Monte-Carlo Tree Search for Scalable Coalition Formation

Feng Wu[1] and **Sarvapali D. Ramchurn**[2]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]School of Electronics and Computer Science, University of Southampton
wufeng02@ustc.edu.cn, sdr1@soton.ac.uk

Abstract

We propose a novel algorithm based on Monte-Carlo tree search for the problem of coalition structure generation (CSG). Specifically, we find the optimal solution by sampling the coalition structure graph and incrementally expanding a search tree, which represents the partial space that has been searched. We prove that our algorithm is complete and converges to the optimal given sufficient number of iterations. Moreover, it is anytime and can scale to large CSG problems with many agents. Experimental results on six common CSG benchmark problems and a disaster response domain confirm the advantages of our approach comparing to the state-of-the-art methods.

1 Introduction

Coalition formation is an important research topic in multi-agent systems where a group of agents come together to form coalitions in the performance of a specific task [Shehory and Kraus, 1998] or or provisioning of a service such as social ride-sharing [Bistaffa *et al.*, 2017]. One of the key challenges that arise in this formation process is that of *Coalition Structure Generation* (CSG), which involves partitioning the set of agents into exhaustive and disjoint coalitions, called a coalition structure, so as to maximise the social welfare. Unfortunately, finding the optimal coalition structure is difficult and has been proved to be NP-complete [Sandholm *et al.*, 1999].

To date, a number of approaches have been proposed to solve CSG problems either optimally or approximately. Given that the solution space is exponential with the number of agents, optimal methods [Yeh, 1986; Rahwan *et al.*, 2009] are usually not *scalable* and can only handle problems with small number of agents (e.g., <40 [Rahwan *et al.*, 2015]). This limits its applicability to real-world problems requiring many agents. On the other hand, approximate techniques [Di Mauro *et al.*, 2010; Farinelli *et al.*, 2013] can return a solution quickly but usually provide no guarantees on solution quality. This makes them unsatisfactory for some serious applications (e.g., disaster response). To amend this, approaches [Service and Adams, 2011; Farinelli *et al.*, 2017] were proposed to offer bounds for the solution quality. However, they require the utility to have certain special structures that may not be realistic for some applications.

Against this background, we propose CSG-UCT — a scalable and anytime approach for coalition formation in multi-agent systems. Notably, it is *scalable* and can solve large CSG problems with hundreds of agents, which is computationally intractable for existing optimal methods. Furthermore, it is *anytime* and can return the current best solution if time is limited, or converge to the optimal solution if time is sufficient. Specifically, we borrow ideas from the research on *Monte-Carlo Tree Search* (MCTS), which has been successful in solving large games (e.g., AlphaGo [Silver *et al.*, 2016]), to find the best solution in the large coalition structure graph [Sandholm *et al.*, 1999]. In more detail, we develop a variation of the UCT algorithm [Kocsis and Szepesvári, 2006] for CSG, which selects the most promising moves towards the optima using the UCB1 heuristic [Auer *et al.*, 2002], and expands the search tree based on random sampling of the solution space. To the best of our knowledge, this is the first work to solve CSG problems using the MCTS techniques.

In this paper, we advance the state-of-the-art with the following contributions: 1) In Section 3, we propose a novel method to search the coalition structure graph using the MCTS framework. 2) In Section 4, we prove the proposed algorithm is complete and anytime, and discuss its connections with other methods and some implementation tricks. 3) In Section 5, we empirically evaluate the proposed approach on six common benchmark problems and a disaster response domain, and confirm that it indeed converges to the optimal solution given sufficient number of iterations and outperforms the leading approximate methods for large problems. All together, we contribute with the first MCTS-based method for CSG, which is promising to improve the applicability of coalition formation techniques in real-world applications.

2 Background

2.1 Coalition Structure Generation

Given a set of n agents denoted by $A = \{a_1, a_2, \cdots, a_n\}$, a *coalition* C is defined as a non-empty subset of A, i.e., $C \neq \emptyset$ and $C \subseteq A$. For any coalition C, the *characteristic function* $v(C) \in \Re$ specifies a value that represents a cost or profit for that coalition. A *coalition structure* $CS = \{C_1, C_2, \cdots, C_k\}$ is a collection of k coalitions, such that $\bigcup_{i \in 1..k} C_i = A$ and

$C_i \cap C_j = \emptyset$ for any $i, j \in 1..k$ and $i \neq j$, i.e., each agent is selected at least in and only in one coalition. The value of a coalition structure CS is the sum of the coalition values in the structure, i.e., $v(CS) = \sum_{C \in CS} v(C)$. For example, given a set of four agents $A = \{a_1, a_2, a_3, a_4\}$, $C = \{a_2, a_3\}$ is a coalition of A and a possible coalition structure over A is $CS = \big\{\{a_1\}, \{a_2, a_3\}, \{a_4\}\big\}$, with the value computed as: $v(CS) = v(\{a_1\}) + v(\{a_2, a_3\}) + v(\{a_4\})$.

