

A Multiagent Path Search Algorithm for Large-Scale Coalition Structure Generation

Redha Taguelmimt¹, Samir Aknine², Djamila Boukreda³, Narayan Changder⁴,
Tuomas Sandholm^{5,6,7,8}

¹Clermont Auvergne University, Clermont Auvergne INP, CNRS, LIMOS, F-63000 Clermont-Ferrand, France.

²Univ Lyon, UCBL, CNRS, INSA Lyon, Centrale Lyon, Univ Lyon 2, LIRIS, UMR5205, Lyon, France

³Laboratory of Applied Mathematics, Faculty of Exact Sciences, University of Bejaia, Bejaia, Algeria

⁴TCG Centres for Research and Education in Science and Technology, Kolkata, India

⁵Carnegie Mellon University, Computer Science Department, Pittsburgh, USA

⁶Strategy Robot, Inc.

⁷Strategic Machine, Inc.

⁸Optimized Markets, Inc.

redha.taguelmimt@gmail.com, samir.aknine@univ-lyon1.fr, djamila.boukreda@univ-bejaia.dz,
narayan.changder@tcgcrest.org, sandholm@cs.cmu.edu

Abstract

Coalition structure generation (CSG), i.e. the problem of optimally partitioning a set of agents into coalitions to maximize social welfare, is a fundamental computational problem in multiagent systems. This problem is important for many applications where small run times are necessary, including transportation and disaster response. In this paper, we develop SALDAE, a multiagent path finding algorithm for CSG that operates on a graph of coalition structures. Our algorithm utilizes a variety of heuristics and strategies to perform the search and guide it. It is an anytime algorithm that can handle large problems with hundreds and thousands of agents. We show empirically on nine standard value distributions, including disaster response and electric vehicle allocation benchmarks, that our algorithm enables a rapid finding of high-quality solutions and compares favorably with other state-of-the-art methods.

1 Introduction

Coalition formation is a major problem in artificial intelligence that is central to many practical applications. Coalitions of delivery companies can, for instance, be formed to reduce transportation costs by sharing deliveries (Sandholm and Lesser 1997). In disaster response, hundreds of human responders can be quickly organized into teams to coordinate their evacuation and rescue actions (Wu and Ramchurn 2020). Central to coalition formation is the *coalition structure generation* problem. It consists of identifying the optimal partitioning of a set of agents— that is, an optimal set of coalitions among agents that maximizes the sum of the values of the coalitions (an optimal coalition structure).

Several algorithms have been proposed for this problem, including optimal and approximate solutions. Optimal solutions require a huge execution time and a large storage space due to the exponentiality of the input and the solution space. This limits their applicability to large-scale prob-

lems. Indeed, due to this, exact algorithms can only handle small numbers of agents (around 30) (Wu and Ramchurn 2020). Furthermore, we do not expect an algorithm that always finds an optimal solution to this problem for large-scale settings with hundreds of agents, as there is no proven guarantee that an algorithm can find an optimal solution without enumerating all 2^n coalition values. To meet the need of addressing this limitation, scalable solutions that scale to hundreds and thousands of agents have been suggested. Along this line, approaches such as CSG-UCT (Wu and Ramchurn 2020), PICS (Taguelmimt et al. 2022) and C-Link (Farinelli et al. 2013) have been proposed. However, to our knowledge, PICS, and CSG-UCT produce the best results among existing state-of-the-art algorithms for solving large-scale problem instances.

Given the complexity of this problem, we propose SALDAE (Scalable Algorithm with Large-Scale Distributed Agent Exploration)—a scalable and anytime algorithm for solving the *coalition structure generation* problem. Specifically, we develop a variant of multiagent path finding for coalition structure generation by gradually building a search graph of coalition structures based on three expansion steps. To the best of our knowledge, this is the first algorithm for this problem using any variant of path finding. This algorithm is scalable and can be run with large problem instances with thousands of agents, which is hard to achieve with existing state-of-the-art exact algorithms. Moreover, it can return an anytime solution when time is limited. To summarize, our main contributions are:

- We present SALDAE, a new algorithm inspired by multiagent path finding concepts to search the coalition structure graph. It is anytime and scales to thousands of agents.
- To improve the search process, we propose various strategies inspired by MAPF (Multiagent path finding) techniques for connecting the best solutions found at a given step during the execution in an effort to find even better ones. We also explore different techniques for select-

ing child nodes and resolving conflicts between search agents, adapting ideas from MAPF to the CSG domain. Our results show that the use of these MAPF-inspired heuristics can significantly reduce the number of search steps required to find high-quality solutions.

- We empirically show that SALDAE outperforms the state-of-the-art algorithms for solving both small and large problems when generating anytime solutions.

2 Problem Formulation

In CSG, we are given a set of n agents, represented by $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$, and a characteristic function v that assigns a real value to each coalition, indicating the efficiency of the coalition. A coalition \mathcal{C} is any non-empty subset of \mathcal{A} , and the size of \mathcal{C} is denoted by $|\mathcal{C}|$. A coalition structure \mathcal{CS} is a partition of the set of agents \mathcal{A} into disjoint coalitions, formally defined as a collection of coalitions $\mathcal{CS} = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$, where $k = |\mathcal{CS}|$, and the following constraints are satisfied: $\bigcup_{j=1}^k \mathcal{C}_i = \mathcal{A}$ (all agents are included in the coalition structure) and for all $i, j \in \{1, 2, \dots, k\}$ where $i \neq j$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ (each agent is included in exactly one coalition).

