Algorithms for the Shapley and Myerson Values in Graph-restricted Games

Oskar Skibski
University of Warsaw
oskar.skibski@mimuw.edu.pl

Tomasz P. Michalak
University of Oxford
University of Warsaw
tomasz.michalak@cs.ox.co.uk

Talal Rahwan
Masdar Institute
trahwan@gmail.com

Michael Wooldridge
University of Oxford
michael.wooldridge@cs.ox.co.uk

ABSTRACT

Graph-restricted games, first introduced by Myerson [20], model naturally-occurring scenarios where coordination between any two agents within a coalition is only possible if there is a communication channel(a path) between them. Two fundamental solution concepts that were proposed for such a game are the Shapley value and the Myerson value. While an algorithm has been proposed to compute the Shapley value in arbitrary graph-restricted games, no such general-purpose algorithm has yet been developed for the Myerson value.

Our aim in this paper is to develop a more efficient algorithm for computing the Shapley value, and to develop a general-purpose algorithm for computing the Myerson value, in graph-restricted games. Since the computation of either value involves visiting all connected induced subgraphs of the graph underlying the game, we start by developing an algorithm dedicated for this purpose, and show that it is faster that the fastest available one in the literature. This algorithm is then used as the cornerstone upon which we build two algorithms. The first is designed to compute the Shapley value, and is shown to be more efficient than the state of the art. The second is the first dedicated algorithm to compute the Myerson value in arbitrary graphs.

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*

Keywords

Myerson value, Shapley value, enumerating induced connected subgraphs, terrorist networks

1. INTRODUCTION

While the conventional model of a coalitional game assumes that any coalition can be created and may have an arbitrary value, there are many realistic settings where this assumption does not hold. Often, agents can communicate and cooperate only via some limited number of bilateral channels. If there is no direct channel between two agents, cooperation can be still possible indirectly, through an intermediary or a sequence of them. However, when no direct or indirect connection exists between agents, they cannot coordinate their activities. Such restrictions emerge in a variety of domains including: sensor networks, telecommunications, social networks analysis, trade agreements, political alliances, etc.

An influential approach for representing such scenarios was introduced by Myerson [20], who described a coalitional game over a graph in which nodes represent agents and edges represent communication channels between them. Such a game is often called a *graph-restricted game*. Some of the literature on graph-restricted games assumes that a coalition is feasible only if there exists a (direct or indirect) communication channel between any two of its members. Such a coalition is said to be connected, as it induces a connected subgraph of the underlying graph. Other work allows for the existence of both connected and disconnected coalitions. Here, disconnected coalitions are either all assumed to have a value of 0 (as in the model formalized by Amer and Gimenez [3]) or their values equal to the sum of values of the disjoint components they are composed of (as formalized by Myerson [20]).

In this paper, we study the computational aspects of the two key solution concepts proposed for the above graph-restricted games, namely the *Shapley value* and the *Myerson value*. Specifically:

- In his seminal work, Shapley proposed a formula to quantify the contribution of an individual agent to the outcome achieved by all agent working together in one coalition. While this formula, known as the Shapley value, satisfies many desirable properties (see Section 2), it is defined only for settings where all (both connected and disconnected coalitions) are feasible.
- On the other hand, Myerson proposed a solution concept for the settings in which only connected coalitions are feasible [20]. This solution concept, known as the Myerson value, also has a number of attractive properties (again see Section 2).

The computation of both of these values is challenging [18, 2]. As for the Shapley value, Michalak et al. recently proposed an algorithm to compute it for an arbitrary graph-restricted game [18]. As for the Myerson value, its computational aspects were considered for certain classes of graphs and/or games (see Section 7 for more details). However, to date there is no algorithm for computing the Myerson value in arbitrary graphs.

Our aim in this work is to develop efficient algorithms for computing the Shapley value and the Myerson value. In particular, our contributions can be summarised as follows:

- In Section 3, we propose a new algorithm for the enumeration of all connected induced subgraphs of the graphs—one

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.*

of the fundamental operations in graph theory. We show that our algorithm is faster than the state of the art, due to Moerkotte and Neumann [19]. We also show that, unlike the state of the art, our algorithm can easily be extended to capture extra information about each enumerated subgraph.

- Building upon the above enumeration algorithm, in Section 4, we propose a new algorithm to compute the Shapley value for graph-restricted games. We show that it is faster than the state of the art, due to Michalak et al. [18].
- In Section 5, we propose the first dedicated algorithm to compute the Myerson value for an arbitrary graph.
- Finally, in Section 6, we test both algorithms on an interesting application, recently proposed by [15], who used the Shapley value of graph-restricted games to measure importance of different members of a terrorist network. Our results suggest that the Myerson value-based measure is more suitable for this application.

2. PRELIMINARIES

A graph $G = (V, E)$ consists of vertices (or nodes) V and edges $E \subseteq V \times V$. Let $V' \subseteq V$ be a subset of vertices and let $E(V') \subseteq E$ denote the set of all edges between them in G. Now, the subgraph *induced* by V' is the pair $(V', E(V'))$. Since in this paper we do not consider non-induced subgraphs, we will often omit the word *induced* when there is no risk of confusion. A subgraph is *connected* if its edges form a path between any two of its nodes. We will denote the set of all connected induced subgraphs of G by $\mathcal{C}(G)$ (or simply $\mathcal{C}$ wherever G is clear from the context). We say that $v \in V$ is a *cut vertex* in connected graph (V, E) if its removal splits the graph, i.e., if $(V \setminus \{v\}, E(V \setminus \{v\}))$ is not connected. If the subgraph induced by V' is not connected, then it surely consists of several connected components, denoted $K(V') = \{K_1, K_2, \ldots, K_m\}$. Finally, for any vertex $v \in V$, we denote by $\mathcal{N}(v)$ the set of neighbours of v.

Having discussed some fundamental notions of graph theory, let us turn now to cooperative game theory. The players in our paper are represented by nodes in graph G. In other words, V is interpreted as the set of players (or agents). Consequently, we will use the terms coalition and subgraph interchangeably. A coalition S is said to be connected if and only if the subgraph of G induced by S is connected. Otherwise, the coalition is said to be disconnected.