Coalition Structure Generation (CSG) is the problem of finding a coalition structure CS^* over A whose value is maximal, i.e., $CS^* = \arg\max_{CS \in \Pi^A} v(CS)$ where Π^A denotes the set of all possible coalition structures over A. Here, we assume that $v(C) \geq 0$ for any $C \subseteq A$. Generally, the CSG problem implies searching Π^A in order to find one whose value is maximal. Several CSG approaches are based on the observation that the set Π^A has some structure, which can be exploited to speed up the search process. A commonly used representation for Π^A is the *coalition structure graph* [Sandholm *et al.*, 1999], i.e., a undirected graph, where:

- A node represents a coalition structure and is categorized into levels, $\Pi^A_1, \Pi^A_2, \cdots, \Pi^A_n$, where level Π^A_i contains the nodes that represent all coalition structures with exactly i coalitions.
- An edge connects two coalition structures if and only if: they belong to two consecutive levels Π^A_i and Π^A_{i-1} and the coalition structure in Π^A_i can be obtained from the one in Π^A_{i-1} by splitting one coalition into two.

2.2 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree in an incremental and asymmetric manner. It has been proved to be useful in domains that can be represented as trees of sequential decisions, particularly games (e.g., Go [Silver *et al.*, 2016]) and planning problems (e.g., MDPs).

The basic MCTS process is conceptually very simple. To incrementally build a search tree, at each iteration, it starts from the root and consists of the following four steps: 1) *Selection*: it selects successive child nodes down to a leaf node, 2) *Expansion*: it expands the tree by adding a new child node, 3) *Simulation*: it runs several simulations from that node, 4) *Backpropagation*: it uses the result of the simulations to update information in the nodes on the path from the leaf to the root. This process repeats until some termination condition meets. One of the essential step is how to select a child node so that the tree expands towards the most promising subspace.

UCT [Kocsis and Szepesvári, 2006] is a MCTS-based algorithm that has achieved remarkable success in solving large MDPs. Specifically, it treats the node selection as a *multi-armed bandit* problem and applies the UCB1 heuristic [Auer *et al.*, 2002], where the node value is augmented by an exploration bonus that is highest for rarely tried children. More techniques on MCTS and its variances can be found in the recent survey [Browne *et al.*, 2012].

3 Applying MCTS to CSG

We propose the CSG-UCT approach by applying the MCTS method to search the optimal coalition structure in the CS graph. Specifically, we start the search from the top level of the CS graph down to the bottom level and expand the search tree based on random sampling of the solution space.

Similar to the aforementioned CS graph, we define the *search tree* built by our algorithm as follow:

- Each tree node is associated with a coalition structure $CS \in \Pi^A$ and the root node of the tree is with the singleton coalition structure, i.e., $\big\{\{a_1\}, \{a_2\}, \cdots, \{a_n\}\big\}$.
- Each tree branch is linked with a pair-wise join operation, where the child linked by that branch is generated by joining two coalitions of the CS in its parent node, e.g., $\big\{\{a_1, a_2\}, \{a_3\}, \{a_4\}\big\}$ is the child of node $\big\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\big\}$ by joining $\{a_1\}, \{a_2\}$.

Clearly, the maximal depth of the search tree is n, the size of CS in depth k of the tree is k, and each path from the root to the leaf will end with the grand coalition. In the rest of this paper, we will interchangeably use a coalition structure CS to refer the tree node associated with CS.

Starting from the tree with only the root node, we incrementally build the search tree iteration by iteration. A partial search tree represents the solution space that has been sampled (i.e., searched). Following the standard procedure of MCTS, we describe one iteration of our algorithm, given the current (partial) search tree, with the following four steps:

3.1 Selection

The selection step is for each tree node to select a branch in order to form a path from the root to a leaf. This is a critical step for MCTS because it implies which region of the solution space should be searched at the current iteration. We borrow ideas from the UCT algorithm and view the node selection as a *Multi-Armed Bandit* (MAB) problem. Specifically, we use the UCB1 heuristic [Auer *et al.*, 2002], which is a simple yet attractive method to solve MAB.

Let $V(CS) = \max_{CS' \in Tree(CS)} v(CS')$ be the maximal value of the coalition structures in the subtree $Tree(CS)$ rooted by CS. This value is known only when the subtree is complete. Let $\overline{Tree}(CS)$ be the partial subtree built the search process and $\bar{V}(CS) = \max_{CS' \in \overline{Tree}(CS)} v(CS')$ be the current maximal value. Let $N(CS)$ be the frequency count of the node being visited during the search process. The UCB1 heuristic for node selection can be written as:

$$UCB1(CS') = \bar{V}(CS') + c\sqrt{\frac{\log N(CS)}{N(CS')}} \qquad (1)$$

where c is a constant parameter. Given this, we can select the child node that maximizes the UCB1 heuristic as: $CS^* = \arg\max_{CS' \in Child(CS)} UCB1(CS')$, where $Child(CS)$ denotes all possible children of node CS.