Let $\Pi(\mathcal{A})$ denote the set of all coalition structures. The value of a coalition structure \mathcal{CS} is assessed as the sum of the values of the disjoint coalitions that comprise it: $v(\mathcal{CS}) = \sum_{\mathcal{C} \in \mathcal{CS}} v(\mathcal{C})$. The CSG problem aims at finding the optimal solution, which is the most valuable coalition structure $\mathcal{CS}^* \in \Pi(\mathcal{A})$, i.e. $\mathcal{CS}^* = \arg\max_{\mathcal{CS} \in \Pi(\mathcal{A})} v(\mathcal{CS})$, for a given set of agents \mathcal{A} . However, when time is limited having a good-enough quality solution within a reasonable time is more valuable.

The coalition structure graph, first introduced by (Sandholm et al. 1999), is a way to represent the search space as a graph composed of nodes representing the coalition structures. For a given set of n agents, these nodes are organized into n levels, where each level consists of nodes representing coalition structures that contain exactly i coalitions ($i \in \{1, \dots, n\}$). Each edge of this graph connects two nodes belonging to two consecutive levels, such that each coalition structure at level i can be obtained by dividing a coalition from a coalition structure at level $i - 1$ into two coalitions.

3 Related Work

Many approaches have been proposed to solve the CSG problem either optimally or approximately, including dynamic programming algorithms, anytime algorithms, heuristic algorithms, and scalable solutions. Dynamic programming approaches, such as those proposed in (Yeh 1986; Rahwan and Jennings 2008; Michalak et al. 2016), guarantee to find the optimal coalition structure but must be run to completion to do so. Anytime algorithms (Sandholm et al. 1999; Dang and Jennings 2004; Rahwan et al. 2009), on the other hand, allow for premature termination while providing intermediate solutions during execution. The hybrid algorithms that combine dynamic programming algorithms with anytime algorithms (Michalak et al. 2016; Changder et al. 2020, 2021; Taguelmimt et al. 2024, 2023) are the fastest exact algorithms.

Heuristic algorithms, such as those proposed in (Sen and Dutta 2000; Keinänen 2009; Di Mauro et al. 2010), prioritize speed and do not guarantee to find an optimal solution. These algorithms are useful when the number of agents increases and the problem becomes too hard to solve optimally. For instance, the simulated annealing method (Keinänen 2009), a stochastic local search algorithm, explores different neighborhoods of coalition structures by splitting, merging, or shifting agents. It starts with a random coalition structure and moves to a new one in its neighborhood at each iteration, with the movement accepted with a probability that depends on the difference in utility and a decreasing temperature parameter.

Very few scalable solutions to CSG exist. For example, the Monte Carlo tree search method proposed in (Wu and Ramchurn 2020) finds solutions by sampling the coalition structure graph and partially expanding a search tree that corresponds to a partial search space that has been explored. The hierarchical clustering approach proposed in (Farinelli et al. 2013) builds a coalition structure by merging coalitions using a similarity criterion based on the gain that the system achieves if two coalitions merge. The search algorithms FACS and PICS proposed in (Taguelmimt et al. 2021, 2022) generate coalition structures based on code permutations applied to selected initial vectors of a different search space representation. To the best of our knowledge, PICS (Taguelmimt et al. 2022) and CSG-UCT (Wu and Ramchurn 2020) are the best performing of the prior algorithms. Our proposed algorithm evaluates possible splits of coalitions and possible merges of coalition pairs. Unlike some other approaches, such as the C-Link method (Farinelli et al. 2013), which cannot split coalitions once they are merged, our algorithm allows for coalitions to be split and merged multiple times. This makes our algorithm less likely to get trapped in local maxima.

Multiagent path finding (MAPF) (Stern 2019) is another important problem in multiagent systems, where agents must navigate through a given environment to reach their goals while avoiding collisions with each other. This problem of planning paths for multiple agents is also known to be NP-hard (Yu and LaValle 2013) and has been studied extensively in the literature. It is inspired by real-world applications such as warehouse logistics (Ma et al. 2017), autonomous aircraft-towing vehicles (Morris et al. 2016), airport operations (Li et al. 2019), and video games (Li et al. 2020b). One of the most popular algorithms for MAPF is the Conflict-Based Search (CBS) algorithm (Sharon et al. 2015). CBS is a complete algorithm that guarantees to find the optimal solution if one exists. Other algorithms (Gange, Harabor, and Stuckey 2021; Barer et al. 2014; Li et al. 2021; Li, Ruml, and Koenig 2021; Li et al. 2020a; Li, Ruml, and Koenig 2020) based on CBS and other methods have been developed to solve this problem in optimal, suboptimal, or bounded suboptimal ways. In the next section, we explain how we draw inspiration from MAPF concepts to enhance the search for solutions to CSG and propose a new algorithm that adapts ideas from MAPF for solving the CSG problem.

In this paper, we propose a path search algorithm for finding optimal coalition structures in coalition structure gener-

ation. The algorithm operates on a graph where each node represents a coalition structure, and a search agent explores this graph aiming to reach the highest-valued solution. The algorithm starts from a designated start node and iteratively explores neighboring nodes, guided by the values of coalition structures.

The key components of this approach that will be detailed in the following sections are:

- The search space is represented as a graph, with each node corresponding to a coalition structure. Transitions between nodes occur through coalition splitting or merging.
- A search agent explores the graph by moving from one node to another, selecting nodes based on their coalition structure values.
- Upon finding better solutions, the algorithm creates paths between previous and new solutions to explore potential improvements along the path.
- To optimize search efficiency, the algorithm employs memory management techniques, maintaining lists of nodes in memory and dynamically adjusting them based on solution quality.
- Multiple search agents are employed to accelerate finding high-quality solutions, with conflict resolution mechanisms to prevent redundant exploration.

4 Path Search Algorithm for CSG

In this paper, we consider a path finding variant where each node is a solution to the CSG problem, i.e. a coalition structure. The goal is thus to find the optimal one or at least approach its value. This variant is defined by a graph and a search agent that begins at a start node and can move to an adjacent node at each step. The search agent maintains a list of nodes, sorted according to the values of the corresponding coalition structures. A path in this context is a sequence of nodes that are adjacent to each other, starting at the start node. The decision to pursue one path over another is based on the coalition structure value of the reached node. Hence, the cost of a path is the value of the coalition structure of its last expanded node.

