On Coalition Formation with Sparse Synergies

Thomas Voice, Sarvapali D. Ramchurn, Nicholas R. Jennings
Agents, Interaction and Complexity Group
School of Electronics and Computer Science
University of Southampton, UK
{tdv,sdr,nrj}@ecs.soton.ac.uk

ABSTRACT

We consider coalition formation problems for agents with an underlying *synergistic graph*, where edges between agents represent some vital synergistic link, such as communication, trust, or physical constraints. A coalition is infeasible if its members do not form a connected subgraph, meaning parts of the coalition are isolated from others. Current state-of-the-art coalition formation algorithms are not designed for problems over synergistic graphs. They assume that *all* coalitions are feasible and so involve redundant computation when this is not the case. Accordingly, we propose algorithms, namely D-SlyCE and DyCE, to enumerate all feasible coalitions in a distributed fashion and find the optimal feasible coalition structure respectively. When evaluated on a variety of synergistic graphs, D-SlyCE is up to 660 times faster while DyCE is up to 7×10^4 times faster than the state-of-the-art algorithms. For particular classes of graphs, D-SlyCE is the first to enumerate valid coalition values for up to 50 agents and DyCE is the first algorithm to find the optimal coalition structure for up to 30 agents in minutes as opposed to months for previous algorithms.

Categories and Subject Descriptors

I.2.11 [**Distributed AI**]: Multi-Agent Systems

General Terms

Algorithms

Keywords

Coalition Formation, Networks

1. INTRODUCTION

Coalition formation is one of the fundamental approaches in multi-agent systems for establishing collaborations among agents, each with individual objectives and properties [10]. Key computational tasks in coalition formation involve calculating the value of each potential coalition and finding the best set of coalitions to be formed (i.e., the coalition structure generation problem) [9]. To date, existing work has typically studied coalition formation in abstract settings, where every set of agents can be considered to be a potential coalition [3, 5, 7, 10]. Such approaches are limited to solving problems involving at most 30 agents (28 in the case of coalition structure generation). However, this may be severely inefficient for problems where only some coalitions are feasible. We believe that many real-world applications involve constraints as to which coalitions can exist based on sparse synergies (i.e., necessary peer-to-peer connections) between individual agents in the system [2, 8]. These constraints may be due to communication constraints (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 or have life-saving capabilities that match only a limited number of other responders).

Against this background, in this paper we provide efficient algorithms to form coalitions in environments that have an underlying *synergistic graph*, where edges in the graph represent vital synergistic links between agents that constrain which coalitions may form. In more detail, we consider coalition formation problems where each coalition is only feasible if its members form the vertices of a connected subgraph of the constraining graph. By taking advantage of these structures, we aim to improve on the performance of the current state-of-the-art coalition formation algorithms which were not designed with synergistic graphs in mind.

In this setting,[1] we advance the state-of-the-art in the following ways. First, we provide a new exact coalition enumeration procedure, SlyCE (Sequentially connected Coalition Enumeration). Second, we provide D-SlyCE (Distributed SlyCE), a variant which aims to evenly distribute the SlyCE computation amongst agents at negligible communication and computation cost. Third, building upon SlyCE, we provide a complete algorithm, DyCE (Dynamic programming for optimal connected Coalition structure Evaluation) to find the optimal coalition structure. Fourth, we benchmark our solutions against the state-of-the-art algorithms and show that D-SlyCE can be up to 660 times faster and DyCE can be up to 7×10^4 times faster. Moreover, for particular classes of graphs, our algorithms are the first to be able to enumerate coalition values for 50 agents and find the optimal coalition structure for up to 30 agents within minutes compared to months for the state-of-the-art.

The rest of the paper is structured as follows. In Section 2 we discuss related work. Then, in Section 3, we formally describe the problem of coalition formation with sparse synergies. In Section 4 we describe SlyCE and discuss its prop-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

[1]The reader is referred to [2, 4] for a cooperative game theoretic analysis (which is not the goal of this paper) of our setting for the case where agents are self-interested.

erties. We then propose the D-SlyCE algorithm in Section 5, to distribute the computation of SlyCE amongst agents fairly. Then, we turn to the coalition structure generation problem in Section 6 and describe our solution, DyCE. We empirically benchmark D-SlyCE and DyCE in Section 7. Lastly, Section 8 concludes.

2. BACKGROUND

The formation of coalitions within synergistic graphs (as defined in this paper) has typically been studied in the field of economics where the focus is on the definition of cooperative game-theoretic solutions [2, 4]. In contrast, this paper is concerned specifically with the coalition value calculation and coalition structure generation problems over synergistic graphs.

The challenge in coalition value calculation is to enumerate all the feasible coalitions and efficiently distribute this computation among the agents. The main algorithms that deal with this specifically are Shehory et al.'s [10] and DCVC [5]. The latter is the fastest and is able to enumerate coalitions for up to 30 agents in reasonable time. Under DCVC, for each $s = 1, \dots n$, each agent calculates the coalition values for an nth share of a lexicographically ordered list of all coalitions of size s. DCVC does this in such a way that every coalition value is calculated precisely once, and no agent communication is required. While DCVC also efficiently recomputes coalition values where individual agents may be removed or new agents added dynamically, it has no way of avoiding infeasible coalitions.

Turning to the coalition structure generation problem (effectively a set partitioning problem), a number of recent works in this area have attempted to solve the problem in both centralised and distributed ways along with providing anytime quality guarantees [9, 6, 3, 7]. In particular, we note the two approaches taken in this area (i) low-complexity ($O(3^n)$) complete algorithms based on dynamic programming, such as DP and IDP, that have guaranteed run-times for arbitrary coalition value distributions (ii) high worst-case complexity ($O(n^n)$) complete algorithms, based on branch-and-bound techniques, that have anytime properties but heavily depend on the coalition value distribution in order to establish bounds on segments of the search space and therefore prune huge parts during the search process. While the latter algorithms have been shown to be faster than the former given specific coalition value distributions, their approach is undefined for cases where only some coalitions are feasible. Simply treating infeasible coalitions as being feasible but with value $-\infty$ would not be suitable, as they use averages of coalition values to compute lower bounds. In contrast, IDP (the fastest dynamic programming approach) makes no requirements on coalition value distributions and thus, we can treat infeasible coalitions as being feasible coalitions with value $-\infty$. This has no effect on the runtime of IDP, which only depends on n. We therefore use IDP to benchmark DyCE.