Intuitively, if $N(CS')$ is equal for all children, the UCB1 will select the child with the maximal $\bar{V}(CS')$ because the remaining term is equal for all of them. However, if some child is much less frequently visited than others, the term

$\sqrt{\log N(CS)/N(CS')}$ will become significant and bias the selection towards it. For example, if $N(CS) = 100, N(CS') = 1, N(CS'') = 99, \sqrt{\log 100/1} \approx 1.4 > \sqrt{\log 100/99} \approx 0.14$. Therefore, the UCB1 may select CS' instead of CS'' though $\bar{V}(CS'')$ is slightly larger (e.g., less than 1.0). In a word, the UCB1 has the good property of balancing the well-known *exploration and exploitation* tradeoff [Auer *et al.*, 2002].

3.2 Expansion

When a child $CS^* \in Child(CS)$ chosen by the node selection procedure is currently not in the tree, the search tree is then expanded by adding a new leaf node CS^* as a child of CS to the tree.

3.3 Simulation

When a new leaf node CS^* is added to the tree, some default policy is used to estimate $\bar{V}(CS^*)$. In this work, we perform a rollout search by successively joining two coalitions. For example, a rollout trace starting from the singleton coalition structure $\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}$ is:

$$\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\} \rightarrow \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}$$
$$\rightarrow \{\{a_1, a_2\}, \{a_3, a_4\}\} \rightarrow \{\{a_1, a_2, a_3, a_4\}\}$$

Apparently, this process ends with the grand coalition when no join of two coalitions is possible. Then, the value of CS^* is initialized by $\bar{V}(CS^*) \leftarrow \max_{CS' \in Trace(CS^*)} v(CS')$, where $Trace(CS^*)$ is a set of coalition structures encountered during the rollout search starting from CS^*.

Now, the remaining problem is how to select the two coalitions C_1, C_2 to be joined in a coalition structure CS. Here, we use a simple myopic heuristic as follow:

$$C_1, C_2 = \arg \max_{C'_1, C'_2 \in CS} [v(CS') - v(CS)] \qquad (2)$$

where $CS' = CS + \{C'_1 \cup C'_2\} - \{C'_1\} - \{C'_2\}$ is the coalition structure after joining two coalitions C'_1 and C'_2 in CS. Intuitively, we join two coalitions in CS when it results in the maximal improvement in the coalition structure value.

3.4 Backpropagation

After the value $\bar{V}(CS^*)$ of the newly added node CS^* is initialized, the values of all its ancestors in the tree are updated by backpropagting the value $\bar{V}(CS^*)$ from the leaf node CS^* to the root. Specifically, for every level-l node CS_l in the propagation path of the tree, its value is updated by $\bar{V}(CS_l) = \max\{\bar{V}(CS_l), \bar{V}(CS_{l+1})\}$, where CS_{l+1} is the child of CS_l in the path.

To put the above four steps together, we have one iteration of the overall CSG-UCT algorithm. Starting from the singleton coalition structure, the iteration continues until it runs out of time or all nodes have be added to the tree, which implies the complete solution space Π^A has been searched. In the later case, the algorithm returns the optimal solution while in the former case it returns the currently best solution.

An example of the partial search tree built by CSG-UCT is shown in Figure 1. An iteration on the tree starts from the root node $\{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}$. In the *selection* step, the UCB1 heuristic first selects tree node $\{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}$

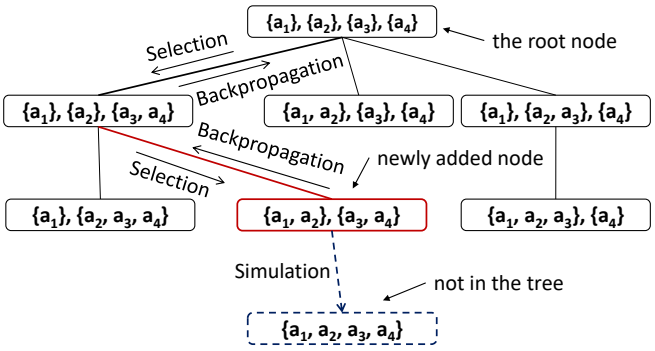


Figure 1: A partial search tree built by the CSG-UCT method.

and then recursively selects $\{\{a_1, a_2\}, \{a_3, a_4\}\}$. Since this (red) node is not currently in the tree, the tree is expanded by adding it as a leaf of the tree in the *expansion* step. Then in the *simulation* step, the value of the newly added node $\{\{a_1, a_2\}, \{a_3, a_4\}\}$ is initialized by a rollout search ending with the grand coalition $\{\{a_1, a_2, a_3, a_4\}\}$. Finally in the *backpropagation* step, the value of the newly added node is propagated back to the root to update the value of the currently best solution along the path. The search tree grows iteration by iteration until all $CS \in \Pi^A$ are added in the tree.