Multiagent Path Finding (Stern et al. 2019; Stern 2019) also has many variants. Our goal is to find the optimal solution using multiple search agents. Inspired by concepts from MAPF, our variant is defined by a graph and a set of m search agents $\{s_1, \dots, s_m\}$. Each search agent s_i has a designated start node and can move to an adjacent node at each step. Conflicts can occur when two search agents consider the same coalition structure for evaluation. While the paths taken by the search agents are important for finding high-quality solutions quickly, they are not the solution themselves. The optimal solution is the highest-valued coalition structure found in the nodes.

Our path search algorithm for the optimal coalition structure uses a search graph (see Figure 1). This graph is built gradually, starting from a designated start node that represents a starting coalition structure, which can be the top node (that represents the coalition structure containing the singleton coalitions), the bottom node (i.e. the coalition structure

composed of the grand coalition that contains all the agents) or any other node. Each parent node in the search graph is connected to a number of child nodes that can be generated by either splitting a coalition or joining two coalitions in the parent coalition structure. Given a coalition structure $\mathcal{CS} = \{C_1, C_2, \dots, C_k\}$:

- splitting a coalition C_i into C_j and C_k (i.e. $C_i = C_j \cup C_k$ with $C_j, C_k \neq \emptyset$ and $C_j \cap C_k = \emptyset$) in \mathcal{CS} results in a new coalition structure $(\mathcal{CS} \setminus \{C_i\}) \cup \{C_j, C_k\}$.
- merging two coalitions C_i and C_j in \mathcal{CS} , with $i \neq j$, results in a new coalition structure $(\mathcal{CS} \setminus \{C_i, C_j\}) \cup \{C_i \cup C_j\}$.

Each node in the search graph represents a potential solution. For the sake of clarity, we will consider the bottom node as the start node in the remainder of this section. The search graph is constructed iteratively by adding new nodes generated according to the following steps, which constitute a type of greedy algorithm. Throughout this process, the terms "node", "coalition structure", and "solution" may be used interchangeably.

1. *Step 1: Generation:* In this step, child coalition structures are generated from the current start node by either splitting a coalition or joining two coalitions. For the first iteration, if the bottom node is the start node, child nodes can only be generated by splitting the grand coalition. Each newly generated child node is added to the search graph. Figure 1 shows an example with 4 agents. The numbers in Figure 1 represent the agents. For example, $\{1, 2, 3, 4\}$ represents the coalition structure $\{a_1, a_2, a_3, a_4\}$.
2. *Step 2: Selection:* This step defines the start node selection procedure. When a set of child nodes is generated, they all form a set of candidate coalition structures to be the start node for the next iteration. Given this, the algorithm selects the most promising node to consider as the start node \mathcal{SN} for the next iteration. The start node is chosen as the highest-valued child coalition structure, i.e. $\mathcal{SN} = \arg \max_{\mathcal{CS} \in \text{Child}} V(\mathcal{CS})$, where *Child* is the current child nodes generated in the graph, not just the child nodes generated for one node. Figure 1 illustrates this procedure. For this example, we consider the node $\{\{a_1, a_4\}, \{a_2, a_3\}\}$ as the highest valued child coalition structure. All the nodes in level 2 were candidates for this selection (see Figure 1).
3. *Step 3: Comparison to incumbent:* This step involves evaluating the selected coalition structure to determine if it is better than the current best solution found in previous iterations. If it is, the best solution is updated with the new one. The value of a coalition structure is determined by summing the values of the coalitions that comprise it. After evaluating the coalition structure, another iteration begins with the new start node, which is the tested coalition structure. Figure 1 shows a 4-agent illustration example.

While these steps provide the foundational framework of the algorithm, additional optimizations and strategies are subsequently introduced to enhance performance, offering further refinements beyond the initial three-step process.

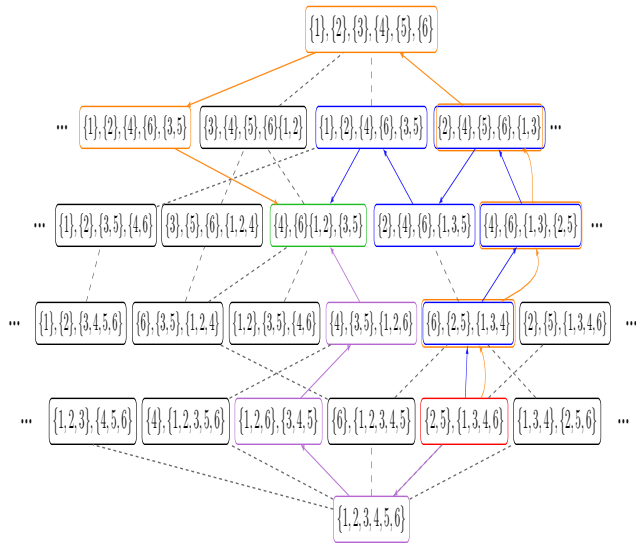


Figure 2: An illustration of the 3 strategies with a partial graph of 6 agents. The numbers represent the agents. For example, 1 represents agent a_1 . The last best solution is the red node and the new best one is the green node. The nodes of the path between the two current best solutions of the SPLIT-THEN-MERGE, MERGE-THEN-SPLIT and APPROACH-THEN-SWAP strategies are represented by the orange, purple, and blue nodes, respectively.

