A Tractable and Expressive Class of Marginal Contribution Nets and Its Applications

Edith Elkind
Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ, U.K.

Leslie Ann Goldberg, Paul W. Goldberg, Michael Wooldridge
Computer Science
University of Liverpool
Liverpool L69 3BX, U.K.

ABSTRACT

Coalitional games raise a number of important questions from the point of view of computer science, key among them being how to represent such games compactly, and how to efficiently compute solution concepts assuming such representations. *Marginal contribution nets* (MC-nets), introduced by Ieong and Shoham, are one of the simplest and most influential representation schemes for coalitional games. MC-nets are a rule-based formalism, in which rules take the form *pattern* $\longrightarrow$ *value*, where "*pattern*" is a Boolean condition over agents, and "*value*" is a numeric value. Ieong and Shoham showed that, for a class of what we will call "basic" MC-nets, where patterns are constrained to be a conjunction of literals, marginal contribution nets permit the easy computation of solution concepts such as the Shapley value. However, there are very natural classes of coalitional game that require an exponential number of such basic MC-net rules. We present *read-once MC-nets*, a new class of MC-nets that is provably more compact than basic MC-nets, while retaining the attractive computational properties of basic MC-nets. We show how the techniques we develop for read-once MC-nets can be applied to other domains, in particular, computing solution concepts in network flow games on series-parallel networks.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems; I.2.4 [**Knowledge representation formalisms and methods**]; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Theory, Economics

Keywords

coalitional games, representation, complexity, marginal contribution nets

1. INTRODUCTION

Cite as: A Tractable and Expressive Class of Marginal Contribution Nets and Its Applications, E. Elkind, L. A. Goldberg, P. W. Goldberg, and M. Wooldridge, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.
Copyright © 2008, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

Coalitional games raise a number of important questions from the point of view of computer science, key among them being how to represent such games compactly, and how to efficiently compute solution concepts assuming such compact representations. The aim is typically to develop a representation that is compact for cases of interest, and yet allows the efficient computation of solution concepts. Accordingly, a number of compact representation schemes for coalitional games have been proposed, with various computational properties. For example, Deng and Papadimitriou [3] consider a representation of coalitional games based on weighted graphs: nodes in a graph correspond to players, and to compute the value of a particular coalition C, one computes the total weight of the sub-graph induced by C. The representation is compact because the size of a game represented in this way is at most quadratic in the number of players. Deng and Papadimitriou showed that for this representation, computing the Shapley value, one of the key solution concepts in coalitional games, is computationally easy. However, the weighted subgraph representation is not *complete*, in that there are some coalitional games which simply cannot be represented using this approach. An obvious research question is therefore to consider how to generalise the approach of [3] as far as possible, without losing its desirable properties. One of the simplest and most influential such generalisations is the *marginal contribution nets* (MC-nets) scheme of Ieong and Shoham [6]. MC-nets are a rule-based formalism, in which rules take the form *pattern* $\longrightarrow$ *value*, where "*pattern*" is a Boolean condition over agents, and "*value*" is a numeric value. Ieong and Shoham showed that, for a class of what we will call "basic" MC-nets, where patterns are constrained to be a conjunction of literals, marginal contribution nets permit the easy computation of solution concepts such as the Shapley value. However, there are very natural classes of coalitional games that require an exponential number of such MC-net rules. While it is easy to define more compact generalisations of MC-nets, the most obvious such extensions lose the attractive tractability properties of basic MC-nets. The following two questions therefore suggest themselves: (i) Is it possible to extend MC-nets, to allow compact representation of those cases where basic MC-nets fail, while retaining the desirable computational properties of basic MC-nets? (ii) As we generalise basic MC-nets, at what point does the representation cross the threshold from tractable to intractable? It is to these two questions – and in particular the first – that we address ourselves in this paper. Our main contributions

are as follows.

First, we show that the obvious generalisation of basic MC-nets leads to intractability with respect to computing the Shapley value. We then present a simple and intuitive generalisation of basic MC-nets called *read-once* MC-nets. We show that read-once MC-nets are exponentially more compact than basic MC-nets, in that there exists a very natural class of coalitional games that can be represented compactly using read-once MC-nets, but which would require an exponential number of basic MC-net rules, at least if we restrict ourselves to rules with positive values. We then prove that read-once MC-nets retain the desirable computational properties of basic MC-nets: in particular, it is possible to compute in polynomial time the Shapley value of players in a game represented as a read-once MC-net. We then demonstrate that our approach has wider applicability. We show that our algorithm for the Shapley value on read-once MC-nets yields a pseudo-polynomial time algorithm for computing the Shapley value in network flow games on series-parallel networks, which were introduced in multi-agent systems in [1]. Moreover, this algorithm can also be used to compute Banzhaf power index in such games. We begin, in the following section, by recalling the basic framework of coalitional games and MC-nets.

2. PRELIMINARIES AND NOTATION

In this section, we provide definitions of some of the basic concepts in coalitional game theory.

2.1 Coalitional Games

A *coalitional game* $G = (I, v)$ is given by a set of agents $I = \{x_1, \ldots, x_n\}$, $|I| = n$, and a function $v : 2^I \to \mathbb{R}$ that maps any subset (coalition) of the agents to a real value. This value is the total utility these agents can guarantee to themselves when working together. To simplify notation, we will sometimes write i instead of x_i.

A coalitional game is *simple* if v can only take values 0 and 1, i.e., $v : 2^I \to \{0, 1\}$. In such games, we say that a coalition $C \subseteq I$ *wins* if $v(C) = 1$, and *loses* if $v(C) = 0$. An agent i is *critical*, or *pivotal*, to a winning coalition C if the agent's removal from that coalition would make it a losing coalition: $v(C) = 1$, $v(C \setminus \{i\}) = 0$.

2.2 The Shapley Value

The Shapley value of an agent captures his marginal contribution to possible coalitions. Let Π be the set of all permutations (orderings) of n agents. Each $\pi \in \Pi$ is a one-to-one mapping from $\{1, \ldots, n\}$ to $\{1, \ldots, n\}$. Denote by $S_\pi(i)$ the predecessors of agent i in π, i.e., $S_\pi(i) = \{j \mid \pi(j) < \pi(i)\}$. The Shapley value of the ith agent in a game $G = (I, v)$ is denoted by $\phi_G(i)$ and is given by the following expression:

$$\phi_G(i) = \frac{1}{n!} \sum_{\pi \in \Pi} [v(S_\pi(i) \cup \{i\}) - v(S_\pi(i))]. \qquad (1)$$

We will occasionally abuse notation and say that an agent i is pivotal for a permutation π if it is pivotal for the coalition $S_\pi(i) \cup \{i\}$. Also, we will sometimes omit the index G if it is clear from the context.

For simple games, the formula (1) counts the fraction of all orderings of the agents in which agent i is critical for the coalition formed by his predecessors and himself. It thus reflects the assumption that when forming a coalition, any *ordering* of the agents entering the coalition has an equal probability of occurring, and expresses the probability that agent i is critical. In general, the Shapley value can be viewed as an agent's expected marginal contribution to the value of a coalition, under the probability model described above.