Let ν denote the *characteristic function* that assigns to every coalition of agents, $S \subseteq V$, a real number representing its *payoff* (or *value*). Formally, $\nu : 2^V \rightarrow \mathbb{R}$. A cooperative game in characteristic function form is then given by a pair $\langle V, \nu \rangle$. Assuming that the coalition of all agents, V, forms, a solution of a coalitional game is a method of dividing $\nu(V)$—the payoff from cooperation—among the agents. One of the most important such solution concepts is the Shapley value [23], which is basically a weighted average marginal contribution of an agent to every coalition this agent could possibly belong to:

$$SV_i(\nu) = \sum_{S \subseteq V, v_i \in S} \xi_S \left(\nu(S) - \nu(S \setminus \{v_i\}) \right), \qquad (1)$$

where

$$\xi_S = \frac{(|S|-1)!(|V|-|S|)!}{|V|!}$$

An alternative equivalent formula for the Shapley value of agent v_i is:

$$\phi_i = \frac{1}{|V|!} \sum_{\pi \in \Pi} \left(\nu(S_i^\pi) - \nu(S_i^\pi \setminus \{v_i\}) \right), \qquad (2)$$

where Π is the set of all permutations of V, and S_i^π is the coalition consisting of v_i and the agents that precede v_i in permutation π. As such, according to the Shapley value, the payoff of an agent, $v_i \in V$, equals its average marginal contribution to the agents that precede it in an arbitrary permutation. This payoff is referred to as *the Shapley value of* v_i. The importance of the Shapley value stems from the fact that it is the unique solution concept that has the following four desirable properties: (1) it is efficient (i.e., the sum of agents' shares equals $\nu(V)$), (2) symmetric agents obtain symmetric payoffs, (3) agents who do not contribute to the game obtain no payoff, and (4) the division scheme is additive.

In graph-restricted games, which were first studied by Myerson [20], only connected coalitions could be assigned an arbitrary value (as the agents within are able to communicate and create value added). To formalise this class of games, let us first consider a new value function which corresponds to ν but is only defined over connected coalitions:

$$\nu_G : \mathcal{C}(G) \rightarrow \mathbb{R} \text{ and } \forall_{S \in \mathcal{C}(G)} \nu_G(S) = \nu(S)$$

This definition can be extended to incorporate disconnected coalitions; this has been done in two ways:

- Myerson argued that it is natural to consider a disconnected coalition as a set of disjoint, connected components. Each such component S' is, by definition, a coalition in $\mathcal{C}(G)$ whose members are able to attain a payoff of $\nu_G(S') = \nu(S')$. This leads to the following characteristic function, defined over both connected and disconnected coalitions [20]:

$$\nu_G^{\mathcal{M}}(S) = \begin{cases} \nu(S) & \text{if } S \in \mathcal{C}(G) \\ \sum_{K_i \in K(S)} \nu(K_i) & \text{otherwise,} \end{cases} \qquad (3)$$

  where $\mathcal{M}$ stands for Myerson. In other words, the payoff available to a disconnected coalition is the sum of payoffs of its connected components.

- More recently, Amer and Gimenez [3] formalised an alternative approach to evaluate disconnected coalitions, where they assumed that all such coalitions have a value of 0. Under this assumption, they defined the following characteristic function of a simple game:[1]

$$\nu_G^{\mathcal{A}}(S) = \begin{cases} 1 & \text{if } S \in \mathcal{C}(G) \\ 0 & \text{otherwise,} \end{cases} \qquad (4)$$

  where $\mathcal{A}$ stands for Amer and Gimenez. The game with the above function will be called a 0-1-*connectivity game*. This function was later on extended by Lindelauf et al. [15] to:

$$\nu_G^{f}(S) = \begin{cases} f(S, G) & \text{if } S \in \mathcal{C}(G) \\ 0 & \text{otherwise,} \end{cases} \qquad (5)$$

  where f is an arbitrary function.

Since $\nu_G^{\mathcal{M}}$ and $\nu_G^{\mathcal{A}}$ are defined over all $2^{|V|}$ coalitions, the Shapley value can be applied as a solution concept to both of these functions. However, this is not the case with ν_G. Now, a celebrated result of Myerson [20] is the solution concept he proposed for ν_G. In particular, Myerson showed that by allocating to agent $v_i \in V$ the payoff $MV_i(\nu_G)$, which is defined as follows:

$$MV_i(\nu_G) = SV_i(\nu_G^{\mathcal{M}}), \qquad (6)$$

we obtain the unique payoff division scheme that is efficient and rewards any two connected agents equally from the bilateral connection between them. This payoff division scheme is known as the *Myerson value*.

[1] Simple coalitional games are a popular class of games, where every coalition has a value of either 1 or 0.

3. DFS ENUMERATION OF INDUCED CONNECTED SUBGRAPHS

In this section, we present our algorithm for enumerating all induced connected subgraphs, and then benchmark its performance against the state of the art. As mentioned earlier in the introduction, this enumeration algorithm will be used in subsequent sections as the cornerstone upon which we build our algorithms for computing the Shapley value for connectivity games and the Myerson value for arbitrary graph-restricted games.

Broadly speaking, our enumeration algorithm traverses the graph in a depth-first manner, and uses a *divide-and-conquer* technique. We start with a single node and try to expand it to a bigger connected subgraph. Whenever a new node is analyzed, we explore all its edges one by one, and when we find a new—not yet discovered—node, we split the calculations into two parts: in the first one, we add a new node to our subgraph; in the second one, we mark this node as forbidden and never enter it again. Thus, the first part enumerates subgraphs with, and the second one without, the new node.