APPROACH-THEN-SWAP This strategy consists of two phases: an approach phase and a swap phase. In the approach phase, the algorithm generates an intermediate solution that is at the same level as \mathcal{CS}_n . To reach this intermediate solution, the algorithm performs a series of splits or a series of merges on the coalitions of \mathcal{CS}_l . If the level of \mathcal{CS}_n is higher than that of \mathcal{CS}_l , the algorithm performs $l_2 - l_1$ splits, where l_1 and l_2 are the levels of \mathcal{CS}_l and \mathcal{CS}_n , respectively. If the level of \mathcal{CS}_n is lower than that of \mathcal{CS}_l , the algorithm performs $l_1 - l_2$ merges. If the two coalition structures are at the same level, no splitting or merging is performed.

Once the intermediate solution \mathcal{CS}_i that has the same number of coalitions and the same number of agents in each coalition as \mathcal{CS}_n is reached, the swap phase begins. This phase involves swapping agents between coalitions in order to reach \mathcal{CS}_n . Given a coalition structure $\mathcal{CS} = \{C_1, C_2, \dots, C_k\}$, swapping an agent a_i of a coalition C_i with an agent a_j of a coalition C_j results in a new coalition structure $(\mathcal{CS} \setminus \{C_i, C_j\}) \cup \{C'_i, C'_j\}$, where $C'_i = (C_i \setminus \{a_i\}) \cup \{a_j\}$ and $C'_j = (C_j \setminus \{a_j\}) \cup \{a_i\}$. This rule is equivalent to two series of split and merge. To perform a swap, the algorithm splits C_i into $C_i \setminus \{a_i\}$ and $\{a_i\}$. Then, merges $\{a_i\}$ with C_j to obtain $C_j \cup \{a_i\}$. Again, splits $C_j \cup \{a_i\}$ into $(C_j \cup \{a_i\}) \setminus \{a_j\}$ and $\{a_j\}$. Finally, merges $C_i \setminus \{a_i\}$ with $\{a_j\}$ to obtain $(C_i \setminus \{a_i\}) \cup \{a_j\}$. For example, swapping a_2 with a_3 in the coalition structure $\{\{a_4\}, \{a_1, a_2\}, \{a_3, a_5\}\}$ is done as shown in Figure 3.

All the intermediate coalition structures that the path gen-

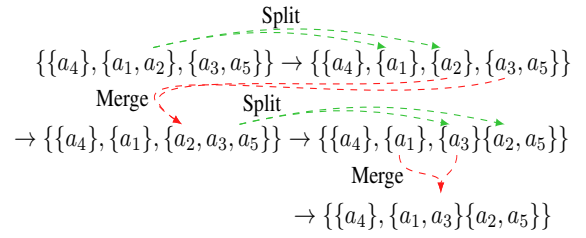


Figure 3: An illustration of the APPROACH-THEN-SWAP strategy.

erates are evaluated with the aim to find better solutions. Figure 2 shows an illustration of this strategy (see the blue path).

4.2 Optimizing the Search for Global Optima through Memory Management

SALDAE maintains three lists of nodes in memory: OPEN, SUBSTITUTE, and RESERVE. OPEN is sorted in descending order according to the values of the coalition structures represented by the nodes it contains. It is constructed using the three steps of the search process (see Section 4). SUBSTITUTE contains the nodes visited by the algorithm during the path search between the current best solutions. These nodes are not expanded unless they improve the current solution. Two hyperparameters, θ and ω , are introduced to control the memory usage of the algorithm. θ determines the number of child nodes to keep in memory for each expanded node, while ω is the lower bound on the minimum value required for a node to be maintained in the list OPEN. Let \mathcal{CS}^+ be the current best solution. The list OPEN contains nodes \mathcal{CS} such that $v(\mathcal{CS}) \geq \omega \times v(\mathcal{CS}^+)$, while RESERVE contains nodes \mathcal{CS} with $v(\mathcal{CS}) < \omega \times v(\mathcal{CS}^+)$. The list OPEN is used by SALDAE for the search and is replaced by either the SUBSTITUTE or RESERVE list if it becomes empty.

Replacing OPEN with SUBSTITUTE Since the list OPEN contains the nodes where $v(\mathcal{CS}) \geq \omega \times v(\mathcal{CS}^+)$, it is possible that after several iterations OPEN becomes empty. This can happen if the nodes expanded by SALDAE do not generate child nodes that meet the condition ($v(\mathcal{CS}) \geq \omega \times v(\mathcal{CS}^+)$), i.e. the child nodes are directly added to the list RESERVE. In this case, the algorithm replaces OPEN with SUBSTITUTE and continues the search. This allows SALDAE to explore different areas of the solution space that may be more promising, instead of staying in a part of the space that has not produced better solutions or at least solutions that would have remained in OPEN.

Replacing OPEN with RESERVE In SALDAE, SUBSTITUTE is constructed after finding a new best solution. However, when OPEN is replaced with SUBSTITUTE, the list SUBSTITUTE remains empty, until a better solution is found. If before finding a better solution OPEN becomes empty, SALDAE replaces OPEN with the list RESERVE and continues the search. This way, SALDAE prioritizes searching in parts of the solution space that are more promis-

ing, but can also return to the list RESERVE if it does not find more promising parts.

By using these three lists, SALDAE can effectively guide its search to focus on more promising areas of the solution space, while still being able to revisit areas that may have been overlooked.

4.3 Multiagent Search

To speed up finding high-quality solutions, SALDAE employs several search agents $\{s_1, \dots, s_m\}$. Each agent s_i has a start node and maintains three lists of nodes $OPEN_i$, $RESERVE_i$, and $SUBSTITUTE_i$. SALDAE affects one agent for each of the bottom and top nodes, and random nodes to other agents. Then, each agent moves from a node to an adjacent node searching for better solutions.