Another candidate set of techniques to solve the coalition structure generation problem is Integer Programming based solvers. These have been shown to be particular inefficient when all coalitions are deemed feasible (due to the size of the input) but tend to be very efficient in solving other combinatorial optimisation problems (combinatorial auctions or set packing problems). Hence, we also benchmark DyCE against IBM's ILOG CPLEX in Section 7.

Finally, another related algorithm to ours is Rahwan et al.'s CCF algorithm [8] which does solve coalition structure generation problems with constraints on which coalitions can be formed but they consider a different constraint model to ours.

3. MODEL

In this section, we model the problem of Coalition Formation with Sparse Synergies (CFSS). We identify a set of agents with the set of positive integers $I = \{1, 2, ..., n\}$, where n is the total number of agents in the system. Agents can form coalitions $C \subseteq I$, however the set of feasible coalitions, $\mathcal{C}$, is constrained by a graph $G = (I, E)$, where E is a set of edges between agents. We consider the situation where a coalition of agents C is feasible if and only if there exists a connected subgraph $G' = (C, E')$ with edges $E' \subseteq E$ and vertex set exactly equal to C. By forming coalitions, agents can perform tasks in the environment and their effectiveness in performing such tasks depends on the synergies in their abilities generated by them being in the same coalition. To capture such synergies, we define the value of a coalition using a characteristic function $v : \mathcal{C} \to \mathbb{R}$. The function $v(\cdot)$ may be arbitrarily defined according to the domain, however coalition values are always independent of any other coalitions that may exist. We address the problem of enumerating and evaluating all feasible coalitions for CFSS problems by providing the SlyCE and D-SlyCE algorithms, in Section 4. Now, given the coalition values, a key challenge is to choose the best *coalition structure*, that is, the best set of disjoint coalitions that collectively cover all agents. This problem is termed the coalition structure generation problem. To find the optimal coalition structure in CFSS, we need to consider the set of *feasible coalition structures*, $\mathcal{F}(G)$, that is, the set of coalition structures that only contain feasible coalitions. The coalition structure generation problem which DyCE, given in Section 6, attempts to solve is then to find $\arg\max_{CS \in \mathcal{F}(G)} \sum_{C \in CS} v(C)$. In the next section, we proceed with our description of the SlyCE and D-SlyCE algorithms.

4. THE SLYCE ALGORITHM

The SlyCE algorithm proceeds by conducting a depth first search over a tree representation of the set of feasible coalitions in G. At each point in the search, SlyCE maintains a root node, R, which represents the current coalition being evaluated, and a frontier set, F, that contains agents that may be added to the root node to form a feasible coalition. The individual steps of SlyCE are described in Algorithm 1, where we use the function $\overline{N}(\cdot, \cdot)$ which we define to be:

$$\overline{N}(F, R) = \{j | j > \min(R \cup F)\} \cap N(F) \setminus (N(R) \cup R \cup F),$$

for $N(\cdot)$ denoting the set of neighbours of a subset of I in G, that is, $N(R) = \{j : \exists i \in R, (i, j) \in E\}$. The algorithm recursively traverses the search tree in two phases. The first phase, (lines 2 and 3), involves generating a new root node R by combining the current root node with a subset of the frontier set. Having updated the root node, (and evaluated it on line 4), in the second phase, (line 5), a new frontier set is created by choosing neighbouring agents to expand the root with. The algorithm then calls itself (line 6) to continue the recursive search. The key point here is that SlyCE chooses those agents in the root creation and expansion phases so that it will not recompute an already computed coalition. We elaborate on these two phases next.

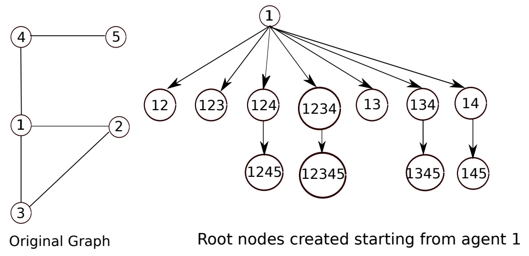

Figure 1: Root expansion process emanating from agent 1.

1. Root expansion — given an existing root $R \subseteq I$ and a frontier set $F \subseteq I$, form a new root node R' such that $R' = R \cup F^*$ where $F^* \subseteq F$. Simply put, given an existing root node, we expand it to include agents that are not currently included and that lie at the frontier of the existing root. Figure 1 shows an example of this.

2. Frontier expansion — given a new root node $R' = R \cup F^*$ formed from existing root node $R \subseteq I$ and set F^*, we set the new frontier set F' to be the set of agents j such that, $j > \min(R')$, $j \notin R'$, $j \notin N(i)$ for all $i \in R$, and $j \in N(k)$ for some $k \in F^*$. That is, the ids of all agents in F' should be numerically higher than the id of the agent with the lowest id in R', and F' should consist of agents who are neighbours of F^*, but are neither neighbours nor members of R'.

Figure 1 shows how SlyCE expands to other root nodes, starting with $R = \{1\}$. These are evaluated and their value stored. The same process, starting from the other agents, will generate totally different coalitions (e.g., 5 will only generate $\{5\}$, while 4 will generate $\{4\}$ and $\{4, 5\}$).

Algorithm 1 $slyce(R, F, m)$

```
1: if $F \neq \emptyset$ and $m > 0$ then
2:     for all $F^* \subseteq F$ with $1 \leq |F^*| \leq m$ do
3:         $R' \leftarrow F^* \cup R$ {Generate new root node.}
4:         compute and store $v(R')$ {Evaluate new root node.}
5:         $F' \leftarrow \overline{N}(F^*, R)$ {Generate new frontier set}
6:         $slyce(R', F', m - |F^*|)$ {Recursive call.}
7:     end for
8: end if
```

In order to enumerate all coalitions of size up to m, SlyCE should search from each agent singleton, which can be achieved by calling $slyce(\emptyset, \{i\}, m)$ for all $i \in I$.