The pseudocode is presented in Algorithm 1. First, let us describe the recursive function $DFSEnumerateRec$. Whenever this function is called, nodes in the graph can be divided into three groups: S—elements of the subgraph; X—forbidden nodes (that we cannot include in the subgraph); and others—not yet discovered nodes. Moreover, nodes in S are either *partially-processed* or *fully-processed*. A node is fully-processed when all its edges have been explored. What is crucial, as in the classic DFS algorithm, is that *all partially-processed nodes form a path to the root of the subgraph tree* (parameter $path$), i.e., we do not process another edge until the node from the previous one is fully-processed. The last parameter of $DFSEnumerateRec$ is $startIt$, which—to avoid redundancy—indicates how many edges of the last node on $path$ have already been processed. This parameter is set to 1 as we enter a new node (lines 4 and 10) and can be deduced from the neighbour list whenever we backtrack from another node (line 14).[2]

Now, the goal of the function is to find all connected subgraphs that contain subgraph S and contain no forbidden nodes X. To this end, we start processing from the last node on the path (line 6), denoted v, and explore sequentially all its edges (lines 7-11). Whenever we find a new node u (that is neither forbidden nor included in S) we first enumerate all subgraphs with u: here, we call $DFSEnumerateRec$ with u added at the end of the path of partially-processed nodes (line 10). Then, to enumerate all subgraphs without u, we add it to the set of forbidden nodes (line 11) and proceed with a new edge. Finally, when all edges have already been explored, we remove v from the path and backtrack to the previous node (lines 12-15). When we have finished processing the last node on the path (root), the set S constitutes a final connected subgraph (line 16).

In the main function $DFSEnumerate$ (lines 1-4), the i-th step of the loop enumerates all subgraphs in which the node with the smallest index is v_i (line 4). To this end, we simply mark previous nodes as forbidden and call the function $DFSEnumerateRec$ with the node v_i as the initial subgraph.

The time complexity of our algorithm is linear in the number of connected subgraphs: $\mathcal{O}(|\mathcal{C}||E|)$. This follows from the fact that the number of steps performed for a given connected subgraph is $\mathcal{O}(|E|)$. To see how this is the case, consider a sequence of calls of $DFSEnumerateRec$ that results in printing subgraph S in line 16. We consider a subgraph to be final if $path$ is empty; thus, all nodes from S must be fully-processed. Moreover, all other nodes are either forbidden or not-discovered; thus, they are not added to $path$ in this sequence (the recursive call in which we consider adding a forbidden node to the subgraph is calculated in our analysis for other connected subgraph). Therefore, the lines 9-11 from the single loop are entered once for every edge adjacent to a node in S, thus no more than $2|E|$ times. Moreover, every call of the function decreases the number of edges to discover, or decreases the number of nodes on the path; thus, other lines are called no more than $2|E| + |S|$ times. As $|S|$ is connected, $|S| \leq |E| + 1$ and the number of steps is $\mathcal{O}(|E|)$.

The running time of the algorithm depends on the order in which we process nodes (line 3) and nodes' neighbours (line 10). The optimal order of nodes is an open problem. In our experimental analysis we found that the order descending by the degree of the node can lead to a smaller number of steps. Therefore in line 2 we sort the nodes accordingly.

Algorithm 1: DFS Enumeration of Induced Connected Subgraphs

Input: Graph $G{=}(V, E)$
Output: List of all induced connected subgraphs of G

```
1  DFSEnumerate begin
2      sort nodes and list of neighbours by degree desc.;
3      for i ← 1 to |V| do
4          DFSEnumerateRec(G, (v_i), {v_i}, {v_1, ..., v_{i-1}}, 1);
5  DFSEnumerateRec(G, path, S, X, startIt) begin
6      v ← path.last();
7      for it ← startIt to |N(v)| do
8          u ← N(v).get(it); // it's neighbour of v
9          if u ∉ S ∧ u ∉ X then
10             DFSEnumerateRec(G, (path, u),
                   {u}, X, 1);
11             X ← X ∪ {u};
12     path.removeLast();
13     if path.length() > 0 then
14         startIt ← N(path.last()).find(v) + 1;
15         DFSEnumerateRec(G, path, S, X, startIt);
16     else print S;
```

3.1 DFS vs. BFS enumeration of induced connected subgraphs

To date, the state-of-the-art algorithm for enumerating connected induced subgraphs was proposed by Moerkotte and Neumann [19]. As opposed to our algorithm, which traverses the graph in a depth-first manner, their algorithm uses breadth-first search. The pseudocode is presented in Algorithm 2. Specifically, in the i-th step of the main function, $EnumerateCsg$, the algorithm enumerates subgraphs with v_i and without previous nodes. The recursive function $EnumerateCsgRec$ is called with four parameters: graph G, an Old part of the subgraph, a New part of the subgraph, and the set of all nodes that we already considered, denoted X (the nodes from the subgraph and nodes we have considered but have not included). Now, $EnumerateCsgRec$ outputs the current subgraph ($Old \cup New$) and tries to enlarge it. In order to do that, it lists all not-yet considered neighbours (set N) and for every subset $S \subseteq N$ analyzes an adequate extension—it calls $EnumerateCsgRec$ with the subgraph enlarged by S and set of considered nodes expanded by all neighbours N.[3]

[2] As function $find(v)$ in line 14 is called multiple times, the proper values can be pre-calculated in the main function $DFSEnumerate$ and stored in the associative array to facilitate constant access time.

Algorithm 2: BFS Enumeration of Induced Connected Subgraphs

Input: Graph $G{=}(V, E)$
Output: List of all Induced Connected Subgraphs of G

```
1  EnumerateCSG begin
2      for i ← 1 to |V| do
3          EnumerateCSGRec(G, ∅, {v_i}, {v_1, ..., v_i});

4  EnumerateCSGRec(G, Old, New, X) begin
5      print{Old ∪ New};
6      N ← ∅; // not yet discovered neighbours of New
7      foreach v ∈ New do
8          foreach u ∈ N(v) do
9              if u ∉ X ∪ N then N ← N ∪ {u};
10     foreach S ⊆ N do
11         EnumerateCSGRec(G, Old ∪ New, S, X ∪ N);
```

Our experiments show that the new algorithm outperforms BFS enumeration two or even three times. In particular, Figure 1 depicts the running time for scale-free graphs, typically used to model contact networks. Graphs were generated using the preferential attachment generation model [1] with parameter $k = 4$ (we obtained analogous results for different values of k). For every $n = 20, \ldots, 30$, the run time and confidence intervals are calculated based on 500 random graphs (same for both algorithms). As can be seen, as n increases, the ratio of both algorithms does not change and oscillates at around 2.4. For instance, for $n = 30$, our algorithm takes on average 67 seconds, while it takes 161 seconds for BFS enumeration to finish.