When a search agent s_i expands a node and generates child nodes, it checks for conflicts among the lists of the search agents. We say that two agents have a conflict iff they consider the same coalition structure for evaluation. If there are no conflicts, the agent s_i adds the generated child node to the list $OPEN_i$ or $RESERVE_i$. Otherwise, s_i resolves the conflict by selecting the search agent that will keep the generated child node, using the following techniques. We introduce the following techniques to resolve the conflicts.

Bypassing Conflicts If a child node is found in an $OPEN_j$ or $RESERVE_j$ list of another search agent s_j , the search agent s_i is prohibited from using the conflicting child node and s_j keeps it. This ensures that the first search agent to consider the node is the one that keeps it. This is different from the bypassing conflicts in CBS, where SALDAE aims to bypass the conflicts by generating coalition structures that avoid leading to conflicts in the first place. The goal of doing this in SALDAE is to guarantee that the search agents do not search the same coalition structures at the same time and explore different areas of the solution space.

Managing Conflicts Since the OPEN and RESERVE lists are not the same for each agent, the search agent s_i compares the ranking of the child node in its own list with the ranking in the list of the conflicting search agent s_j . If the search agent s_i allows the expansion of the conflicting child node first, it will remove the node from the list of the other search agent s_j and add it to its own list. This indicates that the treatment of the conflicting child node is removed from the search agent s_j . Otherwise, the search agent s_i is prohibited from using the conflicting child node and s_j keeps it. This ensures that the search agent that allows the expansion of the conflicting child node first gets to keep it.

Algorithm 1 shows the pseudocode of SALDAE. Lines 8 and 12 are discussed in the section 5 of the paper. SALDAE runs in parallel m search agents s_i . ExecutePathStrategy function constructs a path between the last best solutions and adds the visited nodes to the list $SUBSTITUTE_i$. AddChildNodesToOPENorRESERVE function distributes the nodes between the lists $OPEN_i$ and $RESERVE_i$ according to their coalition structure values:

1. if $v(\mathcal{N}) \geq \omega \times v(\mathcal{CS}^+)$ then add \mathcal{N} to $OPEN_i$;
2. else add \mathcal{N} to $RESERVE_i$.

Algorithm 1: SALDAE algorithm

Input: A number of agents n and the values $v(\mathcal{C})$ of the coalitions \mathcal{C} .
Output: The highest-valued coalition structure found \mathcal{CS}^* and its value.

- 1 Generate root node R
- 2 $\mathcal{CS}^* \leftarrow R$
- 3 $\mathcal{V}^* \leftarrow v(R)$
- 4 Generate start nodes R_i for the search agents s_i
 - ▷ Begin parallel
 - ▷ SALDAE runs in parallel m search agents s_i
- 5 **if** $v(R_i) > \mathcal{V}^*$ **then**
- 6 | $\mathcal{CS}^* \leftarrow R_i$
- 7 | $\mathcal{V}^* \leftarrow v(R_i)$
- 8 Childs \leftarrow ComputeChildNodes(R_i)
- 9 AddChildNodesToOPENorRESERVE(Childs)
- 10 **while** $OPEN_i$ is not empty **do**
- 11 | $\mathcal{N} \leftarrow OPEN_i.pop()$ ▷ this selects the highest-valued node
- 12 | Childs \leftarrow ComputeChildNodes(\mathcal{N})
- 13 | AddChildNodesToOPENorRESERVE(Childs)
- 14 | **if** $v(\mathcal{N}) > \mathcal{V}^*$ **then**
- 15 | | ExecutePathStrategy($\mathcal{CS}^*, \mathcal{N}$)
- 16 | | **if** $v(\mathcal{N}) > \mathcal{V}^*$ **then**
- 17 | | | $\mathcal{CS}^* \leftarrow \mathcal{N}$
- 18 | | | $\mathcal{V}^* \leftarrow v(\mathcal{N})$
- 19 | RemoveNodeFrom $OPEN_i$ (\mathcal{N})
- 20 | **if** $OPEN_i$ is empty **then**
- 21 | | **if** $SUBSTITUTE_i$ is empty **then**
- 22 | | | Replace $OPEN_i$ by $RESERVE_i$
- 23 | | **else**
- 24 | | | Replace $OPEN_i$ by $SUBSTITUTE_i$
- ▷ End parallel
- 25 **Return** $\mathcal{CS}^*, \mathcal{V}^*$

5 Selecting Child Nodes

Due to the large number of possible child nodes in the coalition structure graph, it is infeasible to generate all of them, especially for large numbers of agents, this makes the search space intractable and highlights the importance of using more efficient selection strategies. Here, we present and introduce two child node selection methods.

5.1 Quantity-Based Selection

A straightforward idea for generating child nodes that can reduce computational burden and memory requirements is to select a number of child nodes and then keep the best ones in $OPEN_i$ and $RESERVE_i$. Let \mathcal{N}_c and \mathcal{N}_a be the number of child nodes to generate for the list $OPEN_i$ and the number of child nodes to actually add to the list $OPEN_i$, with $\mathcal{N}_a < \mathcal{N}_c$. In the first step of generating child nodes, we generate \mathcal{N}_c child nodes, when they exist, whose values are greater than or equal to a threshold value, $\omega \times v(\mathcal{CS}^+)$. From these generated child nodes, we select the \mathcal{N}_a nodes with the highest values and add them to the list $OPEN_i$. Any generated nodes that have values less than $\omega \times v(\mathcal{CS}^+)$ are added

