2-coloured graph representing CS_i and then contracting all the remaining green edges (which is equivalent to find the connected components in the graph after the removal of all red edges). What we are doing can be interpreted as finding the coarsest partition forgetting that we decided not to contract some edges. Clearly, any partition in the subtree will be at most as coarse as this one. This procedure can be implemented in time $O(N+E)$, visiting all the vertices that can be reached from any starting node by means of a breadth-first search, and iterating this procedure starting from any node that could not be reached, until all the vertices have been visited.

PROPERTY 4. *The function f is an $m+a$ function ($f = f^+ + f^-$), i.e., it is the sum of a monotonic function f^+ and an anti-monotonic function f^-.*[6]

[6]A function g is monotonic (resp. anti-monotonic) if, for all x and y such that $x \leq y$, one has that $g(x) \leq g(y)$ (resp. $g(x) \geq g(y)$).

Functions that satisfy this property are interesting because they allow us to efficiently provide a tight bound that can drive the branch and bound strategy. Moreover, as shown in Section 4.2, Property 4 captures several realistic scenarios considered in the coalition formation domain.

It is interesting to note that several relevant optimisation problems, beyond CSG, satisfy these properties. As an example, consider the winner determination problem in combinatorial exchanges, where the objective is to label (winning or losing) several bids submitted by some buyers, so as to maximise the auctioneer's revenue under the constraint that each item can be allocated to at most one bidder. The domain of such problem can be understood as the lattice of subsets of bids with inclusion ordering. Moreover, the objective function can be split into a monotonic part, adding accepted buy offers (with positive values) and an anti-monotonic part adding accepted sell offers (with negative values).[7]

PROPOSITION 4. *If a domain satisfies the aforementioned properties, an efficient technique can be used to compute a tight bounding function. In particular, given a node CS_i:*

$$M(CS_i) = f^-(CS_i) + f^+\left(\overline{CS_i}\right)$$

is an admissible bound for the subtree rooted at CS_i:

$$M(CS_i) \geq \max\{f(CS_i) | CS_j \in ST(CS_i)\}$$

PROOF. Consider that, for the subtree rooted at CS_i, the maximum of an anti-monotonic function f^- will be achieved at CS_i (Property 2), i.e., $f^-(CS_i) \geq \max\{f^-(CS_j) | CS_j \in ST(CS_i)\}$. On the other hand, the maximum of a monotonic function f^+ will be reached at one of the leaves. However, since assessing the supremum $\overline{CS_i}$ of the subtree is computationally efficient (Property 3), we can bound f^+ in the subtree as follows: $f^+\left(\overline{CS_i}\right) \geq \max\{f^+(CS_j) | CS_j \in ST(CS_i)\}$. Finally, since our function is $m+a$ (Property 4), then we can easily provide a bound for the function f we are trying to maximise by composing these two results, i.e., $M(CS_i) = f^-(CS_i) + f^+\left(\overline{CS_i}\right)$. □

Building on Property 4, we can efficiently assess a bound for each subtree and prune it if the value of such bound is smaller than the value of the best solution found so far (as shown in Algorithm 2):

Algorithm 2 CFSS(G^c)

```
1: best ← G^c                          ▷ Initialise current best solution with singletons
2: F ← ∅                               ▷ Initialise frontier F with an empty stack
3: F.PUSH(G^c)                         ▷ Push G^c as the first node to visit
4: while F ≠ ∅ do                      ▷ Branch and bound loop
5:     node ← F.POP()                  ▷ Get current node
6:     if M(node) > f(best) then       ▷ Check bound value
7:         if f(node) > f(best) then   ▷ Check function value
8:             best ← node             ▷ Update current best solution
9:         F.PUSH(CHILDREN(node))      ▷ Update frontier F
10: return best                        ▷ Return optimal solution
```

Notice that such branch and bound procedure can be directly applied to compute an overall bound of an $m + a$ function, with anytime properties. More precisely, let us consider the frontier F reported in Algorithm 2. When we expand the frontier F (Line 9) we can keep track of the highest value of the function f evaluated in all visited nodes. Hence, given a frontier F on the search tree, the bound $B(F)$ is defined as:

$$B(F) = \max\left\{f(best), \max_{CS \in F} M(CS)\right\} \quad (2)$$

In other words, $B(F)$ is the maximum between the values assumed by f inside the frontier and an estimated bound outside of it.

