Handling Negative Value Rules in MC-net-based Coalition Structure Generation

Suguru Ueda[1], Takato Hasegawa[1], Naoyuki Hashimoto[1], Naoki Ohta[2],
Atsushi Iwasaki[1], and Makoto Yokoo[1]
[1]Kyushu University, Fukuoka, 819-0395, Japan
{ueda@agent., hasegawa@agent., hashimoto@agent., iwasaki@, yokoo@}inf.kyushu-u.ac.jp
[2]Ritsumeikan University, Shiga, 525-8577, Japan n-ohta@fc.ritsumei.ac.jp

ABSTRACT

A Coalition Structure Generation (CSG) problem involves partitioning a set of agents into coalitions so that the social surplus is maximized. Recently, Ohta *et al.* developed an efficient algorithm for solving CSG assuming that a characteristic function is represented by a set of rules, such as marginal contribution networks (MC-nets).

In this paper, we extend the formalization of CSG in Ohta *et al.* so that it can handle negative value rules. Here, we assume that a characteristic function is represented by either MC-nets (without externalities) or embedded MC-nets (with externalities). Allowing negative value rules is important since it can reduce the efforts for describing a characteristic function. In particular, in many realistic situations, it is natural to assume that a coalition has negative externalities to other coalitions.

To handle negative value rules, we examine the following three algorithms: (i) a full transformation algorithm, (ii) a partial transformation algorithm, and (iii) a direct encoding algorithm. We show that the full transformation algorithm is not scalable in MC-nets (the worst-case representation size is $\Omega(n^2)$, where n is the number of agents), and does not seem to be tractable in embedded MC-nets (representation size would be $\Omega(2^n)$). In contrast, by using the partial transformation or direct encoding algorithms, an exponential blow-up never occurs even for embedded MC-nets. For embedded MC-nets, the direct encoding algorithm creates less rules than the partial transformation algorithm.

Experimental evaluations show that the direct encoding algorithm is scalable, i.e., an off-the-shelf optimization package (CPLEX) can solve problem instances with 100 agents and rules within 10 seconds.

Categories and Subject Descriptors

I.2.11 [**ARTIFICIAL INTELLIGENCE**]: Distributed Artificial Intelligence – Multiagent systems

General Terms

Algorithms, Theory

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

Keywords

coalition structure generation, constraint optimization, cooperative games

1. INTRODUCTION

Coalition formation is an important capability in automated negotiation among self-interested agents. Coalition Structure Generation (CSG) involves partitioning a set of agents so that social surplus is maximized. This problem has become a popular research topic in AI and multi-agent systems. Solving CSG is equivalent to the *complete set partitioning problem* [17], and various algorithms for solving CSG have been developed [2, 9, 11, 12, 13, 15, 17]. Also, the computational complexity of CSG in various domains has been analyzed [1, 16].

Another active research area in the agent research community is compact representation schemes of a characteristic function. If we naively represent a characteristic function as a table, we require $\Theta(2^n)$ numbers, where n is the number of agents. When the number of agents becomes large, we need a compact method to represent a characteristic function. The main idea of compact representation schemes is to use a set of rules to represent a characteristic function. These compact representation schemes include marginal contribution networks (MC-nets) [5, 7], Synergy Coalition Groups (SCG) [4], etc. The computational complexity for finding various solution concepts when a characteristic function is represented by these compact representation schemes is analyzed in [6].

Recently, Ohta *et al.* [10] introduces an innovative direction for solving CSG by utilizing these compact representation schemes. More specifically, they show that a CSG problem can be formalized as a problem of finding the subset of rules that maximizes the sum of rule values under certain constraints. They also develop mixed integer programming (MIP) formulations of the above optimization problem and experimentally showed that this approach is far more scalable than traditional approaches, e.g., it can solve instances with 120 agents/rules in less than 20 seconds.

In this paper, we extend the formalization of CSG in [10] to handle negative value rules. We concentrate on MC-net-based representations since this representation scheme is more compact and natural than other representation schemes. Also, it can be easily extended to handle externalities among coalitions, i.e., a coalition can affect the performance of other coalitions. This extended representation scheme is called *embedded MC-nets* [8]. Although a rule may have a negative value in the original definition of [7], rule values are restricted to be positive to make the optimization problem simpler in [10].

Although any characteristic function can be represented without using negative value rules (as long as no coalition has a negative value), this restriction can make the representation size of a problem significantly larger. Also, allowing negative value rules can reduce the efforts for describing a characteristic function. Assume the president of a company is trying to reorganize the grouping of workers to maximize the productivity of the company. To utilize CSG techniques, the president needs to represent her knowledge about the characteristic function. When using MC-nets, we can assume most of default situations can be concisely represented by using positive value rules only. Negative value rules would be useful for describing some exceptional situations. Furthermore, to represent externalities among coalitions, the externalities can be either positive or negative.

However, handling negative value rules is a challenging task. If we simply add negative value rules, the MIP formulation in [10] cannot properly find an optimal coalition structure. When all rules have positive values, choosing a rule never hurts. Thus, in the MIP formulation, we construct a solver so that it tries to choose as many rules as possible. The constraints only specify the conditions where rules cannot be selected at the same time. Thus, if we simply include a negative value rule, the solver just ignores this rule, since choosing it *hurts*. We must describe the condition where the solver is forced to choose this negative value rule as a result of choosing other positive rules. Such a condition involves interaction among multiple rules, which is difficult to handle efficiently.

In this paper, we develop following three alternative algorithms to handle negative value rules: (i) a full transformation algorithm, which transforms all negative value rules into positive value rules, (ii) a partial transformation algorithm, which transforms all negative value rules into positive value rules and some negative rules that have a special form, and (iii) a direct encoding algorithm. which creates a set of *dummy rules* so that negative value rules are handled appropriately.

We show that the full transformation algorithm is not scalable in MC-nets, i.e., there exists an instance where the number of newly generated rules becomes $\Omega(n^2)$. Here, n is the number of agents. Furthermore, we show that when this transformation algorithm is applied to embedded MC-nets, there exists an instance where the number of newly generated rules becomes $\Omega(2^n)$. Although we have not yet proved that this exponential blowup is really inevitable, the current results are very negative. Since the CSG algorithm presented in [10] is exponential in the number of rules, even the increase of $\Theta(n^2)$ can be prohibitive.

In contrast, by using the partial transformation or direct encoding algorithms, an exponential blow-up never occurs even for embedded MC-nets. For embedded MC-nets, the direct encoding algorithm creates less rules than the partial transformation algorithm.