The Shapley value is the only payoff division scheme that satisfies the following natural axioms:

Efficiency: It fully distributes the total payoff available to the agents:

$$\sum_{i=1}^{n} \phi_G(i) = v(I).$$

Symmetry: If agents i and j are interchangeable, i.e., for any $S \subseteq I \setminus \{i, j\}$ we have $v(S \cup \{i\}) = v(S \cup \{j\})$, then $\phi_G(i) = \phi_G(j)$.

Dummy: If an agent i does not contribute to any coalition, i.e., $v(S) = v(S \cup \{i\})$ for any $S \subseteq I \setminus \{i\}$, then $\phi_G(i) = 0$.

Additivity: For any two coalitional games $G = (I, v)$ and $G' = (I, v')$ defined over the same set of agents I, for the coalitional game $G + G'$ given by $(G + G') = (I, v + w)$ we have

$$\phi_{G+G'}(i) = \phi_G(i) + \phi_{G'}(i).$$

2.3 Banzhaf Power Index

The Banzhaf power index was originally defined in the context of weighted voting games. However, it can be used to measure an agent's power in any coalitional game. Similarly to the Shapley value, it reflects the agent's expected marginal contribution to the value of a coalition. However, the underlying probabilistic model of coalition formation is different: rather than assuming that the agents join the coalition in random order and thus all permutations of the agents are equally likely, it assigns equal probability to all 2^n possible coalitions. Formally, the Banzhaf share $\beta_G(i)$ of an agent i in a game G is computed as follows:

$$\beta_G(i) = \frac{1}{2^{n-1}} \sum_{S : i \notin S} [v(S \cup \{i\}) - v(S)]. \qquad (2)$$

While Banzhaf power index satisfies the symmetry axiom and the dummy axiom, it may violate the efficiency axiom as well as the additivity axiom.

3. MARGINAL CONTRIBUTION NETS

In this section, we introduce the notion of marginal contribution networks, overview the previous results on computing various solution concepts for a restricted subclass of MC-nets, and show how to extend these results to a larger class of MC-nets. We also discuss the limitations of our approach, showing that in the most general setting the problem becomes NP-hard.

3.1 Previous Work

Marginal contribution networks (MC-nets) were proposed by Ieong and Shoham [6] in 2005. MC-nets provide a flexible and fully expressive representation scheme, which describes

a coalitional game using a set of rules. In more detail, a *marginal contribution network* is given by a set of agents $I = \{x_1, \ldots, x_n\}$ and a finite collection of *rules* of the form $(P \rightarrow V)$, where $V \in \mathbb{R}$ is the *value* of the rule, and P, which is a Boolean expression over the set I, is the *pattern* of the rule. A set $S \subseteq I$ of agents is said to *meet the requirements* of a given pattern P (denoted by $S \models P$) if P evaluates to `true` when the values of all Boolean variables that correspond to agents in S are set to `true`, and the values of all Boolean variables that correspond to agents in $I \setminus S$ are set to `false`. The value $v(S)$ of a group of agents S is defined to be the sum over the values of all rules that apply to it, i.e.,

$$v(S) = \sum_{P \rightarrow V : S \models P} V.$$

While the definition of MC-nets proposed by Ieong and Shoham [6] is quite general, the computational results in [6] are limited to a special case of this representation, namely, the setting where the patterns in all rules are required to be conjunctions of literals, i.e., expressions of the form

$$x_{i_1} \wedge \cdots \wedge x_{i_k} \wedge \bar{x}_{j_1} \wedge \cdots \wedge \bar{x}_{j_l}.$$

In the rest of the paper, we call such rules *basic*, and say that an MC-net is *basic* if it only uses basic rules. In [6], the authors show (Proposition 1, [6]) that even this restricted class of marginal contribution networks is universal, i.e., can be used to represent *any* coalitional game (perhaps using an exponential number of rules). Moreover, they show that this representation is at least as succinct as some other well-known representation languages, such as multi-issue representation [2] or graphical form representation [3], and in some cases can be exponentially more succinct.

Finally, [6] provides a polynomial-time algorithm for computing the Shapley value in basic MC-nets. It also discusses the problem of determining whether a given payoff vector is in the core of such a game, and provides a polynomial-time algorithm for basic MC-nets whose *graphical representation* has bounded treewidth.

3.2 Limitations of Basic MC-Nets

In basic MC-nets, the formulas used in the rules are required to be conjunctions of literals. For example, $x_1 \wedge \bar{x}_2 \rightarrow V$ is a rule that can appear in a basic MC-net. However, when using coalitional games to describe multi-agent systems, in addition to explicitly listing the agents whose presence/absence is necessary for completing a task, one may want to express the fact that some of the agents are substitutes for a particular task. A natural way to represent this would be by using the $\vee$ connective. For instance, suppose that a certain task can be completed by a team that includes agent x_1 and one of the agents x_2 and x_3 (e.g., because x_2 and x_3 have identical skills, but x_1 has a unique set of skills different from that of x_2 and x_2). This information can be represented by a rule of the form

$$(x_1 \wedge (x_2 \vee x_3)) \rightarrow V, \qquad (*)$$

where V is the value of this task. However, $x_1 \wedge (x_2 \vee x_3)$ is not a conjunction and hence $(*)$ is not a basic rule.

To overcome this difficulty, in [6], the authors show how to represent any coalitional game using basic rules. Indeed, if we want to describe a coalitional game that corresponds to (*), we can do so using two basic rules, namely, $x_1 \wedge x_2 \rightarrow V$ and $x_1 \wedge x_3 \wedge \bar{x}_2 \rightarrow V$. However, in some cases such transformation may lead to a superpolynomial blow-up in the representation size, at least if we restrict the values associated with rules to be non-negative, as we do in this paper. Consider the following example.

EXAMPLE 1. Consider a set of $2n$ agents, where agents $2i$ and $2i-1$ have identical skills. Suppose that there is a task of value $V > 0$ that can be completed using all n skills. This situation is naturally represented by the following rule:

$$(x_1 \vee x_2) \wedge \cdots \wedge (x_{2n-1} \vee x_{2n}) \rightarrow V. \qquad (3)$$

However, we will now show that to replace this rule with a collection of basic rules with non-negative values, we will need a superpolynomial number of rules.

THEOREM 1. *In Example 1, one needs $\Omega(2^n)$ basic rules with non-negative values* [1] *to represent the game given by (3).*

PROOF. The game given by rule (3) will be represented using a collection of basic rules, each of which is satisfiable. Observe that for all i in $\{1, \ldots, n\}$, every basic rule in the collection must contain a non-empty subset of $\{x_{2i}, x_{2i-1}\}$. If that is not the case for some i, consider the set R of rules that do not contain those literals, where $R \neq \emptyset$. If we put $x_{2i} = x_{2i-1} =$ `false`, rules in R should be satisfied by some values of the remaining Boolean variables, and should give associated positive values to the coalition satisfying them. That is a contradiction, since the total value should be 0 if $x_{2i} = x_{2i-1} =$ `false`.