Since each of these bounds is admissible, taking the maximum ensures the admissibility of $B(F)$. Furthermore, the quality of $B(F)$ can only be improved by expanding the frontier F: if F' is such expansion, then $B(F) \geq B(F') \geq \max\{f(CS) | CS \in \mathcal{FCS}(G)\}$. This can be easily verified recalling the definition of M: each bound resulting from the children of a substituted node $n \in F$ must be less or equal to $M(n)$, hence the above inequality holds. Intuitively, the more search space is explored, the better the bound provided. The fastest way to compute a bound for f is considering a frontier formed exclusively by the root: assessing this bound has the same time complexity of computing the function M, and its quality can be satisfactory, as shown in Section 5.4.

Even though Algorithm 2 could be applied to any characteristic function, the result in Section 4.1 can be used to prune significant portions of the search space as long as the characteristic function is the sum of a monotonic and an anti-monotonic function. Next, we provide two examples of functions that fulfil such property.

4.2 Benchmark functions for CSG on synergy graphs

In what follows, we focus on two benchmark functions for the CSG problem on synergy graphs, namely the *collective energy purchasing* function and the *edge sum with coordination cost* function. These functions are typical of realistic coalition formation problems as shown in [4, 5]. In particular, we are interested in their characterisation as $m + a$ functions, showing that they can be seen as the sum of a monotonic and an antimonotonic part, thus enabling the aforementioned bounding techniques to prune part of the search space during the execution of the CFSS algorithm.

4.2.1 Collective energy purchasing function

In the first scenario taken into account, each agent is characterised by an energy consumption profile that represents its energy consumption throughout a day [5]. In more detail, a profile records the energy consumption of a household at fixed intervals (every half hour in our case). Hence each profile is a vector of T elements (where $T = 48$ in our case), whose values represent the actual measurements collected over a month from 2732 households in UK. The characteristic function of a coalition of agents is the total cost that the group would incur if they buy energy as a collective on two different markets: the spot market, a short term market intended for small amounts of energy, and the forward market, a long term one in which bigger portions of energy can be bought at cheaper prices. In particular, following [5] the characteristic function is defined as:

$$v(C) = \sum_{t=1}^{T} q_S^t(C) \cdot p_S + T \cdot q_F(C) \cdot p_F + \kappa(C) \quad (3)$$

where p_S and p_F represent the unit price of energy in the spot and forward market respectively,[8] $q_F(C)$ stands for the time unit amount of electricity to buy in the forward market and $q_S^t(C)$ for the amount to buy in the spot market at time slot t. These quantities are the ones that optimise the buying strategy of the group while satisfying the group electricity demand (see [5] for further details). Finally, $\kappa(C)$ stands for a coalition management cost that depends on the size of the coalition and captures the intuition that larger coalitions are harder to manage.

[7] A more detailed analysis of the CFSS approach in such domain is outside the scope of this paper and will be subject of future work.

[8] Following [5], in our experiments we fixed $p_S = -80$ and $p_F = -70$, using negative unit prices to reflect the direction of payments.

The definition of this cost depends on several low level issues (e.g., the power network capacity of customers in the groups, legal fees, and other costs associated to group contracts, etc.), hence a precise definition of this term goes beyond the scope of this paper. Following [5], we use $\kappa(C) = -|C|^{\gamma}$ to introduce a non-linear element that penalises the formation of big coalitions, so that the grand coalition is not always the best coalition structure.

Hence, the *collective energy purchasing* function is defined as:

$$f(CS) = \underbrace{\sum_{C \in CS} \left[\sum_{t=1}^{T} q_S^t(C) \cdot p_S + T \cdot q_F(C) \cdot p_F \right]}_{f^+(CS)} + \underbrace{\sum_{C \in CS} \kappa(C)}_{f^-(CS)}$$

PROPOSITION 5. *The collective energy purchasing function is an $m + a$ function.*