4 Analysis and Discussion

Theorem 1. *The CSG-UCT algorithm is complete given a sufficient amount of time.*

Proof (Sketch). Given a sufficient amount of time, CSG-UCT terminates when all nodes have be added to the tree. Assuming there exists a coalition structure CS' that cannot be selected by its parent CS in the tree. Therefore, CS' cannot be added to the tree by the iteration of CSG-UCT. This is contradicted with the fact that every arm in the MAB will eventually be pulled using the UCB1 heuristic [Auer *et al.*, 2002]. By induction, we conclude that all possible coalition structures in Π^A will eventually be added to the tree if time is sufficient. Thus, the CSG-UCT algorithm is complete. □

Theorem 2. *The CSG-UCT algorithm is anytime and always returns the currently best solution.*

Proof (Sketch). Note that the value of every searched node is propagated back to the root node at the end of each iteration. Therefore, the root node always maintains the value of the currently best solution. When the algorithm stops, it can return the currently best solution from the root node. After an iteration, $\forall_{CS} : \bar{V}^t(CS) \leq \bar{V}^{t+1}(CS) \leq V(CS)$ holds. With more iterations, more nodes will be searched and added to the tree. As a result, the solution is constantly improved. Thus, the CSG-UCT algorithm is anytime. □

Relation with the DP method. The DP for CSG [Yeh, 1986] can be viewed as evaluating every movement upward in the coalition structure graph by splitting one coalition into two. It then stores the best movements in a table and finally moves upwards in the graph, starting from the bottom node, as long as it is beneficial to do so. In CSG-UCT, we perform

best-first search downward in the coalition structure graph, starting from the top node, by joining two coalitions into one. We store the best movements in a tree structure and speed up the search by combining several heuristics. Similar to DP, CSG-UCT is also complete. In practice, it is often more efficient than DP due to the use of the heuristics.

Relation with the C-Link method. The C-Link [Farinelli *et al.*, 2017] starts from the top node of the coalition structure graph, iteratively computes the most suitable pair of coalitions, and move downward in the graph if joining a coalition pair has benefit to do so. In contrast to CSG-UCT, C-Link is *memory-less* and only keeps the currently best solution. It can be viewed as a local search method and can easily get stuck in local optimal. In fact, we also use similar local search technique in the simulation phase and improve it using a tree structure to memorize the intermediate results.

Implementation. There are several techniques that can make CSG-UCT more memory-efficient. Firstly, the tree nodes representing the same coalition structure can share a single node in the memory. By doing so, the search tree has at most the number of nodes as the coalition structure graph. Secondly, if a subtree rooted with some coalition structure is complete (i.e., with all branches ending with the leaf node of the grand coalition), it can be removed from the search tree in order to save some memory. There are also many well-established techniques for MCTS [Browne *et al.*, 2012] that can be applied to speed up the search process of CSG-UCT, such as parallel MCTS, MCTS on GPUs, etc.

5 Empirical Evaluation

We evaluated the performance of our algorithm on several CSG problems including the six benchmark problems commonly used in the literature and a disaster response domain that motivates our approach. We compared our algorithm with GRASP [Di Mauro *et al.*, 2010] and C-Link [Farinelli *et al.*, 2013], which are currently the leading approximate methods for the CSG problem. All the algorithms are implemented in Java and executed on a Intel(R) Core(TM) i7 CPU, 2.50GHz, with 8GB of memory.

In the experiments, we recorded the following two indicative measures of the solution quality. Firstly, we computed the *Optimal Ratio* $= v(CS)/v(CS^*)$ between the value of the computed solution CS and the optimal solution CS^*. This measure shows how far the value of the computed solution is from the optimal. Here, optimal solutions are computed using the integer programming formulation solved by CPLEX. We limit the computation of the optimal solutions up to 20 agents because this is the maximum number of agents that can be handled on our machine using CPLEX. Secondly, we recorded the *Baseline Gain* $= v(CS) - \sum_{i \in A} v(\{i\})$ between the computed solution CS and the singleton coalition structure when the optimal solution is unavailable as the number of agents are too large to be solved by CPLEX. This measure shows how valuable it is to form the computed coalition structure as opposed to the singletons (i.e., the baseline).