We experimentally compare the full/partial transformation algorithms and the direct encoding algorithm, and show that the direct encoding algorithm is by far superior. Also, this algorithm is scalable, i.e., an off-the-shelf optimization package (CPLEX) can solve problem instances with 100 agents/rules within 10 seconds.

2. MODELS/EXISTING WORKS

2.1 Characteristic Function Game

Let A be the set of agents, where $|A| = n$. We assume a characteristic function game, i.e., the value of coalition S is given by a characteristic function $v : 2^A \to \mathbb{R}$. Without loss of generality, we assume $\forall S \subseteq A$, $v(S) \geq 0$ holds. As shown in [15], even if some coalition's values are negative, as long as each coalition's value is bounded (i.e., not infinitely negative), we can normalize the coalition values so that all values are non-negative. This rescaled game is strategically equivalent to the original game. To save space, where there is no risk of confusion, we omit commas when listing sets, for example, writing $\{ab\}$ as a shorthand for $\{a, b\}$.

Coalition Structure Generation (CSG) involves partitioning a set of agents into coalitions so that social surplus is maximized. A coalition structure $CS = \{S_1, S_2, \ldots\}$ is a partition of A and is divided into disjoint and exhaustive coalitions, i.e., $CS = \{S_1, S_2, \ldots\}$ satisfies the following conditions:

$$\forall i, j (i \neq j), S_i \cap S_j = \emptyset, \bigcup_{S_i \in CS} S_i = A.$$

We denote by $\Pi(A)$ the space of all coalition structures over A. The value of coalition structure CS, denoted as $V(CS)$, is given by $V(CS) = \sum_{S_i \in CS} v(S_i)$.

An optimal coalition structure CS^* is a coalition structure that satisfies the following condition: $\forall CS \in \Pi(A), V(CS^*) \geq V(CS)$.

Definition 1 (MC-nets). *An* MC-net *consists of set of* rules R. *Each rule $r \in R$ is of the form: $(L_r) \to v_r$, where L_r is a condition of this rule, which is the conjunctions of literals over A, i.e., $a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m$. We call $\{a_1, \ldots, a_k\}$ positive literals and $\{a_{k+1}, \ldots, a_m\}$ negative literals. We say rule r is* applicable *to coalition S if L_r is true when the values of all Boolean variables that correspond to the agents in S are set to true, and the values of all Boolean variables that correspond to agents in $A \setminus S$ are set to false, i.e., $\bigwedge_{a \in S} a \wedge \bigwedge_{b \in A \setminus S} \neg b \models L_r$ holds. Without loss of generality, we assume each rule has at least one positive literal*[1].

In MC-nets, the condition of a rule must be the conjunctions of the literals. We say such a rule is *basic*. Also, we call a rule that has more complicated condition as *non-basic* rule. A non-basic rule must be transformed into multiple basic rules, whose conditions are disjointed with each other. For example, a non-basic rule that has form $(a \vee b \vee c) \to v$, is transformed into three basic rules, i.e., $(a) \to v$, $(\neg a \wedge b) \to v$, and $(\neg a \wedge \neg b \wedge c) \to v$.

Ohta *et al.* [10] identified a relation between two rules that can be classified into four non-overlapping and exhaustive cases. Based on this classification, they identified the condition of a set of rules as consistent, i.e., there exists at least one coalition structure CS such that each rule is applicable to a coalition in CS. Furthermore, they develop mixed integer programming (MIP) formulations for finding a consistent rule set that maximizes the sum of the rule values.

[1]For example, if a rule has a form $\neg a_1 \to 1$ and there exist agents $a_1, a_2, \ldots, a_n$, we can create equivalent rules as follows: $\neg a_1 \wedge a_2 \to 1$, $\neg a_1 \wedge \neg a_2 \wedge a_3 \to 1$, $\ldots$, $\neg a_1 \wedge \neg a_2 \ldots \wedge \neg a_{n-1} \wedge a_n \to 1$.

2.2 Partition Function Game

When there exist externalities among coalitions, the value of a coalition depends on the coalition structure in which the coalition belongs. An *embedded coalition* is a pair (S, CS), where $S \in CS \in \Pi(A)$. Let us denote the set of all embedded coalitions as M, i.e., $M := \{(S, CS) : CS \in \Pi(A), S \in CS\}$. A partition function is a mapping $w : M \to \mathbb{R}$.

Michalak *et al.* [8] proposed a concise representation of a partition function called *embedded MC-nets*.

Definition 2 (Embedded MC-nets). *An* embedded MC-nets *consists of set of embedded rules ER. Each embedded rule $er \in ER$ is of the form: $(L_1)|(L_2), \ldots, (L_l) \to v_{er}$, where each $L_1, L_2, \ldots, L_l$ is the conjunctions of literals over A. L_1, which we call the internal condition, is the condition that must be satisfied in the coalition that receives the value. $L_2, \ldots, L_l$, which we call external conditions, must be satisfied in other coalitions. We say that an embedded rule er is* applicable *to coalition S in CS if L_1 is applicable to S and each of $L_2, \ldots, L_l$ is applicable to some coalition $S' \in CS \setminus \{S\}$. For coalition S, $w(S, CS)$ is given as $\sum_{er \in ER_{(S,CS)}} v_{er}$, where $ER_{(S,CS)}$ is the set of embedded rules applicable to S in CS.*

Note that for an embedded rule, there exists an implicit constraint such that external conditions must be satisfied in coalitions $CS \setminus \{S\}$. By adding each positive literal in internal condition L_1 to the negative literals of each external conditions $L_2, \ldots, L_l$, as well as by adding each positive literal in external conditions $L_2, \ldots, L_l$ to the negative literals of internal condition L_1, we can explicitly represent this implicit constraint. We say an embedded rule is in an *explicit form* if the above condition is satisfied. For example, if an original rule is $(a)|(b), (c) \to v$, its explicit form is $(a \wedge \neg b \wedge \neg c)|(b \wedge \neg a), (c \wedge \neg a) \to v$. For simplicity, in the rest of this paper, we assume each embedded rule is in an explicit form.

Example 1. *Let us assume the following rules. Here, er_1 is an embedded rule.*

$r_1 : (a) \to 1$, $r_2 : (b) \to 1$,
$r_3 : (c) \to 1$, $r_4 : (d \wedge \neg a \wedge \neg b) \to 3$,
$r_5 : (a \wedge b) \to 1$, $er_1 : (d \wedge \neg a \wedge \neg b)|(a \wedge b \wedge \neg d) \to -2$.

If $CS = \{\{ab\}, \{c\}, \{d\}\}$, all rules are applicable. Thus, the $V(CS) = 1 + 1 + 1 + 3 + 1 - 2 = 5$.