PROOF. As shown in the above equation, this characteristic function can be seen as a $m + a$ function, being the sum of a monotonic function, consisting of the cost of the energy necessary to fulfil the aggregated consumption profiles of the coalitions, and an anti-monotonic one (i.e., the sum of the coalition management costs). In fact, given any coalition structure CS different from the grand coalition, it is possible to merge two coalition $C_1, C_2 \in CS$ and form a new coalition structure $CS' = CS \setminus \{C_1, C_2\} \cup \{C_1 \cup C_2\}$. Considering Definition 4, it is easy to verify that $CS < CS'$; moreover, $f^+(CS) \leq f^+(CS')$, since in CS' more (or equal) energy can be bought on the forward market at a cheaper price, hence the value of such coalition structure will be higher (if negative costs are considered). On the other hand, $f^-(CS) \geq f^-(CS')$, because $\kappa(C_1) + \kappa(C_2) \geq \kappa(C_1 \cup C_2)$. □

4.2.2 Edge sum with coordination cost function

Our second test case is represented by the *edge sum with coordination cost* function, in which every edge is associated to a real value by means of a function $w : E \rightarrow \Re$.[9] This value defines a pairwise relation between two nodes, that could represent how well (or bad) those agents perform together, or the cost of completing a particular coordination task in a robotic environment [4].

The characteristic function of a coalition is then calculated as the sum of the weights of the edges among its members. In order to have a better description of the management and communication costs in big coalitions, we also introduced a penalising factor $\kappa(C)$,[10] with the same definition given in the previous section:

$$v(C) = \sum_{e \in E(C)} w(e) + \kappa(C) \tag{4}$$

with $E(C)$ defined as the set of all the edges connecting any two members of the coalition C:

$$E(C) = \{(v_1, v_2) | v_1 \in C \wedge v_2 \in C\}$$

To help its characterisation as an $m + a$ function, it is useful to rewrite Equation 4 as:

$$v(C) = \sum_{e \in E(C)} \left[w^+(e) + w^-(e) \right] + \kappa(C)$$

with $w^+(e)$ and $w^-(e)$ defined as:

$$w^+(e) = \begin{cases} w(e) & \text{if } w(e) \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad w^-(e) = \begin{cases} w(e) & \text{if } w(e) < 0 \\ 0 & \text{otherwise} \end{cases}$$

[9]In our experiments we assigned a random weight $\in [-10, 10]$ to each edge.

[10]This was not considered in [13] and it is the element that violates the IDM property.

In other words, $\sum_{e \in E(C)} w^+(e)$ represents the sum of all the positive weights of the edges in $E(C)$, while $\sum_{e \in E(C)} w^-(e)$ represents the sum of the negative ones.

The *edge sum with coordination cost* function is then defined as:

$$f(CS) = \underbrace{\sum_{C \in CS} \left[\sum_{e \in E(C)} w^+(e) \right]}_{f^+(CS)} + \underbrace{\sum_{C \in CS} \left[\sum_{e \in E(C)} w^-(e) + \kappa(C) \right]}_{f^-(CS)}$$

PROPOSITION 6. *The edge sum with coordination cost function is an $m + a$ function.*

PROOF. The above equation highlights the f^+ and f^- components of this characteristic function. The first part is clearly monotonic: if we consider the same CS and CS' defined in the proof of Proposition 5, it is easy to verify that $f^+(CS) \leq f^+(CS')$, because in $f^+(CS')$ more (or none) positive weights are added. Since $f^-(CS)$ refers to the negative ones, then $f^-(CS) \geq f^-(CS')$ (note that $\sum_{C \in CS} \kappa(C)$ has already been proven to be antimonotonic). Hence, this function can be considered $m + a$, and the aforementioned bounding technique can be used. □

Moreover, in this case $f^+\left(\overline{CS}\right)$ can be calculated directly by adding all the positive edges that can still be contracted from CS.

In the next section we detail how these $m + a$ functions are used to compare CFSS against the current state-of-the-art algorithm for CSG on synergy graphs, i.e., DyCE.

5. EMPIRICAL EVALUATION

Having described and analysed our branch and bound approach for the CSG problem on synergy graphs, we now present the empirical evaluation. In what follows, we first discuss the methodology we use for comparison and then present the results obtained in the two domains described in Section 4.2.