To support our empirical results, we provide two lemmas, which show that for cliques our algorithm performs approximately two times fewer steps (examining edges is the key component of main loops in both algorithms).

LEMMA 1. *$EnumerateCSG$ examines edges $2^{n-1}(n^2-3n+2)+(n-1)$ times for an n-clique.*

LEMMA 2. *$DFSEnumeration$ examines edges $2^{n-2}(n^2-n+4)-(n+1)$ times for an n-clique.*

4. SHAPLEY VALUE FOR THE CONNECTIVITY GAMES

In this section, we present a new algorithm for calculating the Shapley value of connectivity games proposed by Amer and Gimenez [3]. As mentioned earlier in the introduction, there already exists an algorithm designed for this purpose, due to [18], and we aim to develop a more efficient algorithm.

As noted by Michalak et al. [18], to calculate the Shapley value it suffices to traverse only the connected coalitions, because every non-zero marginal contribution involves the addition, or removal, of a player from a connected coalition. In more detail, let S be an arbitrary connected coalition. Now, players' contributions can be divided into three groups:

- a cut vertex (i.e., a node whose removal disconnects the subgraph S) contributes the entire value of the coalition, i.e., $\nu^f_G(S)$;
- any other member of S (whose removal does not disconnect S) contribute the following change in the value: $\nu^f_G(S) - \nu^f_G(S \setminus \{v\})$ for node v;
- finally, we have the nodes that are not members nor neighbours of S. The addition of any such node disconnects S, implying that it makes a negative contribution equal to $-\nu^f_G(S)$.

Note that we did not consider the contribution of neighbours of S. This is because, for every such neighbour, v, its contribution will be taken into account when dealing with $S \cup \{v\}$ instead of S.

Based on the above observations it is crucial to not only enumerate all connected subgraphs, but also identify the cut vertices, and the neighbours, of each enumerated subgraph. As for the identification of neighbours, it can easily be done. The harder part is the to identify the cut vertices. To this end, in [18], the authors combined Moerkotte and Neumann's enumeration algorithm with the state-of-the-art algorithm for finding cut vertices, due to Hopcroft and Tarjan [13]. The way in Michalak et al. combined the two algorithms involved some additional improvements, see their paper for more details.

Against this background, we present the first dedicated algorithm that not only enumerates all connected subgraphs, but at the same time identifies cut vertices in each subgraph. To make this possible, our algorithm traverses all connected subgraphs in a depth-first-search (DFS) manner (as discussed in the previous section). Consequently, unlike the case with Moerkotte and Neumann's breadth-first-search (BFS) technique, our DFS techniques ensures that the edges which are used to enlarge the subgraph always form what is known as a Tremaux tree [13]—an important structure in graph theory. More specifically, a Tremaux tree of graph G is a rooted spanning tree—a subgraph consisting of all nodes and a subset of edges, which forms a tree, with one node selected as a the root. Importantly, for any Tremaux tree, T, and any two nodes that have an edge between them in G, it is guaranteed that one of those two nodes is an ancestor of the other in T. Now, let us show how this property of Tremaux trees helps identify cut vertices in a subgraph. To this end, let v be an arbitrary node in G. Consider a subtree S rooted at a child of v. The removal of v from graph G disconnects nodes from S if and only if there is no edge in G that connects S

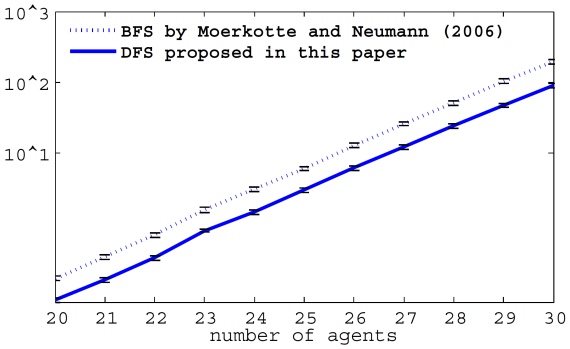

Figure 1: Comparison between algorithms for enumerating induced connected subgraphs: our new DFS-based algorithm and the state-of-the-art, BFS-based algorithm by Moerkotte and Neumann.

[3]Our pseudocode is more detailed than the original. If we merge both parts of the subgraph (Old and New) in the declaration (and calls) of $EnumerateCSGRec$, then to find neighbours in lines $7-11$ we have to consider also nodes from the old part of the subgraph. This is clearly redundant, as all their neighbours are already in the set X.

Algorithm 3: DFS-based algorithm for calculating Shapley value for the connectivity games

Input: Graph $G{=}(V, E)$, function $\nu : \mathcal{C} \to \mathbb{R}$
Output: Shapley value for game $\nu_G^{\mathcal{A}}$

```
1  DFSConnSV(G) begin
2      sort nodes and list of neighbours by degree desc.;
3      for i ← 1 to |V| do SV_i(ν_G^A) = 0;
4      for i ← 1 to |V| do
5          DFSConnSVRec(G, (v_i), (∞), {v_i}, {v_1, ..., v_{i-1}}, ∅, ∅, 1
6  DFSConnSVRec(G, path, low, S, X, SC, XN, startIt)
   begin
7      v ← path.last(); l ← low.last();
8      for it ← startIt to |N(v)| do
9          u ← N(v).get(it); // it's neighbour of v
10         if u ∉ S ∧ u ∉ X then
11             DFSConnSVRec(G, (path, u), (low, ∞),
                   S ∪ {u}, X, SC, XN, 1);
12             X ← X ∪ {u}; XN ← XN ∪ {u};
13         else if u ∈ X then XN ← XN ∪ {u};
14         else if (path.find(u) < low.last()) then
15             l ← path.find(u);
16             low.updateLast(l);
17     path.removeLast(); low.removeLast();
18     if path.length() > 0 then
19         if l ≥ path.length() then SC.add(path.last());
20         else if l < low.last() then low.updateLast(l);
21         startIt ← N(path.last()).find(v) + 1;
22         DFSConnSVRec(G, path, low, S, X, SC, XN, startIt);
23     else
24         if v was added only once SC then
25             SC.remove(v);
26         foreach v_i ∈ SC do
27             SV_i(ν_G^A) ← SV_i(ν_G^A) + ξ_S(ν(S));
28         foreach v_i ∈ S \ SC do
29             SV_i(ν_G^A) ← SV_i(ν_G^A) + ξ_S(ν(S) − ν(S \ {v_i}));
30         foreach v_i ∈ V \ (S ∪ XN) do
31             SV_i(ν_G^A) ← SV_i(ν_G^A) − ξ_{S∪{v_i}}(ν(S));
```