4.1 Properties of SlyCE

We now elaborate on the key properties of SlyCE, in particular its correctness, completeness, and non-redundancy. We first note that SlyCE only evaluates *feasible coalitions*. To be more precise, we claim that, provided that R is feasible and $F \subseteq N(R)$, then for any m, $slyce(R, F, m)$ only calls $v(C)$ for feasible coalitions $C \subseteq I$. To see this, note that for such F and R, for any $F^* \subseteq F$, $F^* \subseteq N(R)$ and so when $v(R')$ is evaluated, $R' = F^* \cup R$ is a feasible coalition. Furthermore, for $F' = \overline{N}(F^*, R)$, $F' \subseteq N(F^*) \subseteq N(F^* \cup R) = N(R')$, and so the recursive call to $slyce(R', F', m - |F^*|)$ also satisfies $F' \subseteq N(R')$. This recursively proves our claim. It remains to note that when $slyce(\emptyset, \{i\}, m)$ is called for some $i \in I$, after the first level of recursion, all subsequent F and R satisfy $F \subseteq N(R)$. Hence, SlyCE only evaluates feasible coalitions and is thus, *correct*. It is also important to show that SlyCE is *complete*, that is, that it can enumerate all feasible coalitions. We do so in the following proposition.

PROPOSITION 1. *For each $i \in I$, and $m \geq 1$, if the SlyCE algorithm is called with parameters $slyce(\emptyset, \{i\}, m)$ then every feasible coalition $C \subseteq I$ such that $i = \min(C)$ and $1 \leq |C| \leq m$ will be evaluated.*

PROOF. First, we should note that since at least one agent is added to R before it is passed to the next level of recursion, we have that at the lth level of recursion, $|R| \geq l$. Thus, the algorithm cannot go beyond the mth level of recursion, and must terminate. This means that if $slyce(R, F, k)$ is evaluated, the main **for all** loop reaches every subset of F of size up to k.

Now, suppose C is a feasible coalition with $i = \min(C)$ and $1 \leq |C| \leq m$. Let $E' \subseteq E$ be the set of all edges $(j, k) \in E$ such that $j, k \in C$. For all $j, k \in C$, let $d(j, k)$ be the number of vertices in the minimal path between j and k over the subgraph $G' = (E', C)$, and let s be the maximum value of $d(i, j)$ for $j \in C$. If $s = 1$ then $C = \{i\}$ and $v(C)$ is evaluated during the initial call of $slyce(\emptyset, \{i\}, m)$.

We now consider the case where $s > 1$. Let us define the sets $F_1, F_2, \ldots, F_s$ as being $F_l = \{j : d(i, j) = l\}$, for $l = 1, \ldots, s$, and $H_0, H_1, \ldots, H_s$ as being $H_0 = \emptyset$, $H_l = \{j : d(i, j) \leq l\}$ for $l = 1, \ldots, s$. Consider the set F_{l+1} for some $l = 1, \ldots, m - 1$. Since $i = \min(C)$, we must have that, for all $j \in F_{l+1}$, $j > i$. Furthermore, for all $j \in F_{l+1}$, j cannot lie in H_{l-1} nor can it be a neighbour of H_{l-1}, otherwise there would be a path shorter than $l + 1$ from j to i through G'. So, the first step along the path of length $l + 1$ from j to i through G' must be from j to an agent in F_l and hence $j \in N(F_l)$. Thus, $F_{l+1} \subseteq \overline{N}(F_l, H_{l-1})$.

We now claim that during the operation of $slyce(\emptyset, \{i\}, m)$, $slyce(H_l, \overline{N}(F_l, H_{l-1}), m - |H_l|)$ will be called for all $l = 1, \ldots, s - 1$. This can be proved inductively. For $l = 1$, we have $H_1 = F_1 = \{i\}$, and $H_0 = \emptyset$. From the definition of the algorithm, $slyce(\{i\}, \overline{N}(\{i\}, \emptyset), m - 1)$ is called as part of the main loop of $slyce(\emptyset, \{i\}, m)$. Now suppose that for some $l < s$, $slyce(i, H_l, \overline{N}(F_l, H_{l-1}))$ is called. Since $F_{l+1} \cup H_l \subseteq C$, then $|F_{l+1}| + |H_l| \leq s$ and so $|F_{l+1}| \leq s - |H_l|$. As shown above, $F_{l+1} \subseteq \overline{N}(F_l, H_{l-1})$, and so $slyce(i, H_l \cup F_{l+1}, \overline{N}(F_{l+1}, H_l))$ must be called as part of the operation of $slyce(i, H_l, \overline{N}(F_l, H_{l-1}))$. As $H_l \cup F_{l+1} = H_{l+1}$ this proves our claim by induction. It remains to note, when $slyce(H_{m-1}, \overline{N}(F_{m-1}, H_{m-2}), m - |H_{m-1}|)$ is called, the value of $v(F_m \cup H_{m-1}) = v(C)$ is evaluated, as required. $\square$

Thus, for every feasible coalition $C \subseteq I$, $v(C)$ is evaluated during the operation of $slyce(\emptyset, \{\min(C)\}, n)$. It is also true that SlyCE is *non-redundant*, that is, during its operation, it never evaluates the same coalition twice. We can see that this is true by noting that, when a subset $F^* \subseteq F$ is chosen during SlyCE and $R \cup F^*$ is set as a root, then, from that depth of recursion onwards, the root will always contain R, and thus the frontier nodes will never again contain any agent in $F \subseteq N(R)$. If we have two paths through the root expansion process that diverge, at the point of divergence, they must choose different root expansions, meaning that at least one of the paths adds at least one agent that can never be added at a later stage in the other path. Thus, there is no way to reach the same coalition twice through the recursive calls in SlyCE.

So, if the SlyCE algorithm is run as $\mathit{slyce}(\emptyset, \{i\}, m)$ for each agent $i \in I$, then for every feasible coalition C ($|C| \leq m$), $v(C)$ is evaluated and stored precisely once, and no infeasible coalitions are evaluated. Indeed, this can be run in parallel with each agent $i \in I$ computing $\mathit{slyce}(\emptyset, \{i\}, m)$. Thus, SlyCE may be implemented as a distributed algorithm. Moreover we can use SlyCE to do more than simply enumerate all feasible coalitions. As we describe in the following subsection, we can use SlyCE to cycle through feasible coalitions that are subsets of a given feasible coalition, a process which will be useful in DyCE (see Section 6).