5.1 Evaluation Methodology

The main goals of the empirical analysis are: i) to evaluate the performance of our approach in terms of runtime with respect to DyCE, ii) to evaluate the speedup that can be obtained by using multi-core machines, and iii) to evaluate the anytime performance and guarantees that our approach can provide when scaling to very large number of agents (i.e., more than 2700). Following [14], all our experiments have been made considering scale-free networks generated with the Barabási-Albert model [1] with the parameter $m \in \{1, 2, 3\}$. We compare our approach with DyCE in our two reference domains, measuring the runtime in seconds. Moreover, we implemented a multi-threaded version of our algorithm, and we analyse the speedup of such parallel version using Amdahl's law [2]. Finally, we run our approach on a very large set of agents by fixing the total runtime (100 seconds) and evaluating the approximation ratio, i.e., the ratio between the upper bound on the optimal solution and the value of the solution returned by our method. Our approach is implemented in C and executed on a machine with a 3.40GHz processor and 16 GB of memory. For DyCE we used the implementation provided by the authors of [14].

5.2 Comparison with DyCE

In our experiments, CFSS is found to outperform DyCE when coalition values are shaped by the above detailed benchmark functions (see Figures 4 and 5). Specifically, for the *edge sum with coordination cost* function, CFSS outperforms DyCE by 4 orders of magnitude for $m = 2$, and by 3 orders of magnitude for $m = 3$. Most probably this is due to the very tight bounding function used to prune search nodes.

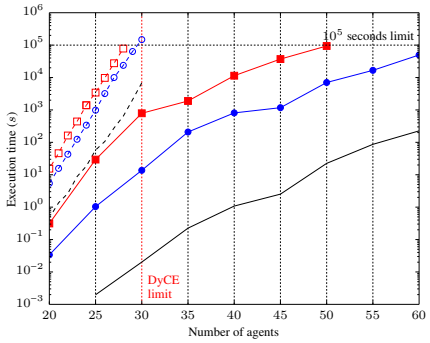


Figure 4: *Edge sum with coordination cost* function

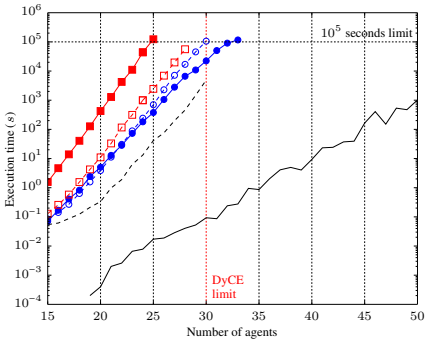


Figure 5: *Collective energy purchasing* function

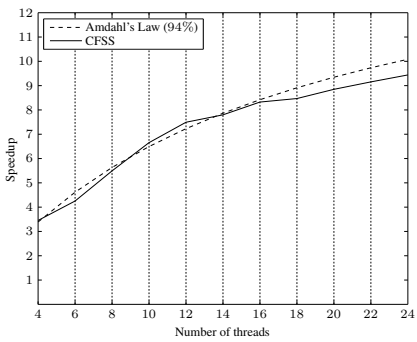
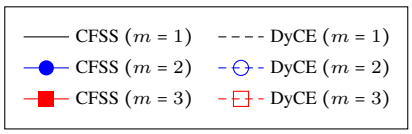


Figure 6: Multi-threading speedup

In the *collective energy purchasing* scenario with $m = 2$ CFSS is 4.7 times faster than DyCE for 30 agents, and with $m = 1$ it is at least 2 orders of magnitude faster. However, DyCE is significantly faster than CFSS with $m = 3$ (44 times). On both figures, we increased the number of agents until the execution time of the algorithm does not exceed 10^5 seconds (which represents a reasonable time limit for a single experiment). It is important to note that, in general, DyCE cannot scale over 30 agents (due to its exponential memory requirements), while CFSS does not have such limitation, hence it is possible to reach instances with thousands of agents, as shown in Section 5.4.



5.3 Parallel CFSS

Here we detail the parallelisation approach of the multi-threaded version of CFSS, analysing the speedup w.r.t. its serial version. Following [3], such parallel version is obtained by having different threads to search different branches of the search tree. The only required synchronisation point is the computation of the current best solution that must be read and updated by every thread. In particular, the distribution of the computational burden among the n available threads is done by considering the first i subtrees rooted in every node of the first generation (starting from the left) and assigning each of them to t_j threads ($1 \leq j \leq i$). The remaining rightmost subtrees are computed by a team of $n - \sum_{j=1}^{i} t_j$ threads using a dynamic schedule.[11] Parameters i and t_j are arbitrarily set, since it is assumed (and verified by an empirical analysis) that the distribution of the nodes over the search tree does not significantly vary among different instances. More advanced techniques found in literature, such as estimating the number of nodes in the search tree as suggested in [7], will be considered in the future.