to other parts of the graph. From the property of Tremaux tree, all such potential edges would go to the ancestors of v (we will call them *backedges*). Thus, to identify cut vertices, it suffices to know the node nearest to the root that can be reached from the children's subtrees. This information can be easily updated recursively when we backtrack in a depth-first search—for the subtree rooted at v, it is one of the nodes connected to v or one of the nearest nodes that can be reached from its subtrees.

The pseudocode is presented in Algorithm 3. To gather extra information, we expand the recursive function from Algorithm 1 by a few new parameters. Assume that a root is on level 1, and its children are on level 2, and so on. Now, for each node v from the $path$, list low stores the lowest level that can be reached from v (using already discovered edges) or its fully-processed children. The set SC contains identified cut vertices. Now, whenever we add a node v to a path (lines 5 and 11) we initialize its low to infinity. Then, we update this value in two situations. The first is when we find a backedge from v to the lower level (lines 14-16). The second is when we backtrack from a child with a lower value (line 20, low for child equals l, parent is the last node on the $path$). When we backtrack and child's value is not lower that the level of the parent (thus, subtree of a son does not have a backedge to any node closer to the root) we add the parent node to the set of cut vertices SC (line 19). Finally, in lines 24-25, we remove root from the set of cut vertices if it has only one child in a tree, i.e., was added to this set just one time. The set of neighbours XN of the nodes in S can be easily updated as we consider all edges of nodes in the main loop (lines 8-16). Now, based on both sets, we calculate the Shapley value in lines 26-31 for every found connected coalition. The asymptotic time of the algorithm has not changed with respect to enumeration of connected subgraphs and equals $\mathcal{O}(|\mathcal{C}||E|)$.[4]

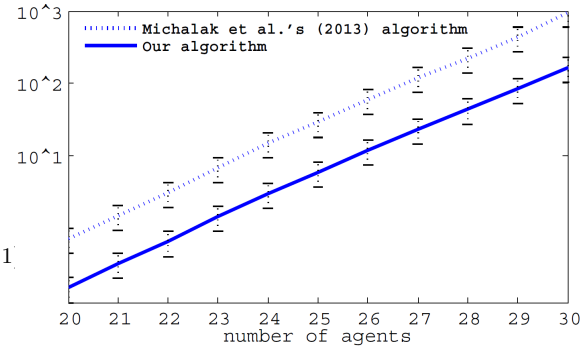

Figure 2: Comparison between algorithms for the Shapley value for connectivity games: our new DFS-based algorithm vs. the state-of-the-art BFS-based algorithm proposed by Michalak et al.

In Figure 2 we compare the performance of our new algorithm with the FasterSVCG proposed by Michalak et al. [18]. As in the comparison of DFS and BFS enumeration in Section 3.1, here we generated 500 random scale-free graphs for every size of graph n from 20 to 30 and computed average running time for both algorithms. As we can see, our DFS-based algorithm is more than 5 times faster for every size n. Importantly, only part of the advantage comes from the faster enumeration algorithm, which is 2-3 times faster than the alternative used in FasterSVCG. Moreover, the advantage increases very slightly with the number of nodes. We justify this as follows: although the pessimistic number of steps for every found connected subgraph in both enumeration algorithms equals $\mathcal{O}(E)$, for sparse graphs it does not exceed a small constant. Thus, separate searching for the cut vertices, which is a part of the FasterSVCG, is an additional action which takes linear time in respect to the number of edges—$\mathcal{O}(E)$—while our new algorithm performs only $\mathcal{O}(V)$ steps updating Shapley value for all nodes.

5. THE MYERSON VALUE OF ARBITRARY GRAPH-RESTRICTED GAMES

In this section we present the first dedicated algorithm for computing the Myerson value for arbitrary graph-restricted games. Depending on the definition of function ν, the complexity of calculating the Myerson value may vary. In our paper we do not make any assumption for the form of function ν: in our algorithm we treat ν as an oracle which gives the values, and for the complexity analysis we assume it does that in a constant time.

As in the previous section, we can argue that only traversing

[4]Note, that to obtain a constant time of $path.find(u)$ operation in line 14 list $path$ should be enriched with a associative array or alternatively additional array of nodes' levels can be passed along.

connected coalitions is necessary. Moreover, based on the oracle-assumption, we have to go through all of them (the size of the input is the number of values of ν). Thus, what is crucial is to minimize the number of steps performed for every connected coalition. In this section we prove that in the context of the Myerson value, identifying of cut vertices is not needed, and the number of steps is linear in the size of graph nodes.

To this end, we will use the permutation interpretation of the Shapley value (see formula (2)) and analyze the marginal contribution of a node, but this time more thoroughly. Let π be a permutation and assume players that precede v_i form the components $K_1, K_2, \ldots, K_m$, the first j of which are connected to v_i. Now, as in Myerson's characteristic function value of a coalition equals sum of values of it's components, we can simplify marginal contribution to $\nu(K_1 \cup \ldots \cup K_j \cup \{v_i\}) - \nu(K_1) - \ldots - \nu(K_j)$: all components not connected with v_i contributes their values to the value of a coalition with and without v_i. Now, instead of considering marginal contribution as a whole, as in the previous section, we will decompose it into two parts: a positive and a negative one. Thus, we will calculate separately how many times (i.e., for which of the permutations) the value of a coalition $\nu(K_1 \cup \ldots \cup K_j \cup \{v_i\})$ with v_i is added, and how many times a value of a given coalition K_l without v_i is subtracted from his payoff.