4.2 SlyCE Over Coalition Subsets

The SlyCE algorithm, as we have defined it, enumerates all feasible coalitions of size up to some limit m for a given graph $G = (I, E)$. However, given a feasible coalition $C \subseteq I$, we can also use SlyCE to enumerate all feasible coalitions that are subsets of C. We simply have to apply SlyCE to the subgraph that G induces over the nodes of C. Since this is, itself, a connected graph, the desirable properties of SlyCE discussed above will still hold. For notational convenience, in preparation for our later description of DyCE, we define the function $\mathit{nextslyce}(\cdot, \cdot, \cdot)$ to be such that, for feasible coalitions $C' \subseteq C \subseteq I$ and integer m, with $|C'| \leq m$, $\mathit{nextslyce}(C', C, m)$ returns the subset of C that would follow C' during the process of SlyCE iterating through all feasible coalition subsets of C of size at most m. Thus, if we begin with $C' = \{a\}$, for $a \in C$, then repeated calls to $C' \leftarrow \mathit{nextslyce}(C', C, m)$ will conduct the entire SlyCE iteration $\mathit{slyce}(\emptyset, \{a\}, m)$ over the subgraph induced by G on C. If C' is the last in this SlyCE iteration, we stipulate that $\mathit{nextslyce}(C', C, m)$ should return the empty set.

Having elaborated on the properties of SlyCE above, we next discuss how the SlyCE computation may be distributed more fairly amongst agents, using our D-SlyCE algorithm.

5. THE D-SLYCE ALGORITHM

As noted above, SlyCE may be run as a distributed algorithm by having each agent $i \in I$ compute $\mathit{slyce}(\emptyset, \{i\}, m)$. However, in that case, agents with higher ids will have fewer computations than those with lower ids. Indeed, the agent with the highest id, only computes the value of one coalition, the singleton containing itself (agent 5 in Figure 1). In contrast, the agent with the lowest id would evaluate all feasible coalitions that contain it. Accordingly, in this section, we propose the Distributed SlyCE algorithm (D-SlyCE), under which the SlyCE algorithm is distributed across the agents such that the task assigned to each agent is determined with no communication overhead. Furthermore it generates no repeated coalition evaluations and spreads tasks reasonably fairly amongst agents (as shown in Section 7).[2] The full operation of D-SlyCE is defined in Algorithm 2. It uses some combinatorial functions, which we define here. We use $\binom{m}{r}$ to denote the number of subsets of size r that may be drawn from a set of m unique agents, so $\binom{m}{r} = m!/r!(m-r)!$. For any set of agents C, for any $r = 1, \ldots, |C|$ and $i = 1, \ldots, \binom{|C|}{r}$ we let $L(C, i, r)$ denote the ith element of the set of all subsets of $|C|$ of size r under the lexicographic ordering induced by the ids of the agents. For any such i, $L(C, i, r)$ can be found easily, and if $L(C, i, r)$ is known then $L(C, i+1, r)$ can be found quickly, using known lexicographic techniques that are described in [5].

Algorithm 2 $\mathit{dslyce}(i, m)$

```
 1: compute and store $v(\{i\})$
 2: $x \leftarrow i - 1$
 3: for all agents $a \in I$ do
 4:     $F \leftarrow \overline{N}(\{a\}, \emptyset)$ {Assign frontier set.}
 5:     for all $r = 1 \ldots, \min(|F|, m)$ do
 6:         $j \leftarrow \lfloor \binom{|F|}{r} \times x/n \rfloor$ {Index of beginning of share.}
 7:         while $j < \lfloor \binom{|F|}{r} \times (x+1)/n \rfloor$ do
 8:             $F' \leftarrow L(F, j+1, r)$ {Set next addition to root.}
 9:             $j \leftarrow j + 1$
10:             compute and store $v(F' \cup \{a\})$ {As in SlyCE.}
11:             $\mathit{slyce}(F' \cup \{a\}, \overline{N}(F', \{a\}), m - (r+1))$
12:         end while
13:         $x \leftarrow (x+1) \bmod n$ {Rotate share allocation.}
14:     end for
15: end for
```

D-SlyCE is called with the agent's id and the maximum coalition size m it should generate (in a similar vein to DCVC), such that if all coalitions must be generated, $m = n$. It then divides up the computation of the SlyCE algorithm amongst the agents by splitting up the operation of the first two levels of recursion of $\mathit{slyce}(\emptyset, \{i\}, m)$ for all $i \in I$. This division is done first by having each agent $a \in I$ evaluate $v(\{a\})$ (line 1). Then, each agent $a \in I$ is apportioned an approximately equal share of the subsets of $\overline{N}(\{i\}, \emptyset)$ for each $i \in I$ (line 6–7), that a must evaluate as per the main loop in $\mathit{slyce}(\{i\}, \overline{N}(\{i\}, \emptyset), m-1)$ (lines 10–11). Since larger frontier sets are likely to represent a greater computational burden, in order to make sure a gets an approximately fair allocation of tasks, for each agent i, a is given a $1/n$ share of the lexicographically ordered list of subsets of $\overline{N}(\{i\}, \emptyset)$ of size r for all $r = 1, \ldots, |\overline{N}(\{i\}, \emptyset)|$ (lines 6–7). Each time this dividing process occurs, the algorithm deterministically rotates the order in which shares are allocated to agents (line 13). This is because lexicographically earlier frontier sets contain lower id numbered agents, and thus are likely to represent a greater computational burden. As the full operation of D-SlyCE covers precisely the calls to $v(\cdot)$ and $\mathit{slyce}(\cdot, \cdot, \cdot)$ as in the first two levels of recursion of $\mathit{slyce}(\emptyset, \{i\}, m)$, for all $i \in I$ (lines 10–11), then D-SlyCE covers exactly the same calls to $v(\cdot)$ as SlyCE. This means that D-SlyCE has the same desirable properties as SlyCE, that is, D-SlyCE is correct, complete and non-redundant.

An example of how the computational burden of SlyCE is

[2] D-SlyCE is suitable for problems where evaluating $v(\cdot)$ is not computationally intensive. If the operation of SlyCE is a negligible overhead compared to the evaluation of $v(\cdot)$ for every feasible coalition, then the fairest way to distribute these evaluations would be for each agent to use SlyCE to create lists of all feasible coalitions, grouped together by size, and then evaluate a fair share from each list.

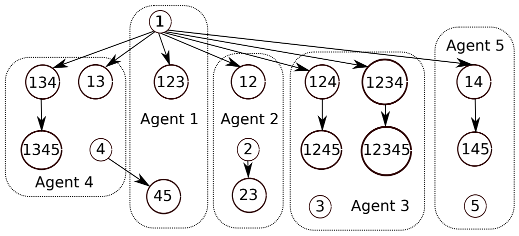

Figure 2: Distribution of coalition evaluation calculations under D-SlyCE.

divided up by D-SlyCE is given in Figure 2, which shows the distribution of coalition valuations prescribed by D-SlyCE for the graph given in Figure 1. Thus, we see that the first generation of descendants from each of the single node sets are divided up between the agents. Note that due to the way the shares of the lexicographical lists of subsets are rotated, agent 1 ends up computing $\{4, 5\}$. In general, the process results in a much fairer distribution than if each agent was simply allocated the frontier expansion tree that branches from their respective singleton coalition (i.e., $\{2\}$ and $\{2, 3\}$ for agent 2 or $\{5\}$ for agent 5 in SlyCE). Indeed, on complete graphs, D-SlyCE mimics the operation of DCVC, and thus is optimally fair.