We ran the algorithms on random instances with 27 agents and $m = 2$, using a machine with 2 Intel® Xeon® E5-2420. The speedup measured during these tests has been compared with the maximum theoretical one provided by Amdahl's Law, considering an estimated non-parallelisable part of 6%, due to memory allocation and thread initialisation. As can be seen in Figure 6, the actual speedup follows quite closely the theoretical one until the number of threads does not exceed 12, the number of physical cores. After that, Hyper-threading still provides some improvement, reaching a final speedup of 9.44 with all 24 threads active.

[11]Once a thread has completed the computation of one subtree, it starts with one of the remaining, if available.

5.4 Anytime performance

Figure 7 shows the value of the approximation ratio (i.e., the ratio between the provided bound and the solution returned by CFSS) obtained in 6 particular configurations for the *collective energy purchasing* scenario (using $n \in \{100, 500, 1000, 1500, 2000, 2732\}$ and $m = 4$) and considering a time limit of 100 seconds.[12] For each configuration, we plot the average and the standard error of the mean over 20 repetitions. The results show that, for 100 agents, the provided bound is only 4.7% higher than the solution found within the time limit, reaching a maximum of +11.65% when the entire dataset is considered. In the worst case, CFSS provides an approximation ratio of 1.12 and thus solutions that are at least 88% of the optimal. This confirms the effectiveness of this bounding technique applied to the energy domain, which allows us to provide solutions and quality guarantees for problems involving a very high number of agents.

In our experiments, the bound is assessed at the root, without any frontier expansion. In this way, the bound can be computed almost instantly thus devoting all the available runtime to the search for a solution. This choice is further motivated by the fact that, in this scenario, the bound improves of a negligible value in the first levels of the search tree, due to the particular definition of the characteristic function. More precisely, if we consider a frontier formed by the children of the root, in each of them the bound of f^- will improve by a factor of $2^\gamma - 2 \approx 1.5$ (i.e., the difference between the κ of the new coalition and the ones of the two merged singletons). On the other hand, the bound of f^+ will remain constant: in fact, since we are taking the maximum (i.e., the worst) bound at the frontier (as shown in Equation 2), the result of this maximisation will still be equal to the f^+ value of the grand coalition (as it was at the root), because in at least one of the children nodes the computation of $\overline{CS}$ will result in joining all the agents together. Hence, for this domain it is not reasonable to expand the frontier from the root, since the gain would be insignificant with respect to the additional computational cost.

6. CONCLUSIONS

In this paper we considered the coalition structure generation problem on synergy graphs, proposing a branch and bound solution approach (the CFSS algorithm) that can be applied to a general class of functions (the $m + a$ functions). Our empirical evaluation shows that CFSS outperforms DyCE, the state-of-the-art algorithm, when applied to two benchmark functions: the *edge sum with coordination cost* and the *collective energy purchasing* functions. Specifically, CFSS is at least 3 orders of magnitude faster than DyCE in the former scenario, while solving bigger instances in the latter one.

[12]Other values for m show a similar behaviour (not reported here).

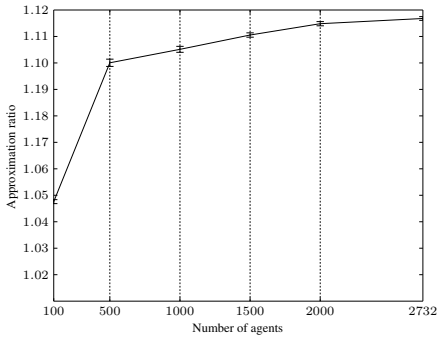


Figure 7: Approximation ratio

Moreover, our algorithm provides approximate solutions with good quality guarantees (i.e., with an approximation ratio of 1.12 in the worst case) for systems of unprecedented scale (i.e., more than 2700 agents).

Future work will look at improving the distribution of the computation in the parallel version of CFSS, focusing on different multi-threading models (e.g., GPGPU). Moreover we aim at extending the algorithm to work on other hard optimisation problems, such as the winner determination problem for combinatorial exchanges, where $m + a$ functions also apply.

7. ACKNOWLEDGEMENTS

Cerquides and Rodríguez-Aguilar are funded by projects COR (TIN 2012-38876-C02-01), AT (CSD2007-0022), and the Generalitat of Catalunya grant 2009-SGR-1434. This work was supported by the EPSRC-Funded ORCHID Project EP/I011587/1.