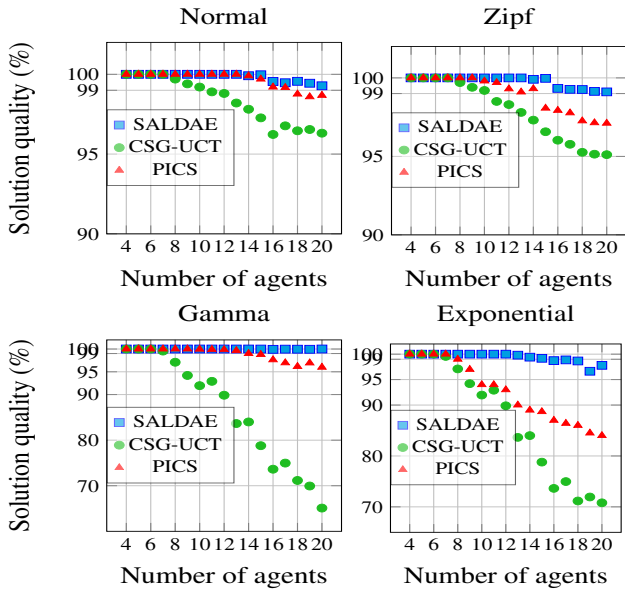


Figure 4: Solution quality of SALDAE, PICS and CSG-UCT for sets of agents between 4 and 20.

to the list RESERVE_i .

5.2 Value-Based Selection

The second idea is to select \mathcal{N}_a child nodes to add to the list OPEN_i and use an exit condition to stop the generation of child nodes based on the values of the child nodes. We start by fixing an initial threshold parameter $\gamma = v(\mathcal{CS}^+)$, which allows us to specify the acceptance of the generated child nodes. At each iteration, we generate \mathcal{N}_a random child nodes and add them to a temporary list. If one of the generated child nodes, \mathcal{CS} , has a value $v(\mathcal{CS}) > \gamma$, we add the \mathcal{N}_a best nodes from the temporary list to OPEN_i or RESERVE_i . Otherwise, we reduce the threshold of acceptance by half the difference between the current threshold and the best value of the generated child nodes, i.e., $\gamma = \gamma - \left(\frac{\gamma - v(\mathcal{CS}^\#)}{2}\right)$, where $v(\mathcal{CS}^\#)$ is the value of the highest valued generated child node. This way, we give a better chance of satisfying the condition after each iteration. We then loop continuously until the exit condition is satisfied.

The pseudo-code of SALDAE is provided in Algorithm 1 in the appendix, where we also demonstrate that our algorithm has the potential to find the optimal solution if given sufficient time and is able to produce solutions at any point during its execution. Furthermore, the algorithm has desirable properties such as no redundancy in storage and computation, which contribute to its accuracy and speed.

6 Empirical Evaluation

We experimentally compare the SALDAE algorithm against representative state-of-the-art CSG algorithms for small and large-scale problems. We compare the solution quality (for small-scale problems) and the gain rate (Taguelmimt et al. 2021) (for large-scale problems). We ran the

PICS (Taguelmimt et al. 2022) algorithm with a number of processes set to 20 and we ran CSG-UCT (Wu and Ramchurn 2020) with a number of iterations set to 10^2 , as suggested by the authors of the papers. We implemented our algorithm in Java and in the comparisons we used the codes provided by the authors of PICS and CSG-UCT, which are also written in Java. The algorithms were run on an Intel Xeon 2.30GHz E5-2650 CPU with 256GB of RAM.

To generate the problem instances, we considered the following value distributions: Agent-based Uniform (Rahwan, Michalak, and Jennings 2012), Agent-based Normal, Beta, Exponential, Gamma (Michalak et al. 2016), Normal (Rahwan et al. 2007), Uniform (Larson and Sandholm 2000), Pascal and Zipf (Changder et al. 2020). The result for each value distribution was produced by computing the average result from 50 generated problem instances per value distribution. The best strategies selection for SALDAE and the hyperparameters are explained in the appendix.

6.1 Small-Scale Benchmarks

In this subsection, we investigate how our algorithm compares to the state-of-the-art algorithms in solving small-scale problems with small numbers of agents. We run the algorithms on the nine value distributions and computed the solution quality achieved by the algorithms. Note that the algorithms behave differently depending on the value distributions. The solutions obtained by the algorithms are compared to the solutions provided by ODP-IP (Michalak et al. 2016), which always yields the optimal solutions. Figure 4 shows the results. In these experiments, we set the number of search agents of SALDAE to 10 and we stopped the algorithms when they finish or at the time when ODP-IP finds the optimal solution in case they take more time to finish than ODP-IP. Moreover, the number of cores used for each algorithm was matched to the number of processes it utilized. Specifically, SALDAE was run on 10 cores, PICS on 20 cores, and CSG-UCT on 1 core. To ensure fairness in the comparison with CSG-UCT, we added results in the appendix of SALDAE using only one search agent and one core, demonstrating that even with limited resources, SALDAE still outperforms the other algorithm.

Figure 4 clearly shows that our algorithm provides higher quality solutions than the other algorithms, which demonstrates its effectiveness. For example, with the Exponential distribution, SALDAE produces up to 28% higher solution quality than CSG-UCT and up to 15% higher solution quality than PICS. A notable exception is the Agent-based Normal, for which there is almost a tie (see the appendix).

In all value distributions, our algorithm provides optimal solutions more frequently than the other algorithms. For example with the Exponential distribution, SALDAE provides optimal solutions in 91% of the cases. With the same instances of Exponential, PICS and CSG-UCT produce the optimal solution in 7% and 5% of the Exponential instances, respectively (see Figure 5). A more detailed experimental results are shown in the appendix. In summary, our algorithm consistently produces higher quality solutions than the PICS and CSG-UCT algorithms, while at the same time providing optimal solutions more frequently than the other algorithms.