Consider a connected coalition S and the Myerson value of v_i:

- if v_i is in S then the value of S is taken into account with a positive sign whenever two conditions are met: (1) all nodes from S appear in the permutation before v_i, and (2) all neighbours of S appear in the permutation after v_i. This happens with a probability of $\frac{(|S|-1)!\cdot|\mathcal{N}(S)|!}{(|S|+|\mathcal{N}(S)|)!}$, where $\mathcal{N}(S)$ is the set of neighbours of S.
- if v_i is not in S, then the value of S is taken into account with a negative sign, but only if all nodes from S appear in the permutation before v, and all neighbours of S appear after v_i, and v_i is a neighbour of S (otherwise, v_i contributes to some other coalition). This happens with a probability of $\frac{|S|!\cdot(|\mathcal{N}(S)|-1)!}{(|S|+|\mathcal{N}(S)|)!}$.

It is worth noting that in the second case we allow neighbours of v that are not connected to S to appear before v_i. Although they change the coalition resulting from the appearance of v_i and also the subtracted part of the marginal contribution, the value of S is *still* a component of this part.

Based on the above observations, we construct our algorithm for computing the Myerson value based on our fast DFS-based algorithm for enumerating connected subgraphs (Algorithm 4). As in our previous algorithm for computing the Shapley value, here the set XN gathers neighbours of the set S, and lines 19-22 update the Myerson value according to the aforementioned marginal contribution analysis.

Algorithm 4: DFS-based algorithm for calculating Myerson value for graph-restricted games

Input: Graph $G{=}(V, E)$, function $\nu : \mathcal{C} \to \mathbb{R}$
Output: Myerson value for game ν_G

```
1  DFSMyersonV begin
2      sort nodes and list of neighbours by degree desc.;
3      for i ← 1 to |V| do MV_i(ν_f) = 0;
4      for i ← 1 to |V| do
5          DFSMyersonVRec(G, (v_i), {v_i}, {v_1, ..., v_{i-1}}, ∅, 1);
6  DFSMyersonVRec(G, path, S, X, XN, startIt) begin
7      v ← path.last();
8      for it ← startIt to |N(v)| do
9          u ← N(v).get(it); // it's neighbour of v
10         if u ∉ S ∧ u ∉ X then
11             DFSMyersonVRec(G, (path, u), S ∪
               {u}, X, XN, 1);
12             X ← X ∪ {u}; XN ← XN ∪ {u};
13         else if u ∈ X then XN ← XN ∪ {u};
14     path.removeLast();
15     if path.length() > 0 then
16         startIt ← N(path.last()).find(v) + 1);
17         DFSMyersonVRec(G, path, S, X, XN, startIt);
18     else
19         foreach v_i ∈ S do
20             MV_i(ν_G) ← MV_i(ν_G) + (|S|-1)!(|XN|)! / (|S|+|XN|)! ν(S);
21         foreach v_i ∈ XN do
22             MV_i(ν_G) ← MV_i(ν_G) - (|S|)!(|XN|-1)! / (|S|+|XN|)! ν(S);
```

6. APPLICATION

There is currently much interest in the possibility of applying social network analysis techniques to investigate terrorist organizations [22]. A particular attention is paid to the problem of identifying key terrorists. This not only helps to understand the hierarchy within these organizations but also allows for a more efficient deployment of scarce investigation resources [14]. One possible approach to this problem is to try and infer the importance of different individuals from the topology of the terrorist network. In graph theory, such an inference can be obtained in various ways, depending on the adopted *centrality measures*, i.e., the adopted way to measure the centrality, i.e., importance, of different nodes in a network, based on its topology.[5] A number of researchers have proposed to incorporate game-theoretic techniques into existing centrality measures [12, 9]. The basic idea behind such *game-theoretic centrality measures* is to define a coalitional game over the network and then to construct a ranking of nodes based on a chosen solution concept. Although such an approach is often computationally challenging, it has the following two advantages. Firstly, the combinatorial analysis of the cooperative game, which is embedded in the solution concept, becomes a combinatorial analysis of the network. Secondly, this approach is very flexible, as it can be changed along three dimensions: (i) the coalitional game can be of an arbitrary form (e.g., partition function form); (ii) the value function can also be arbitrary; and (iii) there are many available solution concepts, each based on different prescriptive and normative considerations.[6]

Lindelauf et al.'s measure: Lindelauf et al. tried to develop a centrality measure that assesses the role played by individual terrorists in a way that accounts for the following two factors: the terrorists' role in connecting the network and additional intelligence available about the terrorists. To this end, Lindelauf et al. proposed to use the Shapley value for v_G^f as defined in formula (5). This function seems to be suitable to achieve Lindelauf et al.'s goal. As discussed at the beginning of Section 4, it assigns high marginal contributions to cut vertices. Such vertices, by definition, play an important role in connecting various parts of the network. As such, any agent who is a cut vertex will be called a pivotal agent.

[5]We refer the reader to [5, 7] for an overview of most commonly used centrality metrics.

[6]We refer the reader to the recent book by Maschler et al. [16] for an excellent overview of solutions concepts in cooperative game theory.

Furthermore, one can manipulate $f(S,G)$ in formula (5) to incorporate available information and analytical needs. In particular, to incorporate additional intelligence on individual terrorist, the authors assigned weights to both edges and nodes in G. Let us denote such weights by ω_{ij} and ω_i, respectively. Based on this, Lindelauf et al. proposed to use the following alternative functions:

(a) $f(S) = |E(S)| / \sum_{(v_i,v_j)\in E(S)} \omega_{ij}$,
(b) $f(S) = \sum_{v_i \in S} \omega_i$.

In words, in (a) $f(S)$ equals the number of edges in the connected coalition divided by the sum of their weights and, in (b) by the sum of its nodes' weights. As an example, the rationale behind function (b) is that terrorists (nodes) with high weights "*play an important part in the operation. When such individuals team up, they have a significant effect on the potential success of the operation.*" [15, p. 237].

Summarising, the key idea of Lindelauf et al. was to develop a measure that evaluates the nodes based on two factors: their role in connecting the network; and additional intelligence. At first glance, v_G^f seems to be a good candidate for this purpose. However, in what follows we will show the disadvantages of this approach and propose an alternative.