8. REFERENCES

[1] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, 2002.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS*, pages 483–485, 1967.

[3] D. A. Bader, W. E. Hart, and C. A. Phillips. Parallel Algorithm Design for Branch and Bound. In G, H J., editor, *Tutorials on Emerging Methodologies and Applications in Operations Research*, chapter 5, pages 5–5–44. 2005.

[4] P. Dasgupta, V. Ufimtsev, C. Nelson, and S. G. M. Hossain. Dynamic reconfiguration in modular robots using graph partitioning-based coalitions. In *AAMAS*, pages 121–128, 2012.

[5] A. Farinelli, M. Bicego, S. Ramchurn, and M. Zucchelli. C-link: A hierarchical clustering approach to large-scale near-optimal coalition formation. In *IJCAI*, pages 106–112, 2013.

[6] G. Grätzer. *Lattice Theory: Foundation*. Springer, 2011.

[7] L. H. S. Lelis, L. Otten, and R. Dechter. Predicting the size of depth-first branch and bound search trees. In *IJCAI*, 2013.

[8] T. Rahwan and N. R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *AAMAS*, pages 1417–1420, 2008.

[9] T. Rahwan, T. Michalak, and N. R. Jennings. A hybrid algorithm for coalition structure generation. In *AAAI*, pages 1443–1449, 2012.

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

[11] T. C. Service and J. A. Adams. Constant factor approximation algorithms for coalition structure generation. *JAAMAS*, 23(1):1–17, 2011.

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

[13] T. Voice, M. Polukarov, and N. Jennings. Coalition structure generation over graphs. *JAIR*, 45:165–196, 2012.

[14] T. Voice, S. Ramchurn, and N. Jennings. On coalition formation with sparse synergies. In *AAMAS*, pages 223–230, 2012.

APPENDIX

We report here the proof of Proposition 3 and some definitions we use in such proof. Let G^c be a 2-coloured graph, each vertex of such graph is labelled with a coalition. Recall that a red edge e between coalitions C_1 and C_2 encodes a constraint imposing that each pair of elements (c_1, c_2), where $c_1 \in C_1$ and $c_2 \in C_2$, do not lie in the same coalition.

DEFINITION 5. *Given a 2-coloured graph G^c, we say that a coalition structure is compatible with G^c if it can be obtained by contracting green edges and without breaking any of the constraints imposed by red edges.*

DEFINITION 6. *A green edge e is red-colourable w.r.t. CS if CS satisfies the constraint imposed by e when we color it red.*

DEFINITION 7. *Given a coalition structure CS and an edge e, we say that e is contractible w.r.t. CS if the coalition formed by contracting e is included into one of the coalitions of CS.*

Note that if CS is compatible with G^c each edge should be either red-colourable or contractible w.r.t. CS. Based on the previous definitions we can prove the following proposition:

PROPOSITION 3. *Given a 2-coloured graph G^c, the tree rooted at G^c contains all the coalition structures compatible with G^c, and each of them appears only once.*

PROOF. By induction on the number of green edges. If there is no green edge, then the tree has just one element which corresponds to the only coalition structure compatible with G^c. Assume that the statement is true for $n - 1$ green edges. Let G^c have n green edges and CS be a coalition structure compatible with G^c. If no edge in G^c is contractible with respect to CS, then CS is the coalition represented by G^c, and it cannot be in any of its children, because each of them contracts an edge in G^c. Thus CS appears in the tree rooted at G^c only once (at the root). Assume then that there is at least one green edge in G^c contractible w.r.t. CS. Then CS cannot be the coalition structure at the root. We would like to identify a child G' such that CS is compatible with G'. The first child of the root contracts an edge e. If e is contractible w.r.t. CS, then the first child of G^c is compatible with CS. Otherwise e is red-colourable w.r.t. CS. The same procedure goes on with the remaining children. Thus, by construction the root has three kind of children w.r.t. CS: some which contract a red-colourable edge, a single child G' that contracts a contractible edge and red-colours some red-colourable edges, and from there on some that red-color a contractible edge. It is easy to see that CS is compatible only with one child, namely G'. Now G' has at most $n - 1$ green edges and by induction CS must appear in that subtree only once. Thus, it appears in the tree rooted at G^c only once. □