#Agents	4	5	6	7	8	9	10	11	12	13
#Iterations	1	5	1	7	11	38	5	33	68	14

Table 1: Maximal Iterations for Finding Optimal Solutions (NDCS)

5.1 Common Benchmark Problems

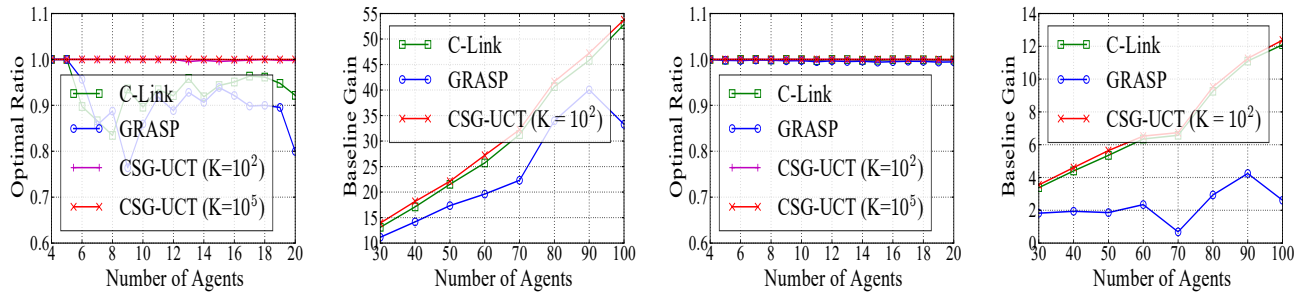
Note that the common practice in evaluating CSG algorithms is to choose some standard instances of the problem and compare the various algorithms that exist without giving them a priori knowledge of the type of utilities they are presented with. Although our algorithm can solve CSG problems with *any* types of utilities, we benchmark with six types of utilities with the following commonly used distributions:

- **Uniform**, as studied in [Larson and Sandholm, 2000]: $v(C) \sim U(a, b)$ where $a = 0$ and $b = |C|$.
- **Normal**, as studied in [Rahwan *et al.*, 2007]: $v(C) \sim N(\mu, \sigma^2)$ where $\mu = 10 \times |C|$ and $\sigma = 0.1$.
- **Modified Uniform**, as studied in [Service and Adams, 2010]: $v(C) \sim U(0, 10 \times |C|)$, and $v(C)$ is increased by a random number $r \sim U(0, 50)$ with probability 0.2.
- **Modified Normal**, as studied in [Rahwan *et al.*, 2012]: $v(C) \sim N(10 \times |C|, 0.1^2)$, and the team value $v(C)$ is increased by a random number $r \sim U(0, 50)$ with probability 0.2.
- **NDCS**, as studied in [Rahwan *et al.*, 2009]: $v(C) \sim \times N(\mu, \sigma^2)$ where $\mu = |C|$ and $\sigma = \sqrt{|C|}$.
- **Agent-based**, as studied in [Rahwan *et al.*, 2012]: $v(C) = \sum_{i \in C} p_i^C$ where $p_i^C \sim U(0, 2p_i)$ and $p_i \sim U(0, 10)$ is a random power for agent i.

Figures 2(a-f) summarize our experimental results on the common benchmark problems with the utilities using the distributions described above. We tested CSG-UCT with different size of iterations (i.e., $K = 10^2$ and $K = 10^5$). As we can see from the figures, CSG-UCT outperforms both GRASP and C-Link with better solution quality in all the tested instances. In particular, with more iterations (e.g., $K = 10^5$), CSG-UCT can find the near-optimal solutions for all the tested problems. This set of experiments shows the advantage of CSG-UCT comparing to the state-of-the-art approximate methods. As we can see, C-Link already performed quite well with the optimal ration of about 0.9 in all the six benchmark problems. Our algorithm can still make significant improvement comparing with C-Link. This confirms the effectiveness of our algorithm.

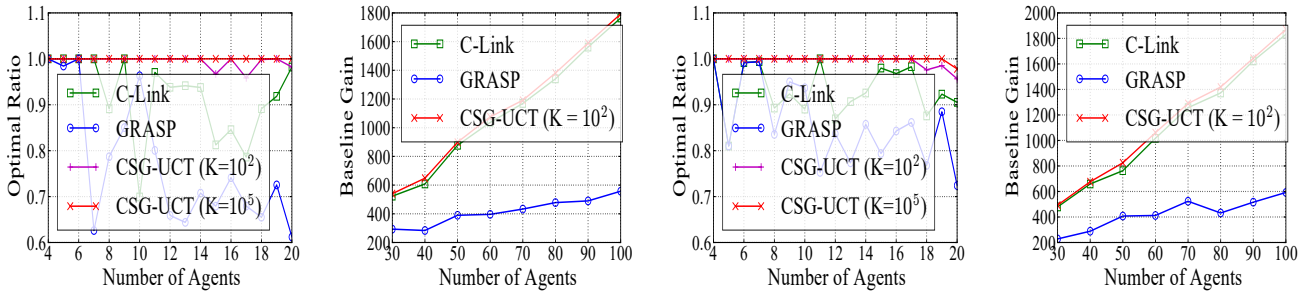
Table 1 illustrates the maximal number of iterations required for CSG-UCT to find the optimal solution on the NDCS benchmark problems with different numbers of agents. As shown in the table, CSG-UCT is very efficient and only requires few iterations (mostly less than 20) to find the optimal solution for the tested problem instance. It is worth noting that the number of searched coalition structures is usually much smaller than the overall solution space.