Since single vertex root nodes with the lowest id represent the greatest computational burden, and the D-SlyCE distribution methods are most effective for root nodes with many frontier sets, we can further increase the fairness of the distribution by ensuring that the agents with highest degree have the lowest id number. This may be done in a fast, deterministic manner by each agent before it runs *dslyce*. In the next section, we move to solve the problem of coalition structure generation in synergistic graphs, using SlyCE as an important building block of DyCE.

6. THE DYCE ALGORITHM

In this section, we describe the operation of the DyCE algorithm, and prove its correctness. DyCE operates in a similar manner to IDP (see Section 2). However, it speeds up the search for the optimal coalition structure by using SlyCE to enumerate all feasible coalitions, thus allowing it to ignore infeasible coalitions during the search. DyCE further improves upon IDP by using SlyCE when evaluating feasible coalition subsets of coalitions (using the procedure described in Section 4.2).

DyCE solves a CFSS coalition structure generation problem by exploring the edges in the *feasible coalition structure graph* of the synergy graph. For any graph $G = (I, E)$, we define the *feasible coalition structure graph* of G, to be the directed graph with node set $\mathcal{F}(G)$, and directed edges $\mathcal{E}(G)$, where there is an edge from each $CS \in \mathcal{F}(G)$ to every $CS^* \in \mathcal{F}(G)$ that can be formed by splitting a coalition in CS into two smaller feasible coalitions. That is, $\mathcal{E}(G)$ is equal to the set of (CS, CS^*) where $CS \setminus \{C\} = CS^* \setminus \{C', C \setminus C'\}$ for some $C \in CS$ and some $C' \subset C$. Crucially, to improve tractability, we consider a reduced set of edges, $\mathcal{E}^*(G) \subset \mathcal{E}(G)$, where for $(CS, CS^*) \in \mathcal{E}(G)$ with $CS \setminus \{C\} = CS^* \setminus \{C', C \setminus C'\}$, $(CS, CS^*) \in \mathcal{E}^*(G)$ if $|C'| \leq n - |C|$ or $C = I$. In [7], a similar coalition structure graph is considered, and it is shown that it is possible to reach any coalition structure from $\{I\}$ along a similar reduced set of edges. DyCE requires the corresponding result to be true for our feasible coalition structure graph. This cannot be derived from the result in [7] because, although their set of reduced edges is similar to ours, paths along their coalition structure graph may pass through intermediary coalition structures that contain *infeasible coalitions* and thus are not in $\mathcal{F}(G)$. Instead, we prove the desired result in the following theorem.

THEOREM 1. *For every feasible coalition structure $CS \in \mathcal{F}(G)$, there is a directed path of edges from $\mathcal{E}^*(G)$ that leads from $\{I\}$ to CS.*

PROOF. Let us suppose this result does not hold. Let CS be the coalition structure with minimal $|CS|$ out of those $CS \in \mathcal{F}(G)$ that cannot be reached from $\{I\}$ by following the directed edges in $\mathcal{E}^*(G)$. We must have that $|CS| > 1$, otherwise CS would equal $\{I\}$. Further, $|CS| > 2$, as otherwise $(\{I\}, CS)$ would be in $\mathcal{E}^*(G)$.

Let C be the coalition in CS with maximal $|C|$. Suppose there were two coalitions $C', C'' \in CS \setminus \{C\}$ such that there is at least one edge in G between C' and C''. Let CS^* be,

$$CS^* = \{C' \cup C''\} \cup CS \setminus \{C', C''\}.$$

Since there is at least one edge between C and C', $C' \cup C''$ is feasible and so $CS^* \in \mathcal{F}(G)$. Since $|CS^*| < |CS|$ we must be able to get from $\{I\}$ to CS^* by following edges in $\mathcal{E}^*(G)$, by choice of CS. However, since $|C'| \leq |C|$ and $C \subset (I \setminus (C' \cup C''))$ we must have $|C'| \leq n - |C' \cup C''|$ and so $(CS^*, CS) \in \mathcal{E}^*(G)$. This leads to a contradiction, as we can now go from $\{I\}$ to CS^* and then to CS.

Thus, no two coalitions in $CS \setminus \{C\}$ have any edges between them. Since G is connected this means that there must be at least one edge between C and each coalition in $CS \setminus \{C\}$. Now, let C' be the coalition in $CS \setminus \{C\}$ with minimal $|C'|$. Let CS^* be,

$$CS^* = \{C \cup C'\} \cup CS \setminus \{C, C'\}.$$

Since there is at least one edge between C and C', $C \cup C'$ is feasible and so $CS^* \in \mathcal{F}(G)$. As $|CS^*| < |CS|$, by choice of CS there must be a directed path from $\{I\}$ to CS^* consisting of edges from $\mathcal{E}^*(G)$. As noted above, $|CS| > 2$, and so there must be some $C'' \in CS \setminus \{C, C'\}$. Since C'' is disjoint from $C \cup C'$, we must have $n - |C \cup C'| \geq |C''| \geq |C'|$, by choice of C'. Thus, $(CS^*, CS) \in \mathcal{E}^*(G)$, which leads to a contradiction, as now we can go from $\{I\}$ to CS^* and then to CS. Hence, no such CS can exist and the result follows. $\square$

In order to explain how the feasible coalition structure graph relates to the operation of DyCE, we first need some definitions. For a graph $G = (I, E)$, and for any coalition structure $CS \in \mathcal{F}(G)$, let $\mathcal{D}(CS)$ be the set of coalition structures that can be reached from CS along directed paths made up of edges from $\mathcal{E}^*(G)$ (including CS itself). Theorem 1 shows that $\mathcal{D}(\{I\}) = \mathcal{F}(G)$. For a graph $G = (I, E)$ and a coalition valuation function $v(\cdot)$, we define the function $w : \mathcal{C} \to \mathbb{R}$ so that for any feasible coalition C, $w(C)$ is set equal to the maximum of: $\sum_{C' \in CS^* : C' \subseteq C} v(C')$, over $CS^* \in \mathcal{D}(CS)$, for $CS \in \mathcal{F}(G)$ such that $C \in CS$. Informally, if C is in some feasible coalition structure CS, then $w(C)$ is the maximum total value of subsets of C for coalition structures in $\mathcal{D}(CS)$. This is well defined and does not depend on choice of CS containing C because the set of splits that can occur as you move along edges in $\mathcal{E}^*(G)$

only depend on the coalition being split. From Theorem 1, $w(I)$ must be equal to the maximum of $\sum_{C' \in CS} v(C)$ over all $CS \in \mathcal{F}(G)$. So, $w(I)$ is the optimal feasible coalition structure value. For notational convenience if $C \subset I$ is not a feasible coalition then we let $w(C) = -\infty$.