Define a *minimum satisfying assignment* (MSA) to be one where for each pair $\{x_{2i}, x_{2i-1}\}$, only one of them is set to `true`. There are 2^n MSAs. We claim that the set of basic rules satisfied by any MSA does not intersect with the rules satisfied by any other MSA, implying that there should be at least 2^n basic rules. To see this, note that for any two distinct MSAs there is some i such that one of them has $x_{2i} =$ `true` and the other has $x_{2i-1} =$ `true`. All rules satisfied by the MSA with $x_{2i} =$ `true` must contain the literal x_{2i}—this is due to the observation that at least one of $\{x_{2i}, x_{2i-1}\}$ must be present, and the literal x_{2i-1} would not be satisfied. However, those rules cannot be satisfied by the other MSA. □

3.3 Intractability of Computing Shapley Value for General MC-nets

The results of the previous section suggest that we can obtain exponentially more compact representation by allowing the use of arbitrary Boolean expressions in the patterns. However, if we do so, we may lose an important property possessed by basic rules, namely, that of polynomial-time computability of agents' payoffs under classical value division schemes.

Indeed, [6] shows how to efficiently compute Shapley values for basic marginal contribution nets. First, the authors observe that the Shapley value is additive over rules, and then they give an explicit formula for the Shapley value of each agent under a given basic rule. This provides a polynomial-time algorithm for computing Shapley values of all agents in basic MC-nets. However, for rules that use arbitrary Boolean formulas such an algorithm is unlikely to exist.

[1] We believe that Theorem 1 remains true even if we do not restrict the values to be non-negative. However, the proof presented here relies on the non-negativity assumption in an essential way.

THEOREM 2. *It is NP-hard to compute agents' Shapley values in general marginal contribution nets, i.e., MC-nets in which the patterns in the rules can be arbitrary Boolean expressions. This holds even if the coalitional game in question is described by a single rule.*

PROOF. The proof is by reduction from 3-SAT. An instance of 3-SAT is given by a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, and a set of clauses $c_1, \ldots, c_m$, where each clause is a disjunction of at most three literals, i.e., the variables from X or their negations. It is a "yes"-instance if the formula $c_1 \wedge \cdots \wedge\ c_m$ is satisfiable and a "no"-instance otherwise.

Given an instance of 3-SAT, consider a coalitional game with $n+1$ agents, and a rule of the form $c_1 \wedge \cdots \wedge c_m \wedge x_{n+1} \rightarrow 1$, where x_{n+1} is a variable that does not appear in X. It is easy to see that the Shapley value of the $(n+1)$st player is greater than zero if and only if the original instance of 3-SAT is satisfiable. As a corollary, it is NP-hard to approximate the Shapley value with any positive multiplicative error; for if we could efficiently approximate the Shapley value, then we could certainly check whether a player's Shapley value is greater than zero. □

REMARK 3. Our hardness result also applies to computing Banzhaf power shares. Indeed, the Banzhaf index satisfies the dummy axiom. Therefore, the proof of Theorem 2 shows that the original instance of 3-SAT is satisfiable if and only if the Banzhaf power share of the $(n+1)$st player is greater than zero.

3.4 Read-once MC-nets

We have seen that there are natural coalitional games that cannot be compactly described by basic MC-nets. On the other hand, the previous subsection shows that using arbitrary patterns in MC-nets leads to computational intractability. Therefore, we would like to identify a class of Boolean formulas that is more expressive than conjunctions (and, in particular, is rich enough to express the fact that certain agents are substitutes with respect to a certain subtask), and yet allows for polynomial-time computation of Shapley values. In what follows, we show that these two conditions are satisfied by the class of *read-once Boolean formulas*. Informally, in a read-once Boolean formula, each variable can only appear once. This condition has a natural interpretation in the coalitional game setting: each agent has a set of skills/disabilities that can be useful/harmful in achieving a certain subtask, the agents can be substitutes with respect to a subtask, but no agent can contribute to more than one subtask. The subtasks are then combined into a task using the Boolean connectives $\wedge$ and $\vee$. The formal definition is given below.

DEFINITION 4. *A* read-once Boolean formula *is a binary rooted tree in which each internal node is labelled with $\wedge$ or $\vee$, the leaves are labelled with literals (i.e., variables or their negations), and each variable appears in at most one leaf.*

REMARK 5. Our algorithm for computing the Shapley value (presented below) can be extended to the case when negations appear as nodes of indegree one throughout the tree. However, it is easy to see that any such formula can be transformed into one of of the form described in Definition 4 with at most polynomial overhead. Therefore, in what follows we restrict our attention to the formulas where negations can only appear in the leaves.

`Shapley`$(N = [\{x_1, \ldots, x_n\}, (P_1 \rightarrow V_1), \ldots, (P_r \rightarrow V_r)])$

```
1.  set t_1 = t_2 ··· = t_n = 0;
2.  for j = 1, ..., r
3.    [A, B, T, F] =Sh(P_j);
4.    for i = 1, ..., n
5.      if P_j contains x_i
6.        set v_i = (1/n!) Σ_{k=0}^{n} k!(n-k-1)! A_{k,i};
7.      if P_j contains x̄_i
8.        set v_i = -(1/n!) Σ_{k=0}^{n} k!(n-k-1)! B_{k,i};
9.      if P_j contains neither, v_i = 0;
10.     set t_i = t_i + V_j · v_i;
11. return (t_1, ..., t_n);
```

Figure 1: Algorithm `Shapley`(N). (The subroutine Sh(F) is given in the appendix and described informally in the proof of Theorem 6, where relevant definitions are given.)

We will refer to MC-nets that use read-once Boolean formulas in the patterns as *read-once marginal contribution networks*. It is easy to see that the formula used in Example 1 is, in fact, a read-once Boolean formula, so this representation can be considerably more compact than that of [6], at least if we restrict ourselves to non-negative values. On the other hand, we will now show that for this class of rules it is still possible to compute the Shapley values of all players in polynomial time.

THEOREM 6. *Given an MC-net of the form $N = [I, (P_1 \rightarrow V_1), \ldots, (P_r \rightarrow V_r)]$, where $I = \{x_1, \ldots, x_n\}$, $V_1, \ldots, V_r \in \mathbb{Z}$, and $P_1, \ldots, P_r$ are read-once Boolean formulas, the algorithm* `Shapley(N)` *presented in Figure 1 computes the vector of Shapley values of all players $(\phi_N(1), \ldots, \phi_N(n))$ and runs in time* $\mathrm{poly}(n, r, \max_j \log |V_j|)$.

PROOF. Let $X_{\mathcal{F}}$ be the set of agents corresponding to variables that appear in a formula $\mathcal{F}$. To simplify notation, we will denote the ith element of I by i rather than x_i. Recall that we say that a coalition X satisfies a formula $\mathcal{F}$ (and write $X \models \mathcal{F}$) if $\mathcal{F}$ is satisfied by a truth assignment in which the variables that correspond to agents in X are set to `true`, while the variables that correspond to agents not in X are set to `false`.