As aforementioned, CSG-UCT is *anytime* that can stop at anytime and return the currently best solution, and continue to improve the solution until it converges to the optimal if more time is available. To demonstrate this property, we conduct experiments on the NDCS benchmark problem with 25



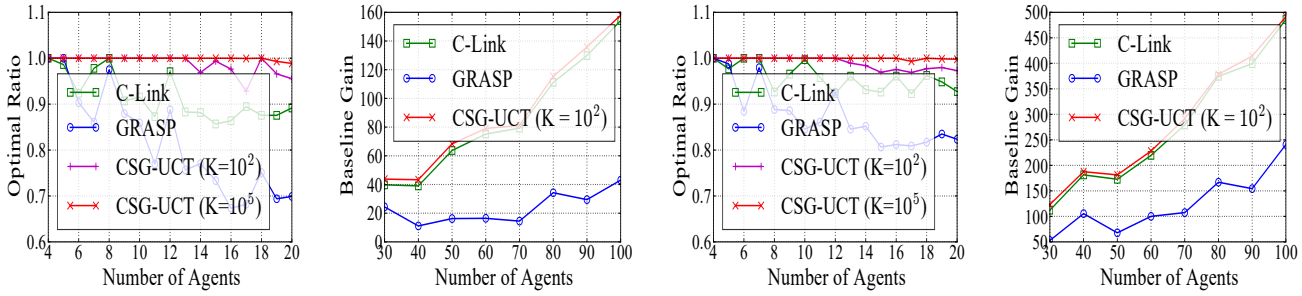
(a) Uniform (Optimal Ration and Baseline Gain)

(b) Normal (Optimal Ration and Baseline Gain)



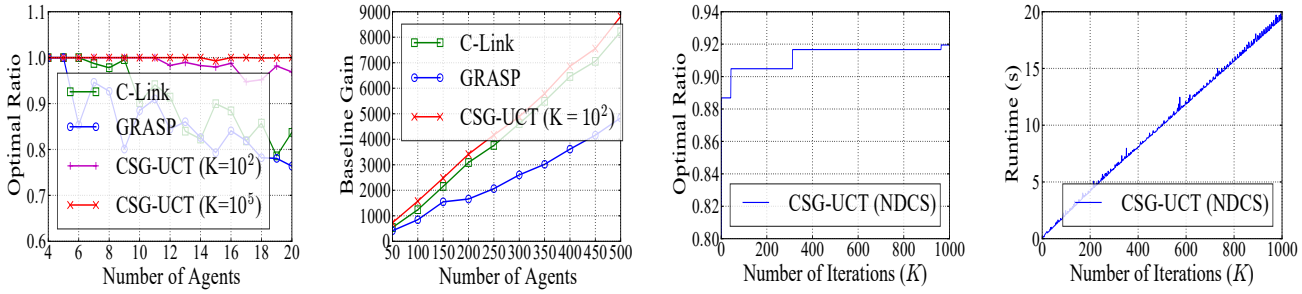
(c) Modified Uniform (Optimal Ration and Baseline Gain)

(d) Modified Normal (Optimal Ration and Baseline Gain)



(e) NDCS (Optimal Ration and Baseline Gain)

(f) Agent-Based (Optimal Ration and Baseline Gain)



(g) Disaster Response (Optimal Ration and Baseline Gain)

(h) Anytime (Iterations vs. Optimal Ration and Runtime)

Figure 2: Experimental Results on Six Common Benchmark Problems and the Disaster Response Domain

agents. Specifically, we vary the number of iterations used in CSG-UCT and observe how it affects the solution quality and runtime of the algorithm. As we can see from Figure 2(h), the solution quality constantly improves when the number of iterations increases. As expected, the runtime increases linearly when the number of iterations grows. In practice, we can repeatedly run the sampling steps until it runs out of time and return the currently best solution. This is appealing in

domains where time and resources are limited.

5.2 Case Study: Disaster Response

Our work is motivated by a disaster scenario [Ramchurn *et al.*, 2015; Ramchurn *et al.*, 2016], in which a satellite, powered by radioactive fuel, has crashed in a sub-urban area. Debris is strewn around a large area, damaging buildings and causing accidents and injuring civilians. Moreover, radioactive particles discharged from the debris are gradually spreading over the area, threatening to contaminate food reserves and people. Hence, emergency services are deployed to evacuate the casualties and key assets before they are engulfed by the radioactive cloud. As in the scenario [Ramchurn *et al.*, 2016], the mission of emergency responders is to evacuate 4 types of targets as: victim, animal, fuel, or other resource, which require a combination of 4 roles played by the responders as: medic, fire-fighter, soldier, or transporter. Before being despatched, hundreds of human responders must coordinate together and form teams quickly in a way that the overall performance is maximized.