The disadvantages of Lindelauf et al.'s measure: We argue that, for sparse networks—and terrorist networks tend to be sparse [14]—the "connectivity" factor is over-represented in the measure based on $SV_i(v_G^f)$. As a result, we claim that the "additional intelligence" factor hardly ever affects the ranking. To see how this is the case, it is sufficient to examine the characteristic function ν_G^f which (for reasonable f) results in relatively very high marginal contributions assigned to pivotal agents, while other agents are assigned incremental values or relatively very big negative values.

To support our claim, let us perform a sensitivity analysis of Lindelauf et al.'s centrality by (i) evaluating different forms of the function $f(S,G)$ and (ii) considering different weights of nodes. Regarding (i), we consider the following general form of $f(S,G)$:

$$f(S,G)=|S|^{\alpha}\cdot|E(S)|^{\beta}\cdot\Big(\sum_{v_i\in S}\omega_i/|S|\Big)^{\gamma}\cdot\Big(\sum_{(v_i,v_j)\in E(S)}\omega_{ij}/|E(S)|\Big)^{\delta}, \tag{7}$$

where $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ are parameters for exponents. We set α and β to be integer values between -2 and 2 and we impose the additional condition that $\alpha + \beta \geq 0$ in order to preserve monotonicity. We performed our tests on the terrorist network responsible for the World Trade Center attacks (36 nodes, 64 edges) [14]. This is a bigger version of the network originally considered by Lindelauf et al., which only contained 19 nodes.

Table 1 presents the results of this sensitivity check. For each configuration of parameters we calculated the ranking of nodes and compared it with the ranking for the 0-1-connectivity game. Here, we concentrate on the top $\sqrt{|V|} = 6$ terrorists, as the main goal of this application is to identify key players. The top part of Table 1 presents the average size of the intersection of the top 6 terrorists in both rankings. In the lower part of Table 1, for the top 6 terrorists from the 0-1-connectivity game, we calculated the average distance between their positions in both rankings, i.e. the ranking from the 0-1-connectivity game and the ranking from $f(S,G)$ (each cell presents the maximum and minimum value of this average).

We observe that the top 6 members have changed in 15% of the tests. Furthermore, in only 2% of the tests, more than one new node has been identified as a key member. Also, within this group, changes of positions are minor; the average change of position rarely exceeded 1.3. The dedicated characteristic function proposed

$\alpha\backslash\beta$		2	1	0	-1	-2
	$\gamma\backslash\delta$	1|0|-1	1|0|-1	1|0|-1	1|0|-1	1|0|-1
2	1|0|-1	6.0	6.0	5.9	5.9	4.9
1	1|0|-1	6.0	6.0	6.0	5.1	—
0	1|0|-1	6.0	6.0	5.7	—	—
-1	1|0|-1	6.0	6.0	—	—	—
-2	1|0|-1	6.0	—	—	—	—

$\alpha\backslash\beta$		2	1	0	-1	-2
	$\gamma\backslash\delta$	1|0|-1	1|0|-1	1|0|-1	1|0|-1	1|0|-1
2	1|0|-1	1.3-1.3	1.3-1.3	1.3-1.3	0.7-1.3	1.3-2.5
1	1|0|-1	1.3-1.3	1.3-1.3	1.0-1.3	0.8-2.0	—
0	1|0|-1	1.3-1.3	1.0-1.3	0.0-1.0	—	—
-1	1|0|-1	1.3-1.3	0.3-1.3	—	—	—
-2	1|0|-1	0.7-1.3	—	—	—	—

Table 1: Comparison between top nodes based on games with the parametrized characteristic function from formula (7) and the 0-1-connectivity game.

$\alpha\backslash\beta$		2	1	0	-1	-2
	$\gamma\backslash\delta$	1|0|-1	1|0|-1	1|0|-1	1|0|-1	1|0|-1
2	1|0|-1	4.8	4.7	4.3	1.1	0.0
1	1|0|-1	5.0	4.9	1.1	0.0	—
0	1|0|-1	4.4	4.2	0.0	—	—
-1	1|0|-1	3.3	0.8	—	—	—
-2	1|0|-1	1.1	—	—	—	—

$\alpha\backslash\beta$		2	1	0	-1	-2
	$\gamma\backslash\delta$	1|0|-1	1|0|-1	1|0|-1	1|0|-1	1|0|-1
2	1|0|-1	1.2-2.3	1.0-2.2	0.8-4.2	6.7-24.5	23.8-28.2
1	1|0|-1	1.2-2.8	1.0-2.8	8.8-28.0	21.3-28.3	—
0	1|0|-1	1.7-3.7	2.5-8.2	23.7-30.0	—	—
-1	1|0|-1	2.0-7.0	13.0-26.2	—	—	—
-2	1|0|-1	11.2-21.8	—	—	—	—

Table 2: Comparison between top nodes based on games with the parametrized characteristic function from formula (7) and the game with value of -1 for every connected component.

by Lindelauf *et al.* for the WTC network obtained with parameters $\alpha = \gamma = 1$ and $\beta = \delta = 0$ yields the same group of 6 terrorists ranked with only minor rotations.

The above simulations show that, for sparse networks, the choice of $f(S,G)$ essentially does not matter. The main reason behind this phenomenon appears to be the fact that ν_G^f assigns relatively very high contributions to pivotal agents, and only incremental marginal contributions to non-pivotal agents. This is magnified by the fact that we deal here with a sparse network. Specifically in this network, out of all 6 billions induced subgraphs, only 0.6% are connected. Furthermore, the average number of pivotal agents in a connected subgraph was high (8 to be precise). Thus, nodes that are crucial from the connectivity point of view will have a high ranking because v_G^f favours pivotal agents.

As already mentioned, we also analysed the sensitivity of the characteristic function ν_G^f with respect to the weights ω_i (these weights were permuted at random in our experiments). We focused on the second function proposed by Lindelauf et al., i.e., $f(S) = \sum_{v_i \in S} \omega_i$. Here, only 3% out of 600 permutations resulted in a change within the top 6 terrorists in the ranking. Furthermore, all these changes concerned only one terrorist (who was replaced by another).

In the next subsection, we argue that the Myerson value is a better centrality measure for terrorist networks.