For each subformula $\mathcal{F}$ of an input formula P_j, for all $i \in X_{\mathcal{F}}$ and $k = 0, \ldots, n$, we now define the following quantities.

- $A_{k,i}(\mathcal{F}) = |\{X \subseteq X_{\mathcal{F}} : |X| = k, i \notin X, X \not\models \mathcal{F}, X \cup \{i\} \models \mathcal{F}\}|$
- $B_{k,i}(\mathcal{F}) = |\{X \subseteq X_{\mathcal{F}} : |X| = k, i \notin X, X \models \mathcal{F}, X \cup \{i\} \not\models \mathcal{F}\}|$
- $T_k(\mathcal{F}) = |\{X \subseteq X_{\mathcal{F}} : |X| = k, X \models \mathcal{F}\}|$
- $F_k(\mathcal{F}) = |\{X \subseteq X_{\mathcal{F}} : |X| = k, X \not\models \mathcal{F}\}|$

The first two of these quantities denote the number of subsets (of a given size k) of agents for which the addition of agent i causes a given formula to become respectively satisfied, or unsatisfied. The latter two denote the number of subsets (of size k) of agents that respectively satisfy, or fail to satisfy, a formula. These quantities are computed recursively as follows.

First, suppose that $\mathcal{F} = x_j$. We have $A_{0,i}(\mathcal{F}) = 1$ if $i = j$ and 0 otherwise, and $A_{k,i}(\mathcal{F}) = 0$ for $k > 0$. Similarly, $B_{0,i}(\mathcal{F}) = 0$ for all $k \geq 0$. Finally, $T_k(\mathcal{F}) = 1$ if $k = 1$ and 0 otherwise, $F_k(\mathcal{F}) = 1$ if $k = 0$ and 0 otherwise.

Next, suppose that $\mathcal{F} = \bar{x}_j$. We have $A_{k,i}(\mathcal{F}) = 0$ for any $k = 0, \ldots, n$, $B_{k,i}(\mathcal{F}) = 1$ if $k = 0$ and 0 otherwise, $T_k(\mathcal{F}) = 1$ if $k = 0$ and 0 otherwise, $F_k(\mathcal{F}) = 1$ if $k = 1$ and 0 otherwise.

Now, suppose that $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$, and we have computed $A_{k,i}(\mathcal{F}_j)$, $B_{k,i}(\mathcal{F}_j)$, $T_k(\mathcal{F}_j)$, and $F_k(\mathcal{F}_j)$ for $i = 1, \ldots, n$, $k = 0, \ldots, n$ and $j = 1, 2$.

Note that since $\mathcal{F}$ is a read-once formula, each variable x_i (or its negation) may appear in $\mathcal{F}_1$ or $\mathcal{F}_2$, but not in both of these subformulas. First, suppose that $i \in X_{\mathcal{F}_1}$. Consider a set $Y \subseteq X_{\mathcal{F}} \setminus \{i\}$ of size k such that $Y_1 = Y \cap X_{\mathcal{F}_1}$ has size s and $Y_2 = Y \cap X_{\mathcal{F}_2}$ has size $k - s$. We have $Y \not\models \mathcal{F}$ if and only if $Y_1 \not\models \mathcal{F}_1$ and $Y_2 \not\models \mathcal{F}_2$. Moreover, in this case $Y \cup \{i\} \models \mathcal{F}$ if and only if $Y_1 \cup \{i\} \models \mathcal{F}_1$. Therefore, Y can contribute to $A_{k,i}(\mathcal{F})$ if and only if Y_1 contributes to $A_{s,i}(\mathcal{F}_1)$ and Y_2 contributes to $F_{k-s}(\mathcal{F}_2)$. Consequently,

$$A_{k,i}(\mathcal{F}) = \sum_{s=0}^{k} A_{s,i}(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2).$$

Similarly, we have $Y \cup \{i\} \not\models \mathcal{F}$ if and only if $Y_1 \cup \{i\} \not\models \mathcal{F}_1$ and $Y_2 \not\models \mathcal{F}_2$. Moreover, in this case $Y \models \mathcal{F}$ if and only if $Y_1 \models \mathcal{F}_1$. Therefore, Y can contribute to $B_{k,i}(\mathcal{F})$ if and only if Y_1 contributes to $B_{s,i}(\mathcal{F}_1)$ and Y_2 contributes to $F_{k-s}(\mathcal{F}_2)$. Consequently,

$$B_{k,i}(\mathcal{F}) = \sum_{s=0}^{k} B_{s,i}(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2).$$

Similarly, if $i \in X_{\mathcal{F}_2}$, we have

$$\begin{aligned} A_{k,i}(\mathcal{F}) &= \sum_{s=0}^{k} A_{s,i}(\mathcal{F}_2) F_{k-s}(\mathcal{F}_1) \\ B_{k,i}(\mathcal{F}) &= \sum_{s=0}^{k} B_{s,i}(\mathcal{F}_2) F_{k-s}(\mathcal{F}_1). \end{aligned}$$

Finally, in both cases we have

$$\begin{aligned} T_k(\mathcal{F}) &= \sum_{s=0}^{k} (T_s(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2) + F_s(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2) + \\ &\quad T_s(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2)) \\ F_k(\mathcal{F}) &= \sum_{s=0}^{k} F_s(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2). \end{aligned}$$

The case when $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$ is similar. First, suppose that $i \in X_{\mathcal{F}_1}$. Consider a set $Y \subseteq X_{\mathcal{F}} \setminus \{i\}$ of size k such that $Y_1 = Y \cap X_{\mathcal{F}_1}$ has size s and $Y_2 = Y \cap X_{\mathcal{F}_2}$ has size $k - s$. We have $Y \cup \{i\} \models \mathcal{F}$ if and only if $Y_1 \cup \{i\} \models \mathcal{F}_1$ and $Y_2 \models \mathcal{F}_2$. Moreover, in this case $Y \not\models \mathcal{F}$ if and only if $Y_1 \not\models \mathcal{F}_1$. Therefore, Y can contribute to $A_{k,i}(\mathcal{F})$ if and only if Y_1 contributes to $A_{s,i}(\mathcal{F}_1)$ and Y_2 contributes to $T_{k-s}(\mathcal{F}_2)$. Consequently,

$$A_{k,i}(\mathcal{F}) = \sum_{s=0}^{k} A_{s,i}(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2).$$

Furthermore, we have $Y \models \mathcal{F}$ if and only if $Y_1 \models \mathcal{F}_1$ and $Y_2 \models \mathcal{F}_2$. Moreover, in this case $Y \cup \{i\} \not\models \mathcal{F}$ if and only if $Y_1 \cup \{i\} \not\models \mathcal{F}_1$. Therefore, Y can contribute to $B_{k,i}(\mathcal{F})$ if and only if Y_1 contributes to $B_{s,i}(\mathcal{F}_1)$ and Y_2 contributes to $T_{k-s}(\mathcal{F}_2)$. Consequently,

$$B_{k,i}(\mathcal{F}) = \sum_{s=0}^{k} B_{s,i}(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2).$$