The representation of embedded MC-nets is fully expressive and at least as concise as the conventional partition function game representation. As far as the authors are aware, the problem of finding an optimal coalition structure when a game is represented by embedded MC-nets has not yet been investigated. Extending the MIP formulation in [10] to handle embedded MC-nets is rather straightforward, as long as the rule has a non-negative value. More specifically, for an embedded rule that has a form $er : (L_1)|(L_2), \ldots, \to v_{er}$, we create basic rules $r_1 : (L_1) \to 0$, $r_2 : (L_2) \to 0$, $\ldots$, $r_l : (L_l) \to 0$. Assume $x_{er}, x_{r_1}, \ldots, x_{r_l}$ are 0/1 decision variables in the MIP formulation, i.e., when the value is 1, the rule is selected. An objective function is given by $\sum_{er} v_{er} \cdot x_{er}$. Also, we add a constraint that x_{er} can be 1 only when all of $x_{r_1}, \ldots, x_{r_l}$ are 1. Note that such a constraint is not linear. However, there exists a well-known encoding trick to represent such a non-linear constraint in MIP formulations [3].

3. CSG WITH NEGATIVE VALUE RULES

Although any characteristic function can be represented without using negative value rules, this restriction can make the representation size of a problem exponentially large. For example, let us assume for set of agents $A = \{a_1, a_2, \ldots, a_n\}$, $v(S) = |S|$ if $S \neq A$ and $v(A) = 0$. If we naively represent this characteristic function without negative value rules, we need $2^n - 1$ rules, where each rule is applicable exactly to one coalition[2]. If we can use a negative value rule, it suffices to have $n + 1$ rules, i.e., for each $a_i \in A$, $(a_i) \to 1$, and one negative value rule $(a_1 \wedge a_2 \wedge \ldots \wedge a_n) \to -n$.

However, handling negative value rules is a challenging task. In general, a negative reward in a reward maximization problem, or a negative cost in a cost minimization problem, is considered as a nuisance. When all rules have positive values, choosing a rule never hurts. Thus, in the MIP formulation, we construct a solver so that it tries to choose as many rules as possible. The constraints only specify the conditions where rules cannot be selected at the same time. Thus, if we simply include a negative value rule, the solver just ignores this rule, since choosing it *hurts*. We must describe the condition where the solver is forced to choose this negative value rule as a result of choosing other positive rules. Such a condition involves interaction among multiple rules, which is difficult to handle efficiently.

4. FULL TRANSFORMATION

Since handling negative value rules is difficult for the MIP formulation in [10], we consider transforming a rule set, which contains both positive/negative value rules into a rule set that contains positive value rules only.

4.1 MC-nets

We first show a full transformation algorithm for MC-nets. We assume that R is divided into two groups, i.e., a set of positive value rules R_+ and a set of negative value rules R_-.

Definition 3 (Full transformation algorithm). *The full transformation algorithm is defined as follows.*

1. *Set $R'_- = R_-$, $R'_+ = R_+$.*
2. *If $R'_- = \emptyset$, return R'_+.*
3. *Remove one rule $r_x : (L_x) \to -v_x$ from R'_-.*
4. *Remove one rule $r_i : (L_i) \to v_i$ from R'_+, such that $L_x \wedge L_i \not\models \bot$. If no such rule exists, return failure.*
5. *If $\neg L_x \wedge L_i \not\models \bot$, create a set of basic rules that is the transformation of non-basic rule $(\neg L_x \wedge L_i) \to v_i$. Add them to R'_+.*
6. *Create new basic rule $(L_x \wedge L_i) \to v_i - v_x$. If $v_i - v_x > 0$, add this rule to R'_+. If $v_i - v_x < 0$, add it to R'_-.*
7. *If $L_x \wedge \neg L_i \not\models \bot$, create a set of basic rules that is the transformation of non-basic rule $(L_x \wedge \neg L_i) \to -v_x$. Add them to R'_-. Goto 2.*

Let us explain the basic ideas of this algorithm. Since we assume that $\forall S, v(S) \geq 0$ holds, if negative value rule $r_x : (L_x) \to -v_x$ is applicable to coalition S, there exists at least one positive value rule $r_i : (L_i) \to v_i$, which is also applicable to S. In other words, r_i can partially eliminate the effect of r_x. We transform r_x and r_i into the following three rules:

[2]If we use a clever encoding trick, we require $O(n^2)$ rules.

r'_1: $(\neg L_x \wedge L_i) \rightarrow v_i$, which is added in Step 5,

r'_2: $(L_x \wedge L_i) \rightarrow v_i - v_x$, which is added in Step 6, and

r'_3: $(L_x \wedge \neg L_i) \rightarrow -v_x$, which is added in Step 7.

It is obvious that the original two rules, r_x and r_i, and these three rules are equivalent. Since r'_1 and r'_3 are non-basic, they must be transformed into multiple basic rules.

We can guarantee that the full transformation algorithm terminates, i.e., the following theorem holds.

Theorem 1. *The full transformation algorithm terminates.*

Proof. By one iteration of this algorithm, the negative value rule r_x is eliminated if $L_x \wedge \neg L_i \models \bot$ and $v_i \geq v_x$. If $L_x \wedge \neg L_i \not\models \bot$, a set of negative value rules are added in Step 7, but the conditions of these rules, i.e., $L_x \wedge \neg L_i$, are more specific than L_x. Also, if $v_i < v_x$, a new negative value rule is added in Step 6, but the condition of this rule, i.e., $L_x \wedge L_i$ is more specific than L_x and also disjoint with $L_x \wedge \neg L_i$. Furthermore, the value of this rule, i.e., $v_i - v_x$ is closer to 0 than the original value $-v_x$. Thus, by one iteration of this algorithm, the conditions of negative value rules become more specific and/or the negative value becomes closer to 0. Therefore, this algorithm cannot iterate infinitely and will terminate eventually. $\square$

Example 2. *Let us describe the full transformation algorithm, assuming $r_x : (L_x) \rightarrow -1$, where $L_x = a \wedge d \wedge e$, and $r_1 : (L_1) \rightarrow 1$, where $L_1 = a \wedge \neg b \wedge \neg c$, are selected. Since $L_1 \wedge \neg L_x = (a \wedge \neg b \wedge \neg c) \wedge (\neg a \vee \neg d \vee \neg e) \not\models \bot$ holds, we create non-basic rule $(L_1 \wedge \neg L_x) \rightarrow 1$ in Step 5. Then, we obtain two basic rules from this rule: $(a \wedge \neg b \wedge \neg c \wedge \neg d) \rightarrow 1$ and $(a \wedge \neg b \wedge \neg c \wedge d \wedge \neg e) \rightarrow 1$. We do not create any new rule in Step 6 since $v_{r_1} + v_{r_x} = 1 - 1 = 0$. Finally, since $\neg L_1 \wedge L_x = (\neg a \vee b \vee c) \wedge (a \wedge d \wedge e) \not\models \bot$ holds, we create non-basic rule $(\neg L_1 \wedge L_x) \rightarrow -1$. Then, we obtain two basic rules from this rule: $(a \wedge b \wedge d \wedge e) \rightarrow -1$ and $(a \wedge \neg b \wedge c \wedge d \wedge e) \rightarrow -1$.*

By using this algorithm, we can eliminate all negative value rules. However, this approach is not scalable. There exists an instance where the number of newly generated rules becomes $\Omega(n^2)$ by using the full transformation algorithm.