The full operation of DyCE is described in Algorithm 3. DyCE proceeds by recursively calculating $w(\cdot)$ for all feasible coalitions, in increasing order of size. First, a memory block large enough to contain all coalitions of I is initialised, with every entry being given the value $-\infty$ (Line 1–3). Then, DyCE uses SlyCE to evaluate each feasible coalition (using procedure *nextslyce* described in Section 4.2) and its value is stored in memory, (lines 5–9). After initialisation, DyCE goes through each coalition of size s for $s = 1, 2, \ldots n$ (lines 10–29). For each coalition C of size s, if C is feasible, then DyCE calculates $w(C)$ and replaces the value of $v(C)$ in memory with $w(C)$. In order to calculate $w(C)$ for a coalition $C \neq I$, DyCE uses SlyCE to cycle through every feasible coalition subset $C' \subset C$ that is smaller than $n-|C|$ and $|C|/2$ (lines 11–14), and evaluates $w(C')+w(C \setminus C')$ (line 24). The value of $w(C)$ is then the maximum of these values and $v(C)$ (line 23). Note, we only go up to subsets C' of size $|C|/2$ as, for larger C', if $C \setminus C'$ is feasible then $w(C \setminus C') + w(C')$ will be evaluated anyway. DyCE also records the most recent set $C' \subset C$ such that $w(C) = w(C') + w(C \setminus C')$, if such exists. Lastly, $w(I)$ is calculated similarly by cycling through subsets $C \subset I$ with $|C| \leq n/2$ and maximising $w(C) + w(I \setminus C)$.

The optimal coalition structure is then determined (using Algorithm 4) from $w(I)$ by starting with $CS = \{I\}$ and then recursively replacing each coalition $C \in CS$ with C' and $C \setminus C'$ where $w(C) = w(C') + w(C \setminus C')$, if such exist. Note, if no such replacement is possible for a coalition $C \in CS$, then we must have that $w(C) = v(C)$. Furthermore, each time a replacement occurs, $\sum_{C \in CS} w(C)$ does not change. Hence the resulting coalition structure will be such that $w(C) = v(C)$ for all $C \in CS$ (since the algorithm only stops when no further subdivisions are possible) and $\sum_{C \in CS} v(CS) = \sum_{C \in CS} w(CS) = w(I)$. Hence CS is optimal. We can reduce memory requirements by not storing the particular subsets required to calculate this optimum, but instead, finding them once the values of $w(\cdot)$ have been found by searching through the subsets, as in [7].

To give an example of DyCE in operation, consider the graph given in Figure 1 with some coalition valuation function $v(\cdot)$. DyCE begins by using SlyCE to go through each feasible coalition C, evaluating $v(C)$ and recording the result. Then it goes through all sets of size $m = 1, 2, 3, 4, 5$ evaluating $w(C)$ for each feasible C, ignoring C if it is infeasible (i.e., if $v(C)$ was not recorded during the initial phase). For $|C| = 1$, $w(C) = v(C)$, and so this phase does not require any work. For $1 < |C| \leq 4$, DyCE must go through some subsets of C in order to evaluate $w(C)$. DyCE has to evaluate for subsets of size up to $\min(|C|/2, n - |C|)$ which is < 2 for such C. Thus $w(C)$ is the maximum of $v(C)$ and $w(\{a\}) + w(C \setminus \{a\})$ for each agent $a \in C$ such that $C \setminus \{a\}$ is feasible. Hence, for example, $w(\{1,4,5\}) = \max(v(\{1,4,5\}), w(\{1\})+w(\{4,5\}), w(\{5\})+w(\{1,4\}))$. For $C = \{1,2,3,4,5\} = I$, we evaluate $w(C') + w(C \setminus C')$ for all feasible coalitions C' such that $I \setminus C'$ is also a feasible coalition and $|C'| \leq n/2 = 2.5$. This is the same as evaluating $w(C') + w(C'')$ for all disjoint pairs of feasible coalitions C', C'' with $C' \cup C'' = I$. To give an example of why this process works, let us consider the set $C = \{1,2,3,4\}$. Now,

Algorithm 3 *dyce*()

```
1: for all $C \subset I$ do
2:   $W(C) \leftarrow -\infty$ {Initialise memory.}
3: end for
4: for all $a \in I$ do
5:   $C \leftarrow \{a\}$
6:   while $C \neq \emptyset$ do
7:     $W(C) \leftarrow v(C)$ {Store coalition value.}
8:     $B(C) \leftarrow \emptyset$ {Initialise best subset.}
9:     $C \leftarrow nextslyce(C, I, n)$ {Get the next coalition generated by SlyCE.}
10:   end while
11: end for
12: for all $s = 1 \ldots n$ do
13:   $m \leftarrow \lfloor s/2 \rfloor$
14:   if $s < n$ then
15:     $m \leftarrow \min(m, n - s)$
16:   end if{Maximum subset size}
17:   for $i = 1, \ldots \binom{n}{s}$ do
18:     $C \leftarrow L(I, i, s)$ {Go through sets of size $s$.}
19:     if $W(C) > -\infty$ then {Ignore $C$ if infeasible}
20:       for all $a \in C$ do
21:         $C' \leftarrow \{a\}$
22:         while $C' \neq \emptyset$ do
23:           if $W(C') + W(C \setminus C') > W(C)$ then
24:             $W(C) \leftarrow W(C') + W(C \setminus C')$
25:             $B(C) \leftarrow C'$
26:           end if{Evaluate subset}
27:           $C' \leftarrow nextslyce(C', C, m)$
28:         end while{Calculate $w(C)$}
29:       end for
30:     end if
31:   end for
32: end for
33: return bestcs($I$)
```

Algorithm 4 *bestcs*(C)

```
if $B(C) = \emptyset$ then
  return {$C$}
else
  return bestcs($B(C)$) $\cup$ bestcs($C \setminus B(C)$)
end if
```

when calculating $w(C)$, all possible splittings into feasible coalition subsets are considered, except $\{1,4\}$ and $\{2,3\}$. This means that $w(C)$ could possibly be less than $v(\{1,4\}) + v(\{2,3\})$. However, the only coalition structure that contains $\{1,4\}$ and $\{2,3\}$ is $\{\{1,4\},\{2,3\},\{5\}\}$ and $w(I)$ is still greater than or equal to $w(\{2,3\}) + w(\{1,4,5\})$ which is greater than or equal to $v(\{2,3\}) + v(\{1,4\}) + v(\{5\})$. So even though not every subset of every subset is considered, every coalition structure value is bounded by $w(I)$.