Generally, given a team of responders C, there are many ways to estimate its team performance and define the utility $v(C)$ required by team formation algorithms. As a case study, we consider a simple model as follows. Each responder i has been assigned a value $c_i^r \sim U(0, 10)$ to indicate her capability to play the role r based on her training, past experience, body condition, etc. For each target k, the upper bound of the team performance is defined as: $\bar{p}_k^C = \sum_{r \in \Theta_k} \sum_{i \in C} c_i^r$, where Θ_k is the set of roles required by the target. Then, the utility for the team is denoted as: $v(C) = \sum_{k=1}^{4} p_k^C$, where $p_k^C \sim U(0, \bar{p}_k^C)$ is the estimate of the performance of team T on target k subject to uncertainty. Note that this is just a specification for the team utility. In practice, we can use *any utility function* designed by the domain experts.

Figure 2(g) reports the experimental results on the disaster response domain. As we can see from the figure, CSG-UCT outperforms both GRASP and C-Link to achieve more gains comparing to the baseline performance if no team is formed. Notice that this domain with hundreds of responders is intractable for existing optimal algorithms. Thus, we must use approximate techniques to solve such CSG problems. However, existing approximate methods such as GRASP and C-Link are not anytime and have no guarantee on solution quality. Therefore, their performance cannot be improved even more time and resources are available.

6 Related Work

Optimal approaches for solving CSG are mostly based on *Dynamic Programming* (DP). The first DP algorithm [Yeh, 1986] works by iteratively determining, for every coalition, whether it is beneficial to split it into two coalitions and storing the best split. The DP algorithm computes an optimal coalition structure in $\mathcal{O}(3^n)$ time and may take hours to handle 30 agents on a modern desktop computer [Rahwan *et al.*, 2015]. Improvements such as ODP and IDP were made by avoiding some operations of DP. Following early efforts [Sandholm *et al.*, 1999; Dang and Jennings, 2004] on identifying the worst-case guarantees if only subspace is searched, an anytime algorithm, named IP [Rahwan *et al.*, 2009], was proposed based on the integer partition-based representation. To further boost the performance, approaches such as IDP-IP and ODP-IP [Michalak *et al.*, 2016] were proposed to combine techniques of DP and IP. These algorithms only take seconds to handle 30 agents but is unlikely to solve problems of 40 agents or more on a modern desktop computer [Rahwan *et al.*, 2015]. Generally, optimal algorithms are not scalable to large problems with more than 40 agents.

A number of heuristics have been developed to solve large CSG problems approximately, ranging from greedy methods [Shehory and Kraus, 1998; Di Mauro *et al.*, 2010] to genetic or local search algorithms [Sen and Dutta, 2000; Keinänen, 2009]. Among them, GRASP [Di Mauro *et al.*, 2010] is a greedy method that, at each iteration, a coalition structure is constructed greedily and then a local search is used to improve the coalition structure by exploring its neighbours. C-Link [Farinelli *et al.*, 2013] is another greedy based approach motivated by data clustering methods. It starts from the structure of all singleton coalitions and iteratively selects a pair of coalitions to be merged. The pair of coalitions are chosen based on the suitability function in a myopic manner, without taking into consideration the future consequences of this choice. Hence, it can be trapped in local maxima of the objective function. To date, both GRASP and C-Link are the leading approximate CSG algorithms with good empirical results. Although approximate methods return solutions relatively quickly, they do not provide any guarantees on solution quality. Some efforts have been made to bound the solution quality but they usually require the characteristic function (i.e., the utility function) to have some special structure (e.g., the $m + a$ functions as in [Farinelli *et al.*, 2017]).

Recently, several approaches [Rahwan *et al.*, 2011; Bistaffa *et al.*, 2017] were proposed to exploit specific domain structures and improve the performance. Existing CSG algorithms can be speeded up using GPU [Pawłowski *et al.*, 2014] or multi-core machines [Cruz *et al.*, 2017]. However, scalability is still an issue. A comprehensive survey on the CSG algorithms can be found in [Rahwan *et al.*, 2015].

7 Conclusions

This paper presented a scalable algorithm based on UCT for the CSG problem. Specifically, we proposed the CSG-UCT algorithm that builds a search tree by sampling the solution space of coalition structures. We show that CSG-UCT is complete and will eventually converge to the optimal given sufficient number of iterations. Moreover, it is anytime and can return the current best solution if time is limited. In the future, we plan to further improve the performance on very large CSG problems by exploiting the properties of coalition structures and test our method on real-world applications.

Acknowledgements

This work was supported in part by the National Key R&D Program of China (2017YFB1002204), the National Natural Science Foundation of China (U1613216, 61603368), and the Guangdong Province Science and Technology Plan (2017B010110011).