Similarly, if $x_i \in X_{\mathcal{F}_2}$, we have

$$\begin{aligned} A_{k,i}(\mathcal{F}) &= \sum_{s=0}^{k} A_{s,i}(\mathcal{F}_2) T_{k-s}(\mathcal{F}_1) \\ B_{k,i}(\mathcal{F}) &= \sum_{s=0}^{k} B_{s,i}(\mathcal{F}_2) T_{k-s}(\mathcal{F}_1). \end{aligned}$$

Also, in both cases we have

$$\begin{aligned} T_k(\mathcal{F}) &= \sum_{s=0}^{k} T_s(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2) \\ F_k(\mathcal{F}) &= \sum_{s=0}^{k} (F_s(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2) + F_s(\mathcal{F}_1) T_{k-s}(\mathcal{F}_2) + \\ &\quad T_s(\mathcal{F}_1) F_{k-s}(\mathcal{F}_2)). \end{aligned}$$

The algorithm $\texttt{Sh}(\mathcal{F})$ for recursive computation of $A_{k,i}(\mathcal{F})$, $B_{k,i}(\mathcal{F})$, $T_k(\mathcal{F})$, and $F_k(\mathcal{F})$ outlined above is formally described in the appendix. For any formula $\mathcal{F}$, the algorithm $\texttt{Sh}(\mathcal{F})$ returns four lists of numbers, namely, $\mathbf{A} = \{A_{k,i} \mid k = 0, \ldots, n, i = 1, \ldots, n\}$, $\mathbf{B} = \{B_{k,i} \mid k = 0, \ldots, n, i = 1, \ldots, n\}$, $\mathbf{T} = \{T_k \mid k = 0, \ldots, n\}$, and $\mathbf{F} = \{F_k \mid k = 0, \ldots, n\}$. These lists correspond to the quantities $A_{k,i}(\mathcal{F})$, $B_{k,i}(\mathcal{F})$, $T_k(\mathcal{F})$, and $F_k(\mathcal{F})$ defined above.

Our main algorithm $\texttt{Shapley}(N)$ is described in Figure 1. It uses the algorithm $\texttt{Sh}(\mathcal{F})$ as a subroutine. To prove the correctness of our algorithm, it remains to show that lines 5–8 of our algorithm correctly compute the Shapley value of agent i in the MC-net N_j given by a single rule $P_j \to 1$: by the linearity of the Shapley value, this would imply that the output of our algorithm is indeed the list of Shapley values of the agents in N.

Consider an agent i in the game described by N_j. As P_j is a read-once formula, it can contain x_i or $\bar{x}_i$, but not both. If the formula P_j contains x_i, then i's contribution to the value of any coalition is non-negative. Furthermore, for any coalition of size k counted in $A_{k,i}(P_j)$, the agent i contributes 1 to the value of this coalition if he appears in a permutation right after the members of this coalition. Therefore, we have

$$\phi_{N_j}(i) = \frac{1}{n!} \sum_{k=0}^{n} k!(n-k-1)! A_{k,i}(P_j). \tag{4}$$

Similarly, if the formula P_j contains $\bar{x}_i$, then i's contribution to the value of any coalition in N_j is non-positive and can be computed as

$$\phi_{N_j}(i) = -\frac{1}{n!} \sum_{k=0}^{n} k!(n-k-1)! B_{k,i}(P_j). \tag{5}$$

This is exactly the formulas used by our algorithm to compute the Shapley values of all players in N_j. We conclude that $\texttt{Shapley}(N)$ correctly computes the Shapley values of all agents. Moreover, it is clear that the running time of our algorithm is polynomial in n, r, and $\max_j \log |V_j|$. □

REMARK 7. A similar algorithm can also be used to compute the Banzhaf power share of each player as long as the value of each coalition is given by a single rule based on a read-once Boolean formula. For any agent i and any $k = 0, \ldots, n$, our algorithm computes the number of coalitions C of size k such that adding i to C changes the value of the coalition. Therefore, if the formula $\mathcal{F}$ contains x_i, we have

$$\beta(i) = \frac{1}{2^{n-1}} \sum_{k=0}^{n} A_{k,i}(\mathcal{F}),$$

and if the formula $\mathcal{F}$ contains $\bar{x}_i$, we have

$$\beta(i) = -\frac{1}{2^{n-1}} \sum_{k=0}^{n} B_{k,i}(\mathcal{F}).$$

However, as Banzhaf power shares do not satisfy the additivity axiom, our algorithm does not give a way to directly compute the Banzhaf values in games that are described by MC-nets with two or more rules, even if these rules are basic.

4. NETWORK FLOW GAMES

In this section, we show how to apply the techniques of Section 3 to computing Shapley values and Banzhaf power shares in network flow games.

4.1 Previous Work

Network flow games have been studied in the game theory literature [4, 5], and, more recently, in the context of multiagent systems [1]. In this paper, we will use the model of [1], which views a network flow game as a simple game, where agents correspond to edges in the network, and a coalition wins if it admits a flow of a given size and loses otherwise. We will first define the network flow problem, and then introduce the respective game.

DEFINITION 8. *A* network flow problem $\mathcal{N} = (V, E, s, t, \mathbf{c})$ *is given by a set of* nodes V*, a set of* edges $E \subseteq V \times V$*, a* source $s \in V$*, a* sink $t \in V$*, and a list of* capacities $\mathbf{c} = \{c_e\}_{e \in E}$*,* $c_e > 0$*. We say that* $\mathbf{f} = \{f_e\}_{e \in E}$ *is a valid flow of size* k *in* $\mathcal{N}$ *if*

- *(capacity constraints):* $f_e \leq c_e$ *for all* $e \in E$*.*
- *(flow preservation constraints): for any* $v \in V \setminus \{s, t\}$*,*

$$\sum_{(v',v) \in E} f_{(v',v)} = \sum_{(v,v'') \in E} f_{(v,v'')}.$$

- *(size of the flow):*

$$\sum_{(s,v) \in E} f_{(s,v)} = \sum_{(w,t) \in E} f_{(w,t)} = k.$$

We are now ready to define the class of network flow games. (Note that we assume that all capacities are integer.)

DEFINITION 9. *A* network flow game *is given by a* network $\mathcal{N} = (V, E, s, t, \mathbf{c})$ *and a* target flow value K*. Each edge* $e \in E$ *is controlled by a different agent; therefore, we identify the set of agents with the set of edges. A coalition* $E' \subseteq E$ *is* winning *if the network* $\mathcal{N}' = (V, E', s, t, \mathbf{c}')$*, where* $\mathbf{c}' = \{c_e\}_{e \in E'}$*, admits a flow of size* K *from* s *to* t *and* losing *otherwise.*

Connectivity games are a special class of network flow games that satisfy $K = 1$ and $c_e = 1$ for any $e \in E$. In other words, in a connectivity game, a coalition is winning if it contains a path from s to t and losing otherwise.