Example 3. *Let us consider the following rules.*

r_0: $(p_0 \wedge \neg n_1 \wedge \neg n_2 \wedge \ldots \wedge \neg n_k) \rightarrow 1$

r_1: $(p_1 \wedge n_1) \rightarrow 1$

r_2: $(p_2 \wedge n_2) \rightarrow 1$

$\ldots$

r_k: $(p_k \wedge n_k) \rightarrow 1$

r_x: $(p_0 \wedge p_1 \wedge p_2 \wedge \ldots \wedge p_k) \rightarrow -1$

This rule set contains $k + 1$ positive value rules and one negative value rule, where the total number of agents is $2k+1$. Figure 1 shows the number of newly generated rules from this rule sets by varying k. We can see that the number of newly generated rules becomes $\Omega(k^2)$, which is also $\Omega(n^2)$.

Then, can we reduce the number of required rules by using more clever encoding trick? Actually, the answer is no, i.e., the following theorem holds.

Theorem 2. *To represent the characteristic function in Example 3 by using positive value rules only, we need $\Omega(n^2)$ rules.*

Proof. For all $1 \leq i < j \leq k$, we denote $\{p_0, p_1, \ldots, p_k, n_i, n_j\}$ as $S_{i,j}$. For $S_{i,j}$, only rules r_x, r_i, r_j are applicable, thus $v(S_{i,j})$ is equal to 1. Assume that a set of positive value rules R'_+ represents v. There must be at least one rule in R'_+ that is applicable to $S_{i,j}$. Let us represent such a rule as $r_{i,j}$.

Now, we show that $r_{i,j}$ is not applicable to any $S_{i',j'}$, where $1 \leq i' < j' \leq k$ and $i \neq i' \vee j \neq j'$. We derive a contradiction by assuming that $r_{i,j}$ is applicable to $S_{i',j'}$.

When $i = i'$ or $i = j'$, let us consider coalition $S = \{p_0, p_1, \ldots, p_k, n_i\}$. For S, only rules r_x, r_i are applicable, thus $v(S)$ is equal to 0. However, we show that $r_{i,j}$ is applicable to S, thus $v(S)$ cannot be 0. $r_{i,j}$ is not applicable to S, if (i) its positive literals include agent n_l, where $l \neq i$, or (ii) its negative literals include at least one of $\{p_0, p_1, \ldots, p_k, n_i\}$. For (i), if $l = j$, $r_{i,j}$ is not applicable to $S_{i',j'}$. Also, if $l \neq j$, $r_{i,j}$ is not applicable to $S_{i,j}$. For (ii), $r_{i,j}$ is not applicable to both of $S_{i,j}$ and $S_{i',j'}$. This contradicts the assumption that $r_{i,j}$ is applicable to both of $S_{i,j}$ and $S_{i',j'}$. We can use a similar argument for the cases where $j = i'$ or $j = j'$.

Then, let us consider the case that i, j, i', j' are different with each other. Let us consider coalition $S = \{p_0, p_1, \ldots, p_k\}$. For S, only rules r_x, r_0 are applicable, thus $v(S)$ is equal to 0. However, we show that $r_{i,j}$ is applicable to S, thus $v(S)$ cannot be 0. $r_{i,j}$ is not applicable to S, if (i) its positive literals include agent n_l, where $1 \leq l \leq k$, or (ii) its negative literals include at least one of $\{p_0, p_1, \ldots, p_k\}$. For (i), if $l = i$ or $l = j$, $r_{i,j}$ is not applicable to $S_{i',j'}$. If $l \neq i$ and $l \neq j$, $r_{i,j}$ is not applicable to $S_{i,j}$. For (ii), $r_{i,j}$ is not applicable to both of $S_{i,j}$ and $S_{i',j'}$. This contradicts the assumption that $r_{i,j}$ is applicable to both of $S_{i,j}$ and $S_{i',j'}$.

Thus, for each i, j, where $1 \leq i < j \leq k$, there must be distinct element $r_{i,j}$ in R'_+, and the number of elements in R'_+ must be at least $k(k-1)/2$, which is $\Omega(n^2)$. $\square$

It remains an open question whether there exists a characteristic function and MC-nets representation with negative value rules, such that representing this characteristic function by a MC-net without negative value rules requires exponentially more space compared to the original MC-net representation. Our current conjecture is that such a characteristic function is likely to exist in embedded MC-nets, but not in MC-nets, assuming the number of negative value rules is bounded. We will discuss this issue in the next subsection.

4.2 Embedded MC-nets

The full transformation algorithm presented in Section 4.1 can be easily extended to embedded MC-nets. We replace a condition such as L_i to the condition for embedded rule C_{er}, which is a pair of internal condition L_1 and external conditions $L_2, \ldots, L_l$.

One tricky point is how to create the negation of C_{er}. Recall that embedded rule er is applicable to coalition S in CS if L_1 is applicable to S and each of $L_2, \ldots, L_l$ is applicable to some coalition $S' \in CS \setminus \{S\}$. Thus, er is not applicable to coalition S in CS if (i) L_1 is not applicable to S, (ii) L_1 is applicable to S, but L_2 is not applicable to *any* coalition in $CS \setminus \{S\}$, (iii) L_1 is applicable to S and L_2 is applicable to some coalition $S' \in CS \setminus \{S\}$, but L_3 is not applicable to *any* coalition in $CS \setminus \{S\}$, and so on. Handling case (i) is easy. Let us examine how to handle case (ii). Assume $L_2 = p_1 \wedge p_2$. We must guarantee that for any coalition $S' \in CS \setminus \{S\}$, $\neg L_2 = \neg p_1 \vee \neg p_2$ holds. If S' does not contain p_1, $\neg L_2$ holds. If S' contains p_1, then S' must satisfy $\neg p_2$. Since there exists exactly one coalition that contains p_1, it is sufficient to guarantee that there exists some coalition $S' \in CS \setminus \{S\}$, such that $p_1 \wedge \neg p_2$ holds.