7. EMPIRICAL EVALUATION

In this section, we evaluate D-SlyCE and DyCE on a number of synergistic graph topologies. We benchmark D-SlyCE against DCVC, and DyCE against IDP and IBM's ILOG CPLEX (which solves the well known Integer Program formulation for the set partitioning problem posed by CFSS). Since our claim is that D-SlyCE and DyCE are particularly good in problems involving sparse graphs, we experiment with a variety of graphs of different densities. While, in the case of D-SlyCE, as argued in Section 5, we expect the distribution of enumeration tasks among agents to depend on the degree of the graph, in DyCE, we expect the degree of the graph to affect the time to evaluate all feasible coalition structures and the cost of checking the size of individual coalitions (as in *nextslyce*).

In our experiments,[3] we focus on those topologies most commonly found in synergistic networks such as social networks, the internet, and peer-to-peer communication between emergency responders, as follows: (i) Scale-free graphs — a network generated according to a power law. We use the standard Barabási-Albert [1] preferential attachment generation model, with parameters $k = 1, 2, 3$. Thus, as the graph is constructed, new agents are attached to k existing agents such that each new agent is attached to existing agent j with probability $\frac{d_i}{\sum_j d_j}$ where d_j is the degree of agent j for all $j \in I$. (ii) Random trees – an acyclic graph rooted at a vertex to simulate hierarchical organisations. These graphs are constructed by attaching each new agent to a single randomly picked existing agent and (iii) Complete graphs — an edge exists between each pair of agents, and so all coalitions are feasible. This is not a sparse graph and is unlikely to arise as a social context for large numbers of agents, but we include it as a worst case scenario. We next elaborate on the experiment results for D-SlyCE and DyCE in turn.

7.1 Benchmarking D-SlyCE

In running D-SlyCE and DCVC on the graphs described above, we recorded the individual runtimes of each agent and computed the ratio between the mean runtime and the maximum for different numbers of agents. This is used to analyse the fairness of the computation distribution among the agents. Using our setup, given this, we are able to run DCVC for up to 40 agents in 2.6 hours. For complete graphs

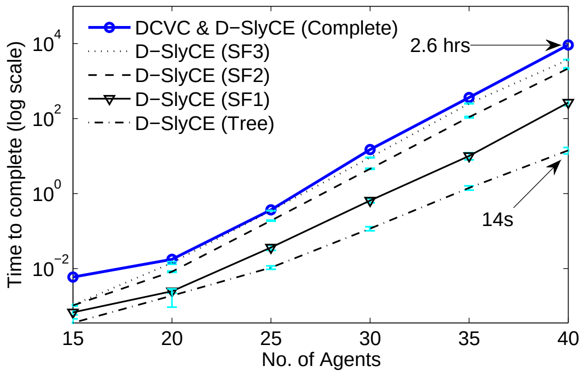

Figure 3: Runtimes for D-SlyCE (SFk = scale-free with parameter k).

[3]All our experiments are carried out on a 64 bit, quad-core PC with 12GB of RAM. In evaluating D-SlyCE, we repeated each experimental run 100 times (except for 40 agents where we ran only 20 times given the long runtimes) and for DyCE (on the more complex coalition structure generation problem), we repeated all the experiments 50 times (except for 28 agents onwards for trees/scale free parameter 1 and 23 agents onwards for the rest, which we repeated 20 times). In both sets of experiments, we recorded the mean, variance and 95% confidence intervals of the runtimes of the algorithms under study.

(see Figure 3 for runtime results), our algorithm matches the performance of DCVC, and, over sparse graphs (trees and scale free), outperforms DCVC for all numbers of agents. For trees and scale-free graphs, the worst case is 2.5 times faster (3700s compared to 9300s for a scale-free graph with $k = 3$ at 40 agents) and, in the best case, it is about 660 times faster (14s compared to 9300s for DCVC on a tree with 40 agents)! We note that in the case of random trees, D-SlyCE can evaluate all feasible coalitions for 40 agents within about 14 seconds and indeed D-SlyCE easily runs on trees of up 50 agents within 14 minutes! To date, no existing distributed coalition value calculation algorithm has been shown to have comparable performance. Also, note that as the density of the graph increases (i.e., as the number of connections per agent increases in k), the runtimes of the D-SlyCE algorithm increase as expected. This confirms that the density of the graph is a key determinant of the improvement that D-SlyCE makes over DCVC. Turning to the fairness of the computation distribution, as measured by the ratio of mean agent runtime against maximum runtime (such that 1 = equal computation distribution), the values obtained were (with SFk = scale-free with parameter k) — Trees: 0.09, SF1: 0.3, SF2: 0.6, SF3: 0.8, and Complete graph: 1 respectively. In the case of trees, a number of vertices that are well connected (e.g., high in the hierarchy) create disproportionately large task shares. However, in the increasingly dense scale-free graphs, all agents tend to get an increasingly fairer share of the computation as their relative degrees get closer (with the complete graph generating a perfect split of shares). When relating these results to the runtimes, however, we can see that in the case of random trees, the low overall computation time more than compensates for the unfair distribution.

7.2 Benchmarking DyCE

In this section, we describe two experiments (Exp1 and Exp2 respectively), one to evaluate how DyCE, IDP, and CPLEX compare in terms of runtime on the same instances and one where we evaluate the performance of DyCE as the graph density increases (see results in Figure 5). The latter is particularly important to consider since the number of feasible coalition structures (and calls to *nextslyce*) increases with graph density (see Algorithm 3). Thus, the gain from identifying feasible coalition structures is expected to tail off as the number of redundant coalition structures decreases (as graph degree increases).