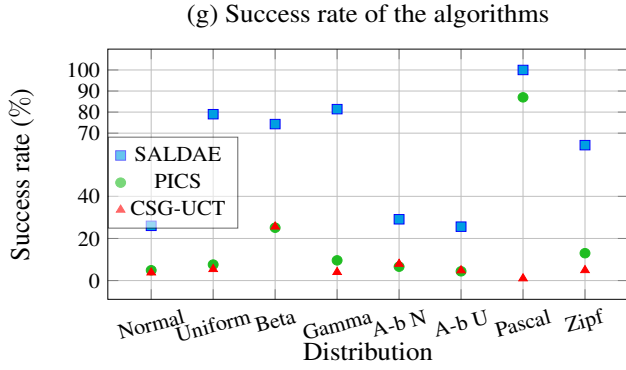


Figure 5: Success rate of the SALDAE, PICS and CSG-UCT algorithms on 2000 executions per distribution.

6.2 Large-Scale Benchmarks

The fastest optimal algorithm, ODP-IP, only scales to 30 - 40 agents. Our algorithm is able to handle large problems with hundreds and thousands of agents. The algorithms PICS and CSG-UCT could also handle large-scale problems. However, for these settings, it is infeasible to guarantee to find an optimal solution due to the exponentiality of the solution space. Hence, we cannot compute the solution quality by comparing the solutions obtained to the optimal solutions. This is why we compare the algorithms using the gain rate, which measures the improvement of the solution achieved by the algorithms relative to the value of the singleton coalition structure. The *gain rate* is computed as

$$\frac{v(\mathcal{CS})}{v(\mathcal{CS}_s)} \cdot \frac{1}{\max_i \left(\frac{v(\mathcal{CS}_i^+)}{v(\mathcal{CS}_s)} \right)},$$

where $i \in \{\text{SALDAE, PICS, CSG-UCT}\}$ and $v(\mathcal{CS}_s)$ is the value of the singleton coalition structure, which represents a partition into n coalitions, each containing a single agent, and $v(\mathcal{CS}_i^+)$ is the value of the best solutions provided by the algorithm i . We also considered a Disaster Response distribution introduced in (Wu and Ramchurn 2020), in which hundreds of human responders must be quickly organized into teams to coordinate their evacuation and rescue actions.

Figure 6 shows the results of our large-scale benchmarks. We compared SALDAE to PICS, FACS (Taguelmimt et al. 2021), and CSG-UCT. The result of each experiment was produced by evaluating all the algorithms on instances of the different value distributions. To make sure that the algorithms competed on a similar search time, we used the same time limit for all the algorithms. First, we can see a clear general trend that SALDAE and PICS outperform the other algorithms. A notable exception is the Pascal distribution, for which there is almost a tie for less than 100 agents. We can also see that the SALDAE algorithm outperforms the PICS algorithm in a majority of the tests, most notably on problem instances of the Gamma, Exponential, Normal, Disaster Response, and Electric Vehicles Allocation distributions. However, PICS performed on par in Beta and Uniform distributions and in some tests of Agent-based Normal distributions (see the appendix). SALDAE’s superior performance comes from the different heuristics that guide the

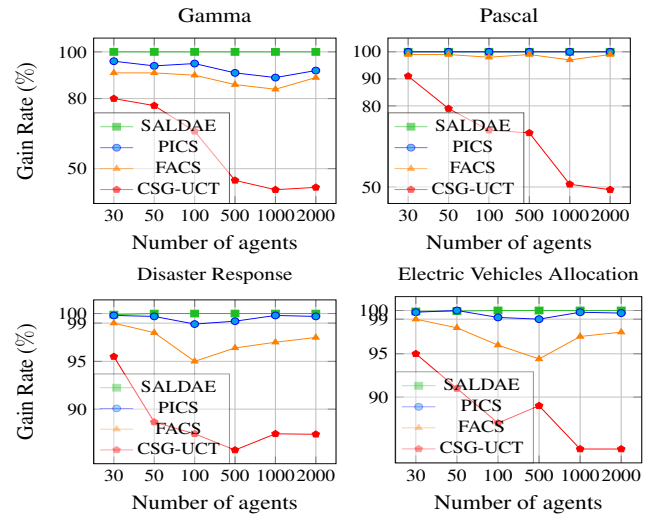


Figure 6: Gain rate of SALDAE, PICS, FACS and CSG-UCT when run with large numbers of agents.

search. For example, the bridging path strategies allow it to search other areas of the solution space and find better solutions quicker. Pathfinding between best solutions aims to explore promising regions of the search space. It provides a form of strategic neighborhood search and diversification to the greedy algorithm’s approach. This helps SALDAE explore better solutions faster than alternative methods.

7 Conclusion

We presented a multiagent path search inspired algorithm for coalition structure generation. In more detail, we developed an algorithm that utilizes multiple search agents to incrementally explore a search graph using various heuristics to guide the search process. Furthermore, we introduced different strategies for connecting the best solutions in order to improve upon them, as well as strategies for resolving conflicts between the search agents. The resulting algorithm is anytime and can handle large-scale problems. We ran experiments on a variety of different value distributions and our results demonstrate that our algorithm can perform better than existing well-established state-of-the-art algorithms in solving the coalition structure generation problem, often achieving significantly higher solution quality and gain rate.

Future work could explore the adaptation and application of other MAPF algorithms to address this problem. Another potential avenue for applying MAPF in CSG involves using a different approach where search agents not only have access to start nodes, as in the current version, but also to goal nodes. Currently, we lack a clear definition of goal nodes beyond the one representing the optimal solution, and we are unaware of its position in the graph. However, we have the flexibility to define additional goal nodes, either randomly or through a predefined function, and then the algorithm can be used to find paths between these start and goal nodes.

Acknowledgments

Tuomas Sandholm's research is supported by the Vannevar Bush Faculty Fellowship ONR N00014-23-1-2876, National Science Foundation grant RI-2312342, ARO award W911NF2210266, and NIH award A240108S001.