To summarize, in order to represent $\neg C_{er}$, where C_{er} is a pair of the internal condition L_0 and external conditions $L_1, \ldots, L_l$, we need following conditions (here, we assume each $L_i = l_{i_1} \wedge l_{i_2} \wedge l_{i_3} \wedge \ldots$): (i) $(\neg L_0)$, (ii) $(L_0)|(l_{1_1} \wedge \neg l_{1_2})$, $(L_0)|(l_{1_1} \wedge l_{1_2} \wedge \neg l_{1_3})$, …, (iii) $(L_0)|(L_1)(l_{2_1} \wedge \neg l_{2_2})$, $(L_0)|(L_1)(l_{2_1} \wedge l_{2_2} \wedge \neg l_{2_3})$, and so on.

Example 4. *Let us consider the following rules:*

r_x: $(a \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_{k+1})$ |
$(\neg a \wedge p_1 \wedge \neg p_2 \wedge \neg p_3 \ldots \wedge \neg p_{k+1})$,
$(\neg a \wedge p_2 \wedge \neg p_1 \wedge \neg p_3 \ldots \wedge \neg p_{k+1}), \ldots,$
$(\neg a \wedge p_{k+1} \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_k) \rightarrow -1,$

r_1: $(a \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_{k+1})$ |
$(\neg a \wedge p_1 \wedge \neg p_2 \wedge \neg p_3 \ldots \wedge \neg p_{k+1} \wedge \neg h_1 \wedge \neg h_2 \ldots \wedge \neg h_k)$,
$(\neg a \wedge p_2 \wedge \neg p_1 \wedge \neg p_3 \ldots \wedge \neg p_{k+1})$,
$\ldots,$
$(\neg a \wedge p_{k+1} \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_k) \rightarrow 1,$

...

r_{k+1}: $(a \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_{k+1})$ |
$(\neg a \wedge p_1 \wedge \neg p_2 \wedge \neg p_3 \ldots \wedge \neg p_{k+1})$,
$(\neg a \wedge p_2 \wedge \neg p_1 \wedge \neg p_3 \ldots \wedge \neg p_{k+1})$,
$\ldots,$
$(\neg a \wedge p_{k+1} \wedge \neg p_1 \wedge \neg p_2 \ldots \wedge \neg p_k \wedge \neg h_1 \wedge \neg h_2 \ldots \wedge \neg h_k) \rightarrow 1$

This rule set contains $k+1$ positive value rules and one negative value rule. The total number of agents is $2k+2$. This rule set is constructed based on the well-known pigeon hole principle. There are $k+1$ pigeons $p_1, \ldots, p_{k+1}$ and k holes $h_1, \ldots, h_k$. r_x requires that all pigeons are in different coalitions. Also, each r_i is true if no hole is assigned to pigeon p_i. Thus, $\neg r_i$ means pigeon p_i has (at least) one hole. Then, as long as r_x is true, at least one rule $r_1, \ldots, r_k$ must be true. Otherwise, each pigeon has a different hole to stay, which is clearly impossible since there exist only k holes while there exist $k+1$ pigeons.

Figure 2 shows the number of newly generated rules (which includes embedded rules) by using our transformation algorithm. We can see that the number of newly generated rules becomes $\Omega(2^k)$, thus it is $\Omega(2^n)$. Although we have not yet proved that this exponential blowup is really inevitable, our current conjecture suggest that it is actually inevitable. Even if this is not the case, the current results are already very negative. Since the CSG algorithm presented in [10] is exponential in the number of rules, even the increase of $\Theta(n^2)$ can be problematic.

5. PARTIAL TRANSFORMATION

In this section, we develop an alternative transformation algorithm, which does not eliminate all negative value rules. More specifically, we are going to transform rules that contain negative value rules into positive value rules and some negative value rules that have a special form defined as follows.

Definition 4 (Singleton Rule). *We say rule $r : (L) \rightarrow v_r$ is singleton if L consists of exactly one positive literal a.*

It is clear that such a singleton rule r is applicable to any CS, since agent a must belong to one coalition. Also, for any other rule r', choosing r never prohibits choosing r'. Thus, if CS^* is optimal without r, then it is optimal with r. Thus, when solving the optimization problem, we can just ignore r. After finding optimal CS^*, we adjust $V(CS^*)$ by adding v_r.

Thus, if a negative value rule is singleton, it is actually harmless. Our new partial transformation algorithm transforms all negative value rules into positive value rules and negative value singleton rules.

Definition 5 (Partial transformation algorithm). *The partial transformation algorithm is defined as follows.*

1. *Set $R'_- = R_-$, $R'_+ = R_+, R_{\text{singleton}} = \emptyset$.*
2. *If $R'_- = \emptyset$, return R'_+ and $R_{\text{singleton}}$.*
3. *Remove one rule $r_x : (L_x) \rightarrow -v_x$ from R'_-.*
4. *Choose one positive literal $a \in L_x$. Add one rule $r_{\text{singleton}} : (a) \rightarrow -v_x$ to $R_{\text{singleton}}$.*
5. *Create a set of basic rules that are the transformation of non-basic rule $(\neg L_x \wedge a) \rightarrow v_x$. Add them to R'_+. Goto 2.*

Let us describe the procedure of this algorithm. In Step 4, we add one negative value singleton rule $r_{\text{singleton}} : (a) \rightarrow -v_x$. Conceptually, we also add one positive value singleton rule $r_+ : (a) \rightarrow v_x$. It is clear that adding these two rules does not change the values of a characteristic function since they negate with each other. Note that the procedure in Step 5 is equivalent to the full transformation algorithm in Steps 5 to 7, assuming that we choose r_+ as a positive value rule. No rule is added since $v_x - v_x = 0$ holds in Step 6, and $L_x \wedge \neg a \models \bot$ holds in Step 7. We iterate this procedure until all negative non-singleton value rules are eliminated. Then, we have a new rule set, which contains only positive value rules R'_+ and negative value singleton rules $R_{\text{singleton}}$.

As described above, we can solve CSG by using only R'_+ and obtain CS^*. The true value of $V(CS^*)$ is obtained by adding the values of the singleton rules in $R_{\text{singleton}}$.