Value division schemes for network flow games were first studied by Bachrach and Rosenschein [1]. Paper [1] shows that for general networks, i.e., ones where there is no restrictions on the structure of the underlying network $\mathcal{N}$, and the capacities and the target flow value are integer numbers given in binary, computing Banzhaf shares of all players is $\#P$-hard. On the positive side, they provide a polynomial-time algorithm for computing the Banzhaf shares in the special case of connectivity games on *bounded-layer networks* (for the definition of a bounded-layer network, see [1]).

4.2 Flow Games on Series-Parallel Networks and MC-nets

In this section, we extend the results of [1] by providing a polynomial-time algorithm for power indices in another important class of networks, namely, series-parallel networks. This is a class of networks with a clear hierarchical structure that enables us to use ideas from the previous section. Our algorithm can be used to compute both Shapley and Banzhaf values. Moreover, unlike the algorithm of [1], it is not restricted to connectivity games, but can be applied for an arbitrary target flow value K. However, the running time of our algorithm is exponential in K. Therefore, our algorithm for general network flow games on series-parallel graphs is a *pseudopolynomial* algorithm, i.e., an algorithm whose running time is polynomial if all input values (i.e., the edge capacities and the target flow value) are given in unary. Alternatively, it can be said to run in polynomial time if all edge capacities and the target flow value are polynomially bounded, which is a realistic scenario in many applications. In Section 4.3, we will see that this is essentially the best we can do, as even for series-parallel networks the problem of computing the Shapley value is NP-hard and is therefore unlikely to have a polynomial-time algorithm.

We start by formally defining series-parallel networks.

DEFINITION 10. *A series-parallel network (SPN) is a network* $\mathcal{N} = (V, E, s, t, \mathbf{c})$*, such that one of the following conditions holds:*

> **Base case:** *A single edge* (s, t)*, i.e., a network of the form* $(\{s, t\}, \{(s, t)\}, s, t, c_{(s,t)})$ *is an SPN.*
>
> **Series:** *Suppose that* $\mathcal{N}^1 = (V^1, E^1, s^1, t^1, \mathbf{c}^1)$ *and* $\mathcal{N}^2 = (V^2, E^2, s^2, t^2, \mathbf{c}^2)$ *are SPN such that* $V^1 \cap V^2 = \emptyset$*. Set* $V = V^1 \cup V^2$*,* $E = E^1 \cup E^2$*, merge* t^1 *with* s^2*, and set* $\mathbf{c} = \mathbf{c}^1 \cup \mathbf{c}^2$*. Then* $(V, E, s = s^1, t = t^2, \mathbf{c})$ *is an SPN.*
>
> **Parallel:** *Suppose that* $\mathcal{N}^1 = (V^1, E^1, s^1, t^1, \mathbf{c}^1)$ *and* $\mathcal{N}^2 = (V^2, E^2, s^2, t^2, \mathbf{c}^2)$ *are SPN such that* $V^1 \cap V^2 = \emptyset$*. Set* $V = V^1 \cup V^2$*,* $E = E^1 \cup E^2$*, merge* s^1 *with* s^2 *and* t^1 *with* t^2*, and set* $\mathbf{c} = \mathbf{c}^1 \cup \mathbf{c}^2$*. Then* $(V, E, s = s^1 = s^2, t = t^1 = t^2, \mathbf{c})$ *is an SPN.*

We will now show that even though network flow games and marginal contribution nets appear to be quite different, we can re-use the algorithm of the previous section to compute the power indices in the connectivity game on a series-parallel network. To do so, we will now establish a

connection between series-parallel networks and read-once Boolean formulas. For connectivity games we have $c_e = 1$ for all $e \in E$, so in this reduction we omit the list of capacities from the description of a network.

Given a series-parallel network $\mathcal{N} = (V, E, s, t)$, $E = \{e_1, \ldots, e_n\}$, we will recursively construct a corresponding read-once Boolean formula $\mathcal{F}_\mathcal{N}$, $X_\mathcal{F} = \{x_1, \ldots, x_n\}$ as follows. If $\mathcal{N}$ consists of a single edge e_i, set $F_\mathcal{N} = x_i$. If $\mathcal{N}$ has been obtained by connecting two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ in parallel, set $\mathcal{F}_\mathcal{N} = \mathcal{F}_{\mathcal{N}_1} \vee \mathcal{F}_{\mathcal{N}_2}$. Finally, if $\mathcal{N}$ has been obtained by connecting two networks $\mathcal{N}_1$ and $\mathcal{N}_2$ in series, set $\mathcal{F}_\mathcal{N} = \mathcal{F}_{\mathcal{N}_1} \wedge \mathcal{F}_{\mathcal{N}_2}$.

It is easy to see that the resulting formula is read-once, and this transformation can be performed in polynomial time. Moreover, $S = \{e_{i_1}, \ldots, e_{i_t}\}$ is a winning coalition in a connectivity game on the network $\mathcal{N}$ if and only if $S' = \{x_{i_1}, \ldots, x_{i_t}\}$ is a winning coalition in the marginal contribution net described by the rule $\mathcal{F}_\mathcal{N} \to 1$. Consequently, the Shapley value of any player e_i in the original game is the same as the Shapley value of its counterpart x_i in the new game. The same is true for the Banzhaf power shares. Consequently, we have the following theorem.

THEOREM 11. *There exists a polynomial-time algorithm to compute Shapley values and Banzhaf power shares in connectivity games on series-parallel graphs.*

To generalise this result to network flow games, we need to modify the algorithm of Section 3 to take into account the size of the flow.

THEOREM 12. *There is an algorithm that, given a series-parallel network $\mathcal{N} = (V, E, s, t, \mathbf{c})$, $c_i \in \mathbb{Z}$ for $i \in E$, $|E| = n$, and a target flow value K, computes the Shapley values of all players in the game $(\mathcal{N}, K)$, and runs in time polynomial in n, $\sum_{e\in E} c_e$, and K.*

PROOF. Given a network $\mathcal{N} = (V, E, s, t, \mathbf{c})$, let $f(\mathcal{N})$ denote the size of the maximum flow in $\mathcal{N}$. It is well-known that $f(\mathcal{N})$ can be computed in time polynomial in n and $\log \sum_{e\in E} c_e$. Set $C = \sum_{e\in E} c_e$; clearly, $f(\mathcal{N}) \le C$. For any network $\mathcal{N} = (V, E, s, t, \mathbf{c})$ and any $S \subseteq E$, set $\mathcal{N}_S = (V, S, s, t, \{c_e\}_{e\in S})$. For all $k = 0, \ldots, n$, $e \in E$, $0 \le X < Y \le C$, let $A_{k,e,X,Y}(\mathcal{N})$ be the number of sets $S \subseteq E$ of size k such that $e \notin S$ and $f(\mathcal{N}_S) = X$, $f(\mathcal{N}_{S\cup\{e\}}) = Y$. Also, let $T_{k,X}$ be the number of sets $S \subseteq E$ of size k such that $f(\mathcal{N}_S) = X$.