Exp1: In terms of runtime, our results (see Figure 4) clearly show that DyCE outperforms IDP and CPLEX significantly on scale-free graphs and random trees. On complete graphs (by up to 7×10^4 times on trees for 30 agents compared to IDP), DyCE incurs an overhead that tails off (indicating a lower growth than that of the algorithm) and runs slower than IDP by a small amount. Due to long runtimes (and given that the trends are deterministic (i.e., growing in $O(3^n)$ for IDP) we extrapolated the results as follows: from 24 agents onwards for DyCE (Complete) and IDP, and from 29 onwards for DyCE (SF3) and 30 agents for DyCE (SF2). CPLEX, instead, is fast on small problems but surprisingly runs only up to 27 agents on trees (19 on SF3, 20 on SF2, 25 on SF1) — we do not plot these graphs for clarity). Moreover, we note that DyCE ran to completion within 5.3 minutes for 30 agents on trees.

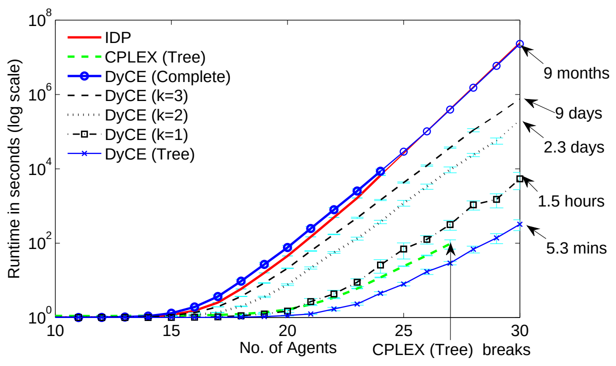

Figure 4: Runtimes for DyCE, IDP, and CPLEX.

Exp2: In this experiment, we evaluated DyCE with 20 agents on a scale-free graph with parameter $k \in \{1, 10\}$, where $k > 5$ results in a graph where each agent will be connected to more than half the number of agents (hence a dense graph). Given this, we note (from figure 5) that DyCE outperforms IDP for values of k up to 5, beyond which the graph is so dense that the coalition size checks in *nextslyce* become slightly more costly than the gain in avoiding infeasible coalitions which renders DyCE slightly slower on complete graphs (a fixed overhead which we believe could be improved through a faster implementation of *nextslyce*). Hence, it can be concluded that DyCE will work very well for most domains which have reasonably sparse synergies.

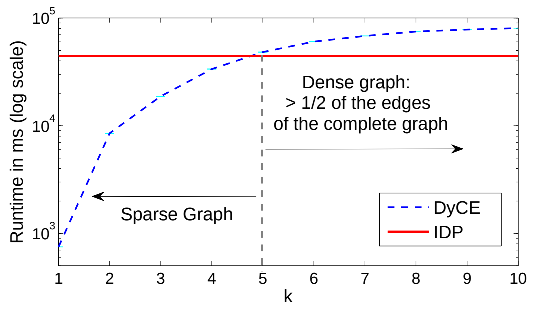

Figure 5: Runtime for $k \in \{1, 10\}$ in SFk ($n = 20$).

When taken together, our results allow us to say that D-SlyCE is the fastest distributed coalition enumeration algorithm over constraining graphs, and can help solve problems of more than 40 agents within reasonable time (in seconds, minutes, or hours depending on the graph). Moreover, we establish the benchmark for coalition structure generation algorithms in constraining graphs as DyCE can solve problems for up to 30 agents within minutes as compared to months for the state-of-the-art IDP.

8. CONCLUSIONS

In this paper we addressed the problem of coalition formation with sparse synergies where the set of feasible coalitions is constrained by the edges of a graph. Our aim was to see whether knowledge of the topology of an underlying social or organisational context graph could be used to speed up the task of coalition enumeration and structure generation.

We first developed the SlyCE algorithm and D-SlyCE to enumerate and evaluate all feasible coalitions over any graph in a distributed fashion such that agents end up with a fair share of computation. Theoretical results showed that (D-)SlyCE is correct, complete, and non-redundant. We then turned to the more challenging problem of coalition structure generation and proposed the DyCE algorithm to solve it using SlyCE as a building block.

Our empirical evaluation of D-SlyCE and DyCE, showed that they both outperformed the state-of-the-art algorithms by orders of magnitude (660 times for D-SlyCE and 7×10^4 times for DyCE), and for the first time, managed to enumerate coalition values for up to 50 agents in reasonable time, and find the optimal coalition structure for 30 agents within 5.3 minutes on random trees.

In general, our algorithms establish the benchmarks for many multi-agent applications where sparse synergies exist (e.g., decentralised coordination of sensors or emergency responders) where computing the optimal solution has, so far, not been possible due to the exponential time required to solve the coalition formation problems they generate. In future work, we aim to further improve DyCE by combining it with branch-and-bound techniques to further speed up the search and prune the synergistic graph to render the search feasible within reasonable time while providing quality guarantees on solutions returned. We also seek methods to automatically identify whether a problem has sparse synergies (i.e., based on graph degree) and thus help in the choice of the right algorithm to use.

9. REFERENCES

[1] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[2] G. Demange. On group stability in hierarchies and networks. *Journal of Political Economy*, 112(4):754–778, 2004.

[3] T. Michalak, J. Sroka, T. Rahwan, M. Wooldridge, P. Mcburney, and N. R. Jennings. A distributed algorithm for anytime coalition structure generation. In *Autonomous Agents And MultiAgent Systems (AAMAS 2010)*, pages 1007–1014, 2010.

[4] R. Myerson. Graphs and cooperation in games. *Mathematics of Operations Research*, pages 225–229, 1977.

[5] T. Rahwan and N. R. Jennings. An algorithm for distributing coalitional value calculations among cooperating agents. *AIJ*, 171(8-9):535–567, 2007.

[6] T. Rahwan and N. R. Jennings. Coalition structure generation: Dynamic programming meets anytime optimisation. In *Proc 23rd Conference on AI (AAAI)*, pages 156–161, 2008.

[7] T. Rahwan and N. R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *Proc 7th Int Conf on Autonomous Agents and Multi-Agent Systems*, pages 1417–1420, 2008.

[8] T. Rahwan, T. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N. R. Jennings. Constrained coalition formation. In *The 25th Conference on Artificial Intelligence (AAAI)*, 2011.

[9] T. W. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *AIJ*, 111(1–2):209–238, 1999.

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