Theorem 3. *The partial transformation algorithm terminates.*

Proof. By one iteration of this algorithm, one negative value rule is eliminated and no negative literal is added. Therefore, this algorithm terminates after $|R_-|$ iterations. □

Theorem 4. *In MC-nets, each negative value rule is transformed into one singleton negative value rule and $m-1$ positive value rules, where m is the maximal number of agents included in the rule.*

Proof. Let us assume r_x is the form $(L_x) \rightarrow -v_x$, where $L_x = (a_1 \wedge a_2 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_m)$. We first create non-basic rule $\neg L_x \wedge a_1 \rightarrow v_x$ in Step 5. Then, it is transformed into $m-1$ basic rules i.e., $(a_1 \wedge \neg a_2) \rightarrow v_x$, $(a_1 \wedge a_2 \wedge \neg a_3) \rightarrow v_x$, …, $(a_1 \wedge a_2 \wedge \cdots \wedge \neg a_k) \rightarrow v_x$, $(a_1 \wedge a_2 \wedge \cdots \wedge a_k \wedge a_{k+1}) \rightarrow v_x$, …, $(a_1 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \cdots \wedge \neg a_{m-1} \wedge a_m) \rightarrow v_x$. Thus, each negative value rule is transformed into one singleton negative value rule and $m-1$ positive value rules. □

THEOREM 5. *In embedded MC-nets, each negative value rule is transformed into one singleton negative value rule and $O(m \cdot l)$ positive value rules, where l is the maximal number of basic rules included in the embedded rule, and m is the maximal number of agents included in each basic rule.*

PROOF. Let us assume er_x has a form $(C_{er_x}) \to -v_x$, where $C_{er_x} = (L_0)|(L_1)\dots(L_{l-1})$. Also, each L_i has a form $(p_{i_1} \wedge p_{i_2} \wedge \dots \wedge p_{i_s} \wedge \neg n_{i_1} \wedge \neg n_{i_2} \wedge \dots \wedge \neg n_{i_t})$, where $s + t \le m$. We first create non-basic rule $\neg C_{er_x} \wedge p_{0_i} \to v_x$ in Step 5. In order to represent $\neg C_{er_x}$, we need following conditions: (i) $(\neg L_0)|\emptyset$, (ii) $(L_0)|(p_{1_1} \wedge \neg p_{1_2})$, $(L_0)|(p_{1_1} \wedge p_{1_2} \wedge \neg p_{1_3})$, $\dots$, $(L_0)|(p_{1_1} \wedge p_{1_2} \wedge \dots p_{1_s} \wedge n_{1_1})$, $\dots$, $(L_0)|(p_{1_1} \wedge p_{1_2} \wedge \dots p_{1_s} \wedge \neg n_{1_1} \wedge \dots n_{1_t})$, (iii) $(L_0)|(L_1)(p_{2_1} \wedge \neg p_{2_2})$, $(L_0)|(L_1)(p_{2_1} \wedge p_{2_2} \wedge \neg p_{2_3})$, $\dots$, $(L_0)|(L_1)(p_{2_1} \wedge p_{2_2} \wedge \dots p_{2_s} \wedge n_{2_1})$, $\dots$, $(L_0)|(L_1)(p_{2_1} \wedge p_{2_2} \wedge \dots p_{2_s} \wedge \neg n_{2_1} \wedge \dots n_{2_t})$, (iv) $(L_0)|(L_1)(L_2)(p_{3_1} \wedge \neg p_{3_2})$, $\dots$, and so on. Thus, we create at most m basic rules from each part of $\neg C_{er_x}$. Since there exist l parts, each negative value rule is transformed into one singleton negative value rule and $O(m \cdot l)$ positive value embedded rules. □

EXAMPLE 5. *Let us consider a negative value embedded rule that has the following form: $r_x : (L_1)|(L_2), (L_3), \dots, (L_n) \to -1$, where each $L_i = a_i \wedge \bigwedge_{j \ne i} \neg a_j$.*

In other words, this rule means each agent creates its own coalition. This rule contains n basic rules, and each basic rule contains n agents. Thus, this rule gives the worst-case where the number of rules added in the partial transformation algorithm is maximized, i.e., it becomes $\Theta(n^2)$. To make matters worse, most of these rules are embedded rules. Each embedded rule contains at most n basic rules. Thus, in this worst case, the total increase of the basic rules becomes $\Theta(n^3)$. In general, the partial transformation algorithm creates $O(m \cdot l^2)$ basic rules.

6. DIRECT ENCODING

In this section, we develop another approach for handling negative value rules. Instead of transforming a negative value rule into positive value rules, we add several dummy rules, whose rule values are zero. Each dummy rule describes some situations where the negative value rule is inapplicable. Furthermore, we add a constraint for the optimization algorithm so that the negative value rule must be selected if all of the dummy rules are not selected.

DEFINITION 6 (DUMMY RULES (FOR BASIC RULE)). *Assume there exists a negative value rule $r_x : (L_x) \to -v_x$ $(v_x > 0)$, where $L_x = a_1 \wedge a_2 \wedge \dots \wedge a_k \wedge \neg a_{k+1} \wedge \neg a_{k+2} \wedge \dots \wedge \neg a_m$. Dummy rules generated by this negative value rule are of the following two types:*

(i) $(a_1 \wedge \neg a_i) \to 0$, where $2 \le i \le k$,

(ii) $(a_1 \wedge a_j) \to 0$, where $k + 1 \le j \le m$.

We denote $D(L_x)$ as a set of dummy rules created from L_x.

It is obvious that the following theorem holds.

THEOREM 6. *For each negative value rule, $m - 1$ dummy rules are created, where m is the maximal number of agents included in the rule.*

Also, the following theorem holds.

THEOREM 7. *A negative value rule is applicable to a coalition in coalition structure CS if and only if all of its dummy rules are not applicable to any coalition in CS.*

PROOF. The condition of a dummy rule can be either $a_1 \wedge \neg a_i$ or $a_1 \wedge a_j$. In either case, it is clear that when this dummy rule is applicable to one coalition in CS, the negative value rule is not applicable to any coalition in CS. Also, if all dummy rules are inapplicable to any coalition in CS, it means that $a_1, a_2, \dots, a_k$ are in identical coalition S, while $a_{k+1}, a_{k+2}, \dots, a_m$ are not in S. Thus, the negative value rule is applicable to S. □

Assume $x_{r_x}, x_{d_1}, \dots$ are 0/1 decision variables for negative value rule r_x and its dummy rules $D(L_x)$ in the MIP formulation. We add a constraint that at least one of $x_{r_x}, x_{d_1}, \dots$ must be 1, i.e., $x_{r_x} + x_{d_1} + \dots \ge 1$ holds.