Given the values $A_{k,e,X,Y}(\mathcal{N})$ for $k = 0, \ldots, n$, $X, Y = 0, \ldots, C$, it is easy to compute the Shapley value of the agent e. Indeed, for each coalition S counted in $A_{k,e,X,Y}(\mathcal{N})$ such that $X < K \le Y$, agent e contributes 1 to the value of the permutation where he appears directly after the members of S. Hence, we have

$$\phi_{(\mathcal{N},K)}(e) = \frac{1}{n!} \sum_{k=0}^{n} \sum_{\substack{X\le K \\ Y\ge K}} k!(n-k-1)! A_{k,e,X,Y}(\mathcal{N}).$$

It remains to show how to compute $A_{k,e,X,Y}(\mathcal{N})$ for all $k = 0, \ldots, n$, $e \in E$, $0 \le X < Y \le C$. This is done recursively, by simultaneously computing $A_{k,e,X,Y}$ and $T_{k,X}$ for subnetworks of the original network $\mathcal{N}$.

If $\mathcal{N}$ is a single edge e with capacity c_e, then we have $A_{k,e,X,Y}(\mathcal{N}) = 1$ if $k = 0$, $X = 0$ and $Y = c_e$; otherwise, $A_{k,e,X,Y}(\mathcal{N}) = 0$. Also, $T_{k,X}(\mathcal{N}) = 1$ if $k = 0$ and $X = 0$, or $k = 1$ and $X = c_e$, and $T_{k,X}(\mathcal{N}) = 0$ otherwise.

Now, suppose that $\mathcal{N}$ is obtained by connecting two networks $\mathcal{N}^1 = (V^1, E^1, s^1, t^1, \mathbf{c}^1)$ and $\mathcal{N}^2 = (V^2, E^2, s^2, t^2, \mathbf{c}^2)$ in parallel. For any $S \subseteq E$, $f(\mathcal{N}_S) = X$ if and only if for some $X' \le X$ we have $f(\mathcal{N}^2_{S\cap E^2}) = X'$ and $f(\mathcal{N}^1_{S\cap E^1}) = X - X'$. Hence, if $e \in E^1$, we have

$$A_{k,e,X,Y}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{X'=0}^{X} A_{k-k',e,X-X',Y-X'}(\mathcal{N}_1) T_{k',X'}(\mathcal{N}_2),$$

and if $e \in E^2$, we have

$$A_{k,e,X,Y}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{X'=0}^{X} A_{k-k',e,X-X',Y-X'}(\mathcal{N}_2) T_{k',X'}(\mathcal{N}_1).$$

Observe that if $X < Y$, then we have $X - X' < Y - X'$, so the quantity $A_{k-k',e,X-X',Y-X'}$ is well-defined. Also, we have

$$T_{k,X}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{X'=0}^{X} T_{k',X'}(\mathcal{N}_1) T_{k-k',X-X'}(\mathcal{N}_2).$$

The case where $\mathcal{N}$ is obtained by connecting two networks $\mathcal{N}^1 = (V^1, E^1, s^1, t^1, \mathbf{c}^1)$ and $\mathcal{N}^2 = (V^2, E^2, s^2, t^2, \mathbf{c}^2)$ in series is similar. Fix an edge $e \in E^1$ and consider a set $S \subseteq E \setminus \{e\}$. Suppose that $f(\mathcal{N}_S) = X$, $f(\mathcal{N}_{S\cup\{e\}}) = Y$, $X \le Y$. This can happen in two cases: either (i) $f(\mathcal{N}^1_{S\cap E^1}) = X$, $f(\mathcal{N}^1_{S\cup\{e\}\cap E^1}) = Y$ and $f(\mathcal{N}^2_{S\cap E^2}) \ge Y$, or (ii) $f(\mathcal{N}^1_{S\cap E^1}) = X$, $f(\mathcal{N}^1_{S\cup\{e\}\cap E^1}) = Y' > Y$ and $f(\mathcal{N}^2_{S\cap E^2}) = Y$.

Consequently, if $e \in E^1$, we have

$$A_{k,e,X,Y}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{Z=Y}^{C} A_{k',e,X,Y}(\mathcal{N}_1) T_{k-k',Z}(\mathcal{N}_2) + \sum_{k'=0}^{k} \sum_{Y'=Y+1}^{C} A_{k',e,X,Y'}(\mathcal{N}_1) T_{k-k',Y}(\mathcal{N}_2),$$

and if $e \in E^2$, we have

$$A_{k,e,X,Y}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{Z=Y}^{C} A_{k',e,X,Y}(\mathcal{N}_2) T_{k-k',Z}(\mathcal{N}_1) + \sum_{k'=0}^{k} \sum_{Y'=Y+1}^{C} A_{k',e,X,Y'}(\mathcal{N}_2) T_{k-k',Y}(\mathcal{N}_1).$$

Also, we have

$$T_{k,X}(\mathcal{N}) = \sum_{k'=0}^{k} \sum_{X'=X}^{C} [T_{k',X}(\mathcal{N}_1) T_{k-k',X'}(\mathcal{N}_2) + T_{k',X}(\mathcal{N}_2) T_{k-k',X'}(\mathcal{N}_1)].$$

This completes the description of the procedure for computing $A_{k,e,X,Y}(\mathcal{N})$, and hence the proof. □

REMARK 13. Just as in the previous section, once we have computed the values $A_{k,e,X,Y}(\mathcal{N})$, we can use them to determine the Banzhaf power shares of all players in time polynomial in K, n, and $\sum_{i\in E} c_i$.

4.3 Hardness Results for Large Capacities

The running time of the algorithm described in the previous subsection is polynomial in $\sum_{e\in E}\{c_e\}$ and K rather than $\log \sum_{e\in E}\{c_e\}$ and $\log K$, i.e., it is exponential in the

size of the input. This leaves open the question of designing a polynomial-time algorithm for computing the Shapley value in network flow games on series-parallel graphs. While [1] shows that this problem is #P-hard for general graphs, no similar result was known for our setting. In this section, we close this gap. Namely, we show that for series-parallel graphs where the edge capacities and the target flow value are given in binary, checking whether a player is not a dummy (and hence whether his Shapley value and Banzhaf power share are non-zero) is NP-hard.

Our reduction is from PARTITION, which is a well-known NP-complete problem.