The Myerson value for terrorist networks: We observe that the characteristic function $\nu_G^{\mathcal{M}}$ used to compute $MV_i(\nu_G)$ does not favour pivotal agents as much as ν_G^f. In particular, given an arbitrary (non-negative and monotonic) function, $f(S, G)$, the marginal contribution of a pivotal agent v to a coalition $S \cup \{v\}$ in the case of the connectivity game will be:

$$f(S \cup \{v\}, G) - 0 = f(S \cup \{v\}, G).$$

However, in the case of $\nu_G^{\mathcal{M}}$, it will be:

$$f(S \cup \{v\}, G) - \sum_{K_i \in K(S)} \nu(K_i) \leq f(S \cup \{v\}, G).$$

This means that the connectivity factor becomes relatively less dominant in the evaluation of the nodes.

In Table 2, we present results of a similar sensitivity check as before, but now for the Myerson value. As visible, this measure is more sensitive to changes of $f(S, G)$ than the Shapley value. This is also confirmed by the sensitivity check with respect to random permutations of nodes' weights. Specifically, in more than 80% of the cases, the top 6 terrorists changed, and, in most of those cases, the change was substantial. In particular, on average about 2.2 terrorists were repeated among the top 6.

7. RELATED WORK

The enumeration of connected induced subgraphs is one of the fundamental algorithmic operations in many applications, e.g., cost-based query optimization [19], computing topological indices for molecular graphs [21], and searching for an optimal coalition structure in cooperative games on graphs [25]. A number of algorithms have been proposed to perform this operation. The early works include reverse search algorithms [4], and a breadth-first search algorithm [24]. Both of these algorithms, however, performed numerous redundant operations. This issue was later on resolved by Moerkotte and Neumann.[7] The problem of identifying cut vertices has been studied in various domains, including network reliability [11], data clustering [8], among others. The computational properties of the Myerson value for some special classes of games were considered in [2, 10, 6]. Finally, we point the reader to works from the computer science literature, where graph-restricted games were considered [17, 25].

Acknowledgements: Tomasz Michalak and Michael Wooldridge were supported by the European Research Council under Advanced Grant 291528 ("RACE").

8. REFERENCES

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

[2] E. Algaba, J. Bilbao, J. Fernández, N. Jiménez, and J. López. Algorithms for computing the myerson value by dividends. *Discrete Mathematics Research Progress*, pages 1–13, 2007.

[3] R. Amer and J. M. Giménez. A connectivity game for graphs. *Math. Methods of OR*, 60:453–470, 2004.

[4] D. Avis and K. Fukuda. Reverse search enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

[5] U. Brandes and E. Thomas. *Network Analysis: Methodological Foundations*. LNCS, 2005.

[6] J. Fernández, E. Algaba, J. M. Bilbao, A. Jiménez, N. Jiménez, and J. López. Generating functions for computing the myerson value. *Annals of Operations Research*, 109(1-4):143–158, 2002.

[7] N. Friedkin. Theoretical foundations for centrality measures. *Am. J. of Sociology*, 96(6):1478–1504, 1991.

[8] G. Gan, C. Ma, and J. Wu. *Data clustering: theory, algorithms, and applications*, volume 20. Siam, 2007.

[9] D. Gómez, E. González, C. Manuel, G. Owen, M. Del Pozo, and J. Tejada. Centrality and power in social networks: A game theoretic approach. *Math. Soc. Sci.*, 46:27–54, 2003.

[10] D. Gómez, E. González-Arangüena, C. Manuel, G. Owen, M. del Pozo, and J. Tejada. Splitting graphs when calculating myerson value for pure overhead games. *Mathematical Methods of Operations Research*, 59(3):479–489, 2004.

[11] D. Goyal and J. Caffery. Partitioning avoidance in mobile ad hoc networks using network survivability concepts. In *Computers and Communications*, pages 553–558, 2002.

[12] B. Grofman and G. Owen. A game theoretic approach to measuring degree of centrality in social networks. *Social Networks*, 4:213–224, 1982.

[13] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[14] V. Krebs. Mapping networks of terrorist cells. *Connections*, 24:43–52, 2002.

[15] R. Lindelauf, H. Hamers, and B. Husslage. Cooperative game theoretic centrality analysis of terrorist networks: The cases of jemaah islamiyah and al qaeda. *European Journal of Operational Research*, 229(1):230 – 238, 2013.

[16] M. Maschler, E. Solan, and S. Zamir. *Game Theory*. CUP, 2013.

[17] R. Meir, Y. Zick, and J. S. Rosenschein. Optimization and stability in games with restricted interactions. In *CoopMAS-2012*, 2012.

[18] T. P. Michalak, T. Rahwan, P. L. Szczepanski, O. Skibski, R. Narayanam, M. J. Wooldridge, and N. R. Jennings. Computational analysis of connectivity games with applications to the investigation of terrorist networks. *IJCAI*, 2013.

[19] G. Moerkotte and T. Neumann. Analysis of two existing and one new dynamic programming algorithm for generation of optimal bushy join trees without cross products. *VLDB*, 2006.

[20] R. Myerson. Graphs and cooperation in games. *Math. Methods of OR*, 2(3):225–229, 1977.

[21] S. Nikolic, G. Kovacevic, A. Milicevic, and B. Trinajstic. The zagreb indices 30 years after. *Croatia Chemica Acta*, 76:113–124, 2003.

[22] S. Ressler. Social network analysis as an approach to combat terrorism: past, present and future research. *Homeland Security Affairs*, 2:1–10, 2006.

[23] L. S. Shapley. A value for n-person games. In *In Contributions to the Theory of Games*, pages 307–317. 1953.

[24] A. Sharafat and O. Marouzi. Recursive contraction algorithm: A novel and efficient graph traversal method for scanning all minimal cut sets. *Iranian Journal Of Science And Technology Transaction B- Eng.*, 30:749–761, 2006.

[25] T. Voice, S. D. Ramchurn, and N. R. Jennings. On coalition formation with sparse synergies. In *AAMAS*, 2012.

[7] Recently, the same breadth-first search algorithm of Moerkotte and Neumann was re-discovered by Voice et al.[25].