Let us show an example of dummy rules. We create the following dummy rules $D(L_x)$ for negative value rule r_x presented in Example 3: $(p_0 \wedge \neg p_1) \to 0$, $(p_0 \wedge \neg p_2) \to 0$, $\dots$, $(p_0 \wedge \neg p_k) \to 0$. These dummy rules $D(L_x)$ are quite similar to the rules added in the partial transformation algorithm, which we denote $R(L_x) = \{(p_0 \wedge \neg p_1) \to 1, (p_0 \wedge p_1 \wedge \neg p_2) \to 1, \dots, (p_0 \wedge p_1 \wedge \dots \wedge p_{k-1} \wedge \neg p_k) \to 1\}$. Actually, both algorithms add the same number of rules. However, dummy rules are much simpler. This is because the rules in $R(L_x)$ must be disjoint with each other, while the dummy rules can overlap (since their rule values are zero).

DEFINITION 7 (DUMMY RULES (FOR EMBEDDED RULE)). *Assume there exists a negative value embedded rule $r_x : (L_1)|(L_2), \dots, (L_l) \to -v_x$ $(v_x > 0)$. Then, dummy rules for r_x is $\bigcup_{L_i} D(L_i)$.*

It is obvious that the following theorem holds.

THEOREM 8. *For each negative value embedded rule, one singleton negative value rule and $(m-1) \cdot l$ dummy rules are created.*

Also, the following theorem holds.

THEOREM 9. *A negative value embedded rule is applicable to a coalition with coalition structure CS if and only if all of its dummy rules are not applicable to any coalition in CS*

We omit the proof since it is basically identical to Theorem 7.

For example, we create the following dummy rules for negative value embedded rule r_x presented in Example 5: $(a_1 \wedge a_2) \to 0$, $\dots$, $(a_1 \wedge a_n) \to 0$, $(a_2 \wedge a_3) \to 0$, $\dots$, $(a_2 \wedge a_n) \to 0$, $\dots$, $(a_{n-1} \wedge a_n) \to 0$.

The number of these dummy rules is about the same as the number of rules added in the partial transformation algorithm. However, each dummy rule is a simple basic rule. Thus, in the worst case, the total increase of basic rules in the direct encoding approach is $\Theta(n^2)$, while it is $\Theta(n^3)$ in the partial transformation algorithm.

7. EVALUATIONS

We experimentally evaluated the performance of proposed methods. All of the tests were run on a Xeon E5540 processor with 12-GB RAM. The test machine runs Windows Vista Business x64 Edition SP2. We used CPLEX 12.1, a general-purpose mixed integer programming package.

We show the performance of our proposed algorithms with randomly generated instances for the following two cases:

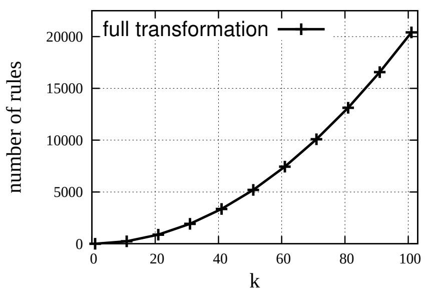

Figure 1: # of Generated Rules (Example 3)

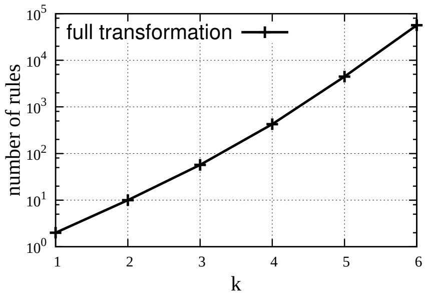

Figure 2: # of Generated Rules (Example 4)

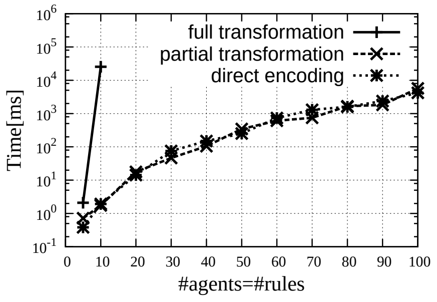

Figure 3: Computation Time: Case (i)

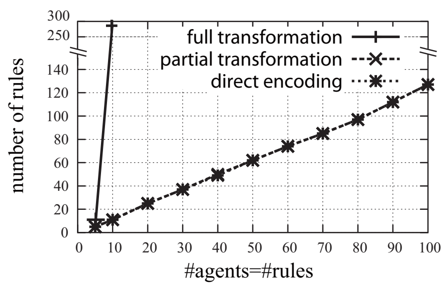

Figure 4: # of Generated Rules: Case (i)

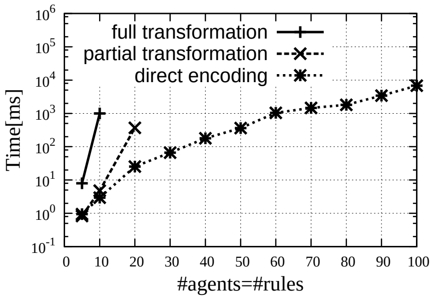

Figure 5: Computation Time: Case (ii)

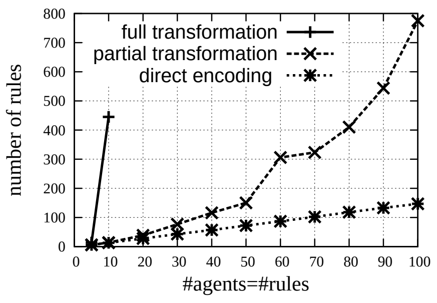

Figure 6: # of Generated Rules: Case (ii)

(i) rules can be either positive/negative and non-embedded, and (ii) rules can be either positive/negative and either non-embedded/embedded. For each case, problem instances are generated in the following method.

For case (i), we use a decay distribution [14] described as follows. Create a coalition with one random agent. Then repeatedly add a new random agent with probability α until an agent is not added or the coalition includes all agents. We use $\alpha = 0.55$. Choose value $v(S)$ between 0 and the number of agents in S uniformly at random. Then we create rule $(\bigwedge_{a \in S} a) \rightarrow v(S)$. Furthermore, we modify each rule by randomly moving an agent from the positive to negative literals with probability p. We use $p = 0.2$. Also, we convert the value to be negative with probability q. This method is basically identical to that used in [10].

For case (ii), we first create rule $(L_1) \rightarrow v_{er}$ in the same way as described in (i). Then, we repeatedly add a new condition of a rule with probability β until a rule is not added. We use $\beta = 0.15$. The value of the generated embedded rule is chosen between 0 and the number of agents in the rule, uniformly at random. Then we convert it to be negative with probability $q = 0.2$.

For all generated problem instances, we make sure that no coalition has a negative value. More specifically, we add positive value rules, if some coalition has a negative value.

We limit the time for problem transformation to five minutes, i.e., if an algorithm fails to transform a problem instance within five minutes, we terminate the transformation and do not examine the computation time for such an instance. Such an early termination occurs frequently in the full transformation approach, but it never happens in the partial transformation and direct encoding algorithms.