DEFINITION 14. *An instance of* PARTITION *is given by a set of n integer weights $W = \{w_1, \ldots, w_n\}$. It is a "yes"-instance if it is possible to partition W into two subsets $W_1 \subseteq W$, $W_2 \subseteq W$ so that $W_1 \cap W_2 = \emptyset$, $W_1 \cup W_2 = W$, and the sum of the weights in each subset is equal: $\sum_{w_i \in W_1} w_i = \sum_{w_i \in W_2} w_i$. If there is no such partition, it is a "no"-instance.*

THEOREM 15. *Given a network flow game $G = (\mathcal{N}, K)$, where $\mathcal{N} = (V, E, s, t, \mathbf{c})$ is a series-parallel network, and an agent $e \in E$, it is NP-hard to decide whether e is not a dummy player.*

PROOF. Given an instance $W = \{w_1, \ldots, w_n\}$ of PARTITION, we construct a network game as follows. We set $V = \{s, v_1, \ldots, v_n, v_{n+1}, t\}$. There are edges in E from s to each v_i, $i = 1, \ldots, n+1$, and from each v_i, $i = 1, \ldots, n+1$, to t. The capacities are chosen as follows. Let $C_{\max} = \sum_i w_i$. Let the capacity of the edge (s, v_i) be $2w_i$ for $i = 1, \ldots, n$, and let the capacity of the edge (v_i, t) be $2C_{\max}$ for $i = 1, \ldots, n+1$. Finally, let the capacity of the edge (s, v_{n+1}) be 1. Set the target flow value to be $\sum_i w_i + 1$. It is easy to see that the agent that owns the edge (s, v_{n+1}) is a dummy if and only if the original instance of PARTITION is a "no"-instance. □

5. CONCLUSIONS

We have presented read-once MC-nets, which retain the attractive computational properties of basic MC-nets while being exponentially more compact. We have also demonstrated how the algorithmic ideas developed in the context of read-once MC-nets can be applied to other domains — in particular, network flow games on series-parallel networks.

Many issues suggest themselves for further study. Most obviously, one might investigate other constraints on the logical construction of MC-net rules which retain tractability while yielding further compactness. Results in the area of Boolean function complexity and representation will perhaps be of value here. Another issue is the extent to which our techniques can be applied to other classes of coalitional games, beyond network flow games.

Acknowledgements: This research was in part supported by the EPSRC under the grant "Market-Based Control".

6. REFERENCES

[1] Y. Bachrach and J. Rosenschein, Computing the Banzhaf power index in network flow games. In *Proc. AAMAS-2007*, pp. 323–329, 2007

[2] V. Conitzer and T. Sandholm, Computing Shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. *In Proc. AAAI-2004*, pp. 219–225, 2004

[3] X. Deng and C. H. Papadimitriou, On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19:257–266, May 1994

[4] E. Kalai and E. Zemel, On totally balanced games and games of flow. Discussion Papers 413, Northwestern University, Center for Mathematical Studies in Economics and Management Science, Jan. 1980, available at `http://ideas.repec.org/p/nwu/cmsems/413.html`

[5] E. Kalai and E. Zemel, Generalized network problems yielding totally balanced games. *Operations Research*, 30:998–1008, September 1982

[6] S. Ieong and Y. Shoham, Marginal contribution nets: a compact representation scheme for coalitional games. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, pp. 193–202, 2005

APPENDIX

A. THE PROCEDURE $\texttt{SH}(\mathcal{F})$

$\texttt{Sh}(\mathcal{F})$
  if $\mathcal{F} = x_j$ **then**
   for $k = 0, \ldots, n$
     if $k = 1$ **then** $T_k = 1$ **else** $T_k = 0$;
     if $k = 0$ **then** $F_k = 1$ **else** $F_k = 0$;
     for $i = 1, \ldots, n$
       if $(i = j,\ k = 0)$ **then** $A_{k,i} = 1$ **else** $A_{k,i} = 0$;
       $B_{k,i} = 0$;
  if $\mathcal{F} = \bar{x}_j$ **then**
   for $k = 0, \ldots, n$
     if $k = 0$ **then** $T_k = 1$ **else** $T_k = 0$;
     if $k = 1$ **then** $F_k = 1$ **else** $F_k = 0$;
     for $i = 1, \ldots, n$
       $A_{k,i} = 0$;
       if $(i = j,\ k = 0)$ **then** $B_{k,i} = 1$ **else** $B_{k,i} = 0$;
  if $\mathcal{F} = \mathcal{F}_1 \vee \mathcal{F}_2$ **then**
   $[\mathbf{A}^1, \mathbf{B}^1, \mathbf{T}^1, \mathbf{F}^1] = \texttt{Sh}(\mathcal{F}_1)$; $[\mathbf{A}^2, \mathbf{B}^2, \mathbf{T}^2, \mathbf{F}^2] = \texttt{Sh}(\mathcal{F}_2)$;
   for $k = 0, \ldots, n$
     $T_k = \sum_{s=0}^{k} (T_s^1 T_{k-s}^2 + F_s^1 T_{k-s}^2 + T_s^1 F_{k-s}^2)$
     $F_k = \sum_{s=0}^{k} F_s^1 F_{k-s}^2$
     for $i = 1, \ldots, n$
       if x_i **does not appear in** $\mathcal{F}$
          then $A_{k,i} = 0,\ B_{k,i} = 0$
       if x_i **appears in** $\mathcal{F}_1$ **then**
          $A_{k,i} = \sum_{s=0}^{k} A_{s,i}^1 F_{k-s}^2$; $B_{k,i} = \sum_{s=0}^{k} B_{s,i}^1 F_{k-s}^2$;
       if x_i **appears in** $\mathcal{F}_2$ **then**
          $A_{k,i} = \sum_{s=0}^{k} A_{s,i}^2 F_{k-s}^1$; $B_{k,i} = \sum_{s=0}^{k} B_{s,i}^2 F_{k-s}^1$;
  if $\mathcal{F} = \mathcal{F}_1 \wedge \mathcal{F}_2$ **then**
   $[\mathbf{A}^1, \mathbf{B}^1, \mathbf{T}^1, \mathbf{F}^1] = \texttt{Sh}(\mathcal{F}_1)$; $[\mathbf{A}^2, \mathbf{B}^2, \mathbf{T}^2, \mathbf{F}^2] = \texttt{Sh}(\mathcal{F}_2)$;
   for $k = 0, \ldots, n$
     $T_k = \sum_{s=0}^{k} T_s^1 T_{k-s}^2$
     $F_k = \sum_{s=0}^{k} (F_s^1 F_{k-s}^2 + F_s^1 T_{k-s}^2 + T_s^1 F_{k-s}^2)$
     for $i = 1, \ldots, n$
       if x_i **does not appear in** $\mathcal{F}$
          then $A_{k,i} = 0,\ B_{k,i} = 0$
       if x_i **appears in** $\mathcal{F}_1$ **then**
          $A_{k,i} = \sum_{s=0}^{k} A_{s,i}^1 T_{k-s}^2$; $B_{k,i} = \sum_{s=0}^{k} B_{s,i}^1 T_{k-s}^2$;
       if x_i **appears in** $\mathcal{F}_2$ **then**
          $A_{k,i} = \sum_{s=0}^{k} A_{s,i}^2 T_{k-s}^1$; $B_{k,i} = \sum_{s=0}^{k} B_{s,i}^2 T_{k-s}^1$;
  return $[\mathbf{A}, \mathbf{B}, \mathbf{T}, \mathbf{F}]$;