References

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[Bistaffa *et al.*, 2017] Filippo Bistaffa, Alessandro Farinelli, Georgios Chalkiadakis, and Sarvapali D Ramchurn. A cooperative game-theoretic approach to the social ridesharing problem. *Artificial Intelligence*, 246:86–117, 2017.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, et al. A survey of monte carlo tree search methods. *IEEE Trans. on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[Cruz *et al.*, 2017] Francisco Cruz, Antonio Espinosa, Juan C Moure, Jesus Cerquides, et al. Coalition structure generation problems: optimization and parallelization of the idp algorithm in multicore systems. *Concurrency and computation*, 29(5):e3969, 2017.

[Dang and Jennings, 2004] Viet Dung Dang and Nicholas R Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Proc. of AAMAS*, pages 564–571, 2004.

[Di Mauro *et al.*, 2010] Nicola Di Mauro, Teresa MA Basile, Stefano Ferilli, and Floriana Esposito. Coalition structure generation with grasp. In *Proc. of Int'l Conf. on Artif. Intel.: Methodology, Systems, and Applications*, pages 111–120, 2010.

[Farinelli *et al.*, 2013] Alessandro Farinelli, Manuele Bicego, Sarvapali D Ramchurn, and Mauro Zucchelli. C-link: A hierarchical clustering approach to large-scale near-optimal coalition formation. In *Proc. of IJCAI*, pages 106–112, 2013.

[Farinelli *et al.*, 2017] Alessandro Farinelli, Manuele Bicego, Filippo Bistaffa, and Sarvapali D Ramchurn. A hierarchical clustering approach to large-scale near-optimal coalition formation with quality guarantees. *Engineering Applications of Artificial Intelligence*, 59:170–185, 2017.

[Keinänen, 2009] Helena Keinänen. Simulated annealing for multi-agent coalition formation. In *KES Int'l Symposium on Agent and Multi-Agent Systems*, pages 30–39, 2009.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proc. of ECAI*, pages 282–293, 2006.

[Larson and Sandholm, 2000] Kate S Larson and Tuomas W Sandholm. Anytime coalition structure generation: an average case study. *J. of Exper. & Theoretical Artif. Intel.*, 12(1):23–42, 2000.

[Michalak *et al.*, 2016] Tomasz Michalak, Talal Rahwan, Edith Elkind, Michael Wooldridge, and Nicholas R Jennings. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230:14–50, 2016.

[Pawłowski *et al.*, 2014] Krzysztof Pawłowski, Karol Kurach, Kim Svensson, Sarvapali Ramchurn, et al. Coalition structure generation with the graphics processing unit. In *Proc. of AAMAS*, pages 293–300, 2014.

[Rahwan *et al.*, 2007] Talal Rahwan, Sarvapali D Ramchurn, Viet Dung Dang, Andrea Giovannucci, and Nicholas R Jennings. Anytime optimal coalition structure generation. In *Proc. of AAAI*, pages 1184–1190, 2007.

[Rahwan *et al.*, 2009] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, et al. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research*, 34:521–567, 2009.

[Rahwan *et al.*, 2011] Talal Rahwan, Tomasz P Michalak, Edith Elkind, Piotr Faliszewski, Jacek Sroka, et al. Constrained coalition formation. In *Proc. of AAAI*, volume 11, pages 719–725, 2011.

[Rahwan *et al.*, 2012] Talal Rahwan, Tomasz Michalak, and Nicholas Jennings. A hybrid algorithm for coalition structure generation. In *Proc. of AAAI*, 2012.

[Rahwan *et al.*, 2015] Talal Rahwan, Tomasz P Michalak, Michael Wooldridge, and Nicholas R Jennings. Coalition structure generation: A survey. *Artificial Intelligence*, 229:139–174, 2015.

[Ramchurn *et al.*, 2015] Sarvapali D. Ramchurn, Feng Wu, Joel E. Fischer, Steven Reece, Wenchao Jiang, et al. Human-agent collaboration for disaster response. *Journal of Autonomous Agents and Multi-Agent Systems*, 30(1):82–111, 2015.

[Ramchurn *et al.*, 2016] Sarvapali D Ramchurn, Trung Dong Huynh, Feng Wu, Yukki Ikuno, Jack Flann, Luc Moreau, et al. A disaster response system based on human-agent collectives. *Journal of Artificial Intelligence Research*, 57:661–708, 2016.

[Sandholm *et al.*, 1999] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1-2):209–238, 1999.

[Sen and Dutta, 2000] Sandip Sen and Partha Sarathi Dutta. Searching for optimal coalition structures. In *Proc. of AAMAS*, pages 287–292. IEEE, 2000.

[Service and Adams, 2010] Travis Service and Julie Adams. Approximate coalition structure generation. In *Proc. of AAAI*, 2010.

[Service and Adams, 2011] Travis C. Service and Julie A Adams. Coalition formation for task allocation: Theory and algorithms. *JAAMAS*, 22(2):225–248, 2011.

[Shehory and Kraus, 1998] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial intelligence*, 101(1):165–200, 1998.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

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