We set #rules = #agents and vary #rules from 5 to 100. We generate 50 problem instances for all cases and #rules. We investigate the computation time for these three algorithms and the number of newly generated rules. The results are illustrated in Figures 3, 4, 5, and 6. Each data point in these Figures is the median of 50 data points.

The results of case (i) are illustrated in Figures 3 and 4. We can see that the CSG problem becomes more difficult when #rules (which is equal to #agents) increases in Figure 3. This is natural since #rules increases, the number of decision variables in the corresponding MIP formulation.

It is clear that the full transformation algorithm is inefficient compared with the other two algorithms. It cannot transform problem instances in five minutes even when #rules = 15. Also, we were able to solve only 45% of the problem instances; the remaining 55% problem instances cause error due to insufficient memory. On the other hand, the partial transformation and direct encoding algorithms are more efficient than the full transformation algorithm. The required time for these algorithms is less than 10 seconds. We can see that the number of the newly generated rules of these two algorithms is almost the same in Figure 4.

The differences of these algorithms become more obvious in Figures 5 and 6. The full transformation algorithm cannot transform the problem instances in five minutes when #rules = 15. Also, we were able to solve only 45% of the problem instances; the remaining 55% problem instances cause error due to insufficient memory. The partial transformation algorithm can solve problem instances within 1.0 seconds when #rules = 20, but it fails to solve problem instances when #rules becomes more than 30 due to insufficient memory. On the other hand, the direct encoding algorithm can solve problem instances for all #rules and the required time is less than 10 seconds. Figure 6 shows the number of newly generated basic rules, i.e., a newly generated embedded rule is further decomposed into multiple basic rules. The partial transformation algorithm adds $O(m \cdot l)$ embedded rules, each of which contains at most l basic rules. Thus, the number of basic rules is $O(m \cdot l^2)$. On the other hand, the direct encoding algorithm adds at most $O(m \cdot l)$ basic rules. Consequently, the problem instances generated by the partial transformation algorithm becomes more difficult than those of the direct encoding algorithm.

To summarize, the direct encoding algorithm is the most scalable among these three algorithms; the required times for the solving problem instances in all cases are all less than 10 seconds. On the other hand, the IP$^{+/-}$ algorithm, which does not make use of compact representations, required around 160 minutes to solve instance with 20 agents [12].

8. CONCLUSIONS

In this paper, we extended the formalization of CSG in [10] so that it can handle negative value rules. Allowing negative value rules is important since (a) it can reduce the efforts for describing a characteristic function, (b) the representation size of a problem can be much more concise, and (c) in many realistic situations, it is natural to assume that a coalition has negative externalities to other coalitions.

Since the current CSG algorithm cannot handle negative value rules, we examine the following three methods: (i) a full transformation algorithm, which transforms all negative value rules into positive value rules, (ii) a partial transformation algorithm, which transforms all negative value rules into positive value rules and some negative rules that have a special form, and (iii) a direct encoding algorithm, which creates a set of *dummy rules* so that negative value rules are handled appropriately.

We show that the full transformation algorithm is not scalable in MC-nets since the worst-case representation size will be $\Omega(n^2)$ for MC-nets, and it can be $\Omega(2^n)$ for embedded MC-nets. On the other hand, by using the partial transformation or direct encoding algorithms, an exponential blow-up never occurs even for embedded MC-nets. We experimentally compared these algorithms and showed that the direct encoding algorithm is by far superior.

It still remains an open question whether a characteristic function exists that inevitably requires exponentially more space without using negative value rules. In our future works, we hope to confirm our current conjectures, i.e., such a characteristic function exists in embedded MC-nets (one promising candidate is the characteristic function presented in Example 4), but not in MC-nets.

9. ACKNOWLEDGMENTS

This work was supported by KAKENHI (20240015) and Grant-in-Aid for JSPS research fellows. The authors would like to thank Hirotaka Ono, Ryuzo Hasegawa, Hiroshi Fujita, Miyuki Koshimura and anonymous reviewers for their helpful comments.

10. REFERENCES

[1] H. Aziz and B. de Keijzer. Complexity of coalition structure generation. In *AAMAS*, pages 191–198, 2011.

[2] B. Banerjee and L. Kraemer. Coalition structure generation in multi-agent systems with mixed externalities. In *AAMAS*, pages 175–182, 2010.

[3] E. Castillo, A. J. Conejo, P. Pedregal, R. Garcia, and N. Alguacil. *Building and Solving Mathematical Programming Models in Engineering and Science.* Wiley-Interscience, 2001.

[4] V. Conitzer and T. Sandholm. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence*, 170(6):607–619, 2006.

[5] E. Elkind, L. A. Goldberg, P. W. Goldberg, and M. Wooldridge. A tractable and expressive class of marginal contribution nets and its applications. In *AAMAS*, pages 1007–1014, 2008.

[6] G. Greco, E. Malizia, L. Palopoli, and F. Scarcello. On the complexity of core, kernel, and bargaining set. *Artificail Intelligence*, 175(12-13):1877–1910, 2011.

[7] S. Ieong and Y. Shoham. Marginal contribution nets: a compact representation scheme for coalitional games. In *ACM EC*, pages 193–202, 2005.

[8] T. Michalak, D. Marciniak, M. Szamotulski, T. Rahwan, M. Wooldridge, P. McBurney, and N. R.Jennings. A logic-based representation for coalitional games with externalities. In *AAMAS*, pages 125–132, 2010.

[9] T. Michalak, J. Sroka, T. Rahwan, M. Wooldridge, P. McBurney, and N. R. Jennings. A distributed algorithm for anytime coalition structure generation. In *AAMAS*, pages 1007–1014, 2010.

[10] N. Ohta, V. Conitzer, R. Ichimura, Y. Sakurai, A. Iwasaki, and M. Yokoo. Coalition structure generation utilizing compact characteristic function representations. In *CP*, pages 623–638, 2009.

[11] T. Rahwan, T. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N. R. Jennings. Constrained coalition formation. In *AAAI*, 2011.

[12] T. Rahwan, T. Michalak, N. R. Jennings, M. Wooldridge, and P. McBurney. Coalition structure generation in multi-agent systems with positive and negative externalities. In *IJCAI*, pages 257–263, 2009.

[13] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research (JAIR)*, 34:521–567, 2009.

[14] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

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

[16] S. Ueda, M. Kitaki, A. Iwasaki, and M. Yokoo. Concise characteristic function representations in coalitional games based on agent types. In *IJCAI*, pages 393–399, 2011.

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