References

- Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Sub-optimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In *Symposium on Combinatorial Search*.
- Changder, N.; Akinine, S.; Ramchurn, S. D.; and Dutta, A. 2020. ODSS: Efficient Hybridization for Optimal Coalition Structure Generation. In *Proc. of AAI*, 7079–7086.
- Changder, N.; Akinine, S.; Ramchurn, S. D.; and Dutta, A. 2021. BOSS: A Bi-directional Search Technique for Optimal Coalition Structure Generation with Minimal Overlapping (Student Abstract). In *Proc. of AAI*, volume 35, 15765–15766.
- Dang, V. D.; and Jennings, N. R. 2004. Generating coalition structures with finite bound from the optimal guarantees. In *Proc. of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 564–571. IEEE Computer Society.
- Di Mauro, N.; Basile, T. M.; Ferilli, S.; and Esposito, F. 2010. Coalition structure generation with grasp. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, 111–120. Springer.
- Farinelli, A.; Bicego, M.; Ramchurn, S.; and Zuchelli, M. 2013. C-link: A hierarchical clustering approach to large-scale near-optimal coalition formation. In *Proc. of IJCAI*, 407–413.
- Gange, G.; Harabor, D.; and Stuckey, P. J. 2021. Lazy CBS: Implicit Conflict-Based Search Using Lazy Clause Generation. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1): 155–162.
- Keinänen, H. 2009. Simulated annealing for multi-agent coalition formation. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*, 30–39. Springer.
- Larson, K. S.; and Sandholm, T. W. 2000. Anytime coalition structure generation: an average case study. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1): 23–42.
- Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020a. New Techniques for Pairwise Symmetry Breaking in Multi-Agent Path Finding. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1): 193–201.
- Li, J.; Ruml, W.; and Koenig, S. 2020. EECBS: A Bounded-Suboptimal Search for Multi-Agent Path Finding. *ArXiv*, abs/2010.01367.
- Li, J.; Ruml, W.; and Koenig, S. 2021. EECBS: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAI Conference on Artificial Intelligence*, volume 35, 12353–12362.
- Li, J.; Sun, K.; Ma, H.; Felner, A.; Kumar, T.; and Koenig, S. 2020b. Moving agents in formation in congested environments. In *Proceedings of the International Symposium on Combinatorial Search*, volume 11, 131–132.
- Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. K. S.; and Koenig, S. 2021. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses. *Proceedings of the AAI Conference on Artificial Intelligence*, 35(13): 11272–11281.
- Li, J.; Zhang, H.; Gong, M.; Liang, Z.; Liu, W.; Tong, Z.; Yi, L.; Morris, R.; Pasareanu, C. S.; and Koenig, S. 2019. Scheduling and Airport Taxiway Path Planning Under Uncertainty.
- Ma, H.; Li, J.; Kumar, T. K. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Adaptive Agents and Multi-Agent Systems*.
- Michalak, T.; Rahwan, T.; Elkind, E.; Wooldridge, M.; and Jennings, N. R. 2016. A hybrid exact algorithm for complete set partitioning. *Artificial Intelligence*, 230: 14–50.
- Morris, R.; Pasareanu, C. S.; Luckow, K. S.; Malik, W. A.; Ma, H.; Kumar, T. K. S.; and Koenig, S. 2016. Planning, Scheduling and Monitoring for Airport Surface Operations. In *AAAI Workshop: Planning for Hybrid Systems*.
- Rahwan, T.; and Jennings, N. R. 2008. An improved dynamic programming algorithm for coalition structure generation. In *Proc. of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems.
- Rahwan, T.; Michalak, T.; and Jennings, N. R. 2012. A hybrid algorithm for coalition structure generation. In *Proc. of AAI*, 1443–1449.
- Rahwan, T.; Ramchurn, S. D.; Dang, V. D.; and Jennings, N. R. 2007. Near-Optimal Anytime Coalition Structure Generation. In *Proc. of IJCAI*, volume 7, 2365–2371.
- Rahwan, T.; Ramchurn, S. D.; Jennings, N. R.; and Giovannucci, A. 2009. An anytime algorithm for optimal coalition structure generation. *Journal of artificial intelligence research*, 34: 521–567.
- Sandholm, T.; Larson, K.; Andersson, M.; Shehory, O.; and Tohmé, F. 1999. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1): 209–238.
- Sandholm, T.; and Lesser, V. R. 1997. Coalitions among computationally bounded agents. *Artificial intelligence*, 94(1): 99–138.
- Sen, S.; and Dutta, P. S. 2000. Searching for optimal coalition structures. In *Proceedings Fourth International Conference on MultiAgent Systems*, 287–292. IEEE.
- Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219: 40–66.
- Stern, R. 2019. Multi-agent path finding—an overview. *Artificial Intelligence: 5th RAAI Summer School, Dolgoprudny, Russia, July 4–7, 2019, Tutorial Lectures*, 96–115.
- Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and

benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Taguelmimt, R.; Aknine, S.; Boukredera, D.; and Changder, N. 2021. Code-based Algorithm for Coalition Structure Generation. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 1075–1082.

Taguelmimt, R.; Aknine, S.; Boukredera, D.; and Changder, N. 2022. PICS: Parallel Index-based Search Algorithm for Coalition Structure Generation. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, 739–746.

Taguelmimt, R.; Aknine, S.; Boukredera, D.; Changder, N.; and Sandholm, T. 2023. Optimal Anytime Coalition Structure Generation Utilizing Compact Solution Space Representation. In Elkind, E., ed., *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, 309–316. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Taguelmimt, R.; Aknine, S.; Boukredera, D.; Changder, N.; and Sandholm, T. 2024. Faster Optimal Coalition Structure Generation via Offline Coalition Selection and Graph-Based Search. In Larson, K., ed., *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, 238–248. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Wu, F.; and Ramchurn, S. D. 2020. Monte-Carlo Tree Search for Scalable Coalition Formation. In *Proc. of IJCAI*, 407–413.

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

Yu, J.; and LaValle, S. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1): 1443–1449.