An Algorithm for Simultaneous Coalition Structure Generation and Task Assignment

Fredrik Präntare, Ingemar Ragnemalm, Fredrik Heintz

Linköping University, 581 83 LINKÖPING, Sweden

Abstract. Groups of agents in multi-agent systems may have to cooperate to solve tasks efficiently, and coordinating such groups is an important problem in the field of artificial intelligence. In this paper, we consider the problem of forming disjoint coalitions and assigning them to independent tasks simultaneously, and present an anytime algorithm that efficiently solves the *simultaneous coalition structure generation and task assignment* problem. This NP-complete combinatorial optimization problem has many real-world applications, including forming cross-functional teams aimed at solving tasks. To evaluate the algorithm's performance, we extend established methods for synthetic problem set generation, and benchmark the algorithm using randomized data sets of varying distribution and complexity. Our results show that the presented algorithm efficiently finds optimal solutions, and generates high quality solutions when interrupted prior to finishing an exhaustive search. Additionally, we apply the algorithm to solve the problem of assigning agents to regions in a commercial computer-based strategy game, and empirically show that our algorithm can significantly improve the coordination and computational efficiency of agents in a real-time multi-agent system.

Keywords: coalition formation, task allocation, multi-agent system, artificial intelligence, optimal assignment

1 Introduction

An important research challenge in the domain of artificial intelligence and multi-agent systems is to solve the problem of how to organize and coordinate agents to improve their efficiency and capabilities when solving problems. Many partial solutions to this problem have already been suggested, including methods for *task allocation*, and algorithms based on the formation of organizations (e.g. coalitions, teams, hierarchies) [2, 9, 12]. For example, *coalition formation* is a technique that has been used to enable cooperation among agents in multi-agent environments by forming coalitions of agents. This technique involves evaluating different coalition structures, and forming the coalitions in the coalition structure that has the highest performance measure (utility value). The formed coalitions may then be used to perform tasks that require several agents to be accomplished efficiently. Optimal *coalition structure generation* is NP-complete, and

many algorithms have been presented that solves this problem, including algorithms based on dynamic programming, evolutionary approaches, and branch-and-bound [6–8, 13].

The *optimal assignment* problem is an important optimization problem in which the goal is to assign workers to tasks to maximize the overall performance measure [3]. In certain settings this problem can be solved in polynomial time (e.g. using the Hungarian algorithm) [4].

In this paper, we consider the simultaneous (or combined) coalition structure generation and task assignment problem. This problem can be solved by first forming coalitions, and then assigning them to tasks. However, this approach may generate suboptimal solutions — even if the coalition structure generation and task allocation algorithms in themselves are optimal, since the generated coalitions may not be the best coalitions for the tasks at hand. The reason for this is that, during the generation of coalition structures, the performance measure of a coalition is given by its members, and not by the task that the coalition is potentially assigned to. Perhaps even worse is the consequence that any generated solution could potentially be arbitrarily worse than the optimal solutions. Additionally, this approach would generally require two different utility functions: one for each of the two subsequent steps, since coalition structure generation algorithms do not consider the tasks that each coalition is to be assigned to. This is disadvantageous, since it may not be a simple task to create good utility functions (or to generate realistic performance measures), and it could potentially be hard to predict how the two utility functions affect the quality of the generated solutions.

To address these issues, we present an efficient anytime algorithm that integrates task assignment into the formation of coalitions. We accomplish this by generating coalition structures where each coalition is assigned to exactly one task. Our algorithm can thus be used to create structured collaboration in multi-agent systems by utilizing task allocation. Furthermore, our algorithm only requires one utility function, has the ability to prune large subspaces of the search space, can give worst-case guarantees on generated solutions, and always generates optimal solutions when run to exhaustion.

To evaluate the algorithm's performance, we extend established methods for synthetic problem set generation provided by Sandholm and Larson [5], and benchmark our algorithm against simple brute-force and branch-and-bound implementations, since there are no algorithms that solves the problem under our assumptions that we can compare to. Such experiments can be replicated by anyone, and are conducted to deduce whether the presented algorithm can handle difficult data sets efficiently. Additionally, we apply our algorithm to solve the problem of assigning groups of agents to regions in the commercial strategy game *Europa Universalis 4*, and empirically show that our algorithm can be used to optimally solve real-world simultaneous coalition structure generation and task assignment problems efficiently. Apart from solving problems that exist in strategy games, our algorithm can potentially be used to solve many important real-world problems. It could, for example, be used to form optimal

cross-functional teams aimed at solving a set of problems, to assist in the organization and coordination of subsystems in an artificial entity, or to allocate tasks in multi-robot systems.

We begin by formalizing the simultaneous coalition structure generation and task assignment problem in Section 2. Then, in Section 3, we give a presentation of our algorithm. In Section 4, we evaluate our algorithm, and present results from our experiments. In Section 5, we conclude with a summary of our results.

2 Problem Formalization

The simultaneous coalition structure generation and task assignment problem is formalized as:

Input: A set of agents $A = \{a_1, ..., a_n\}$, a set of tasks $T = \{t_1, ..., t_m\}$, and the performance measure $v(C, t)$ for assigning a coalition $C \subseteq A$ to a task $t \in T$.

Output: A set of coalitions $\{C_1, ..., C_m\}$ that maximizes the sum $\sum_{i=1}^{m} v(C_i, t_i)$, such that $C_i \subseteq A$, $C_i \cap C_j = \emptyset$ for all $i \neq j$, and $\bigcup_{i=1}^{m} C_i = A$.

3 Algorithm Description

To solve this problem, we propose an anytime algorithm based on branch-and-bound, a novel representation of the search space, and a guided sequential search for solutions. By using branch-and-bound, our algorithm can generate both optimal solutions, and high-quality anytime solutions with worst-case guarantees. The algorithm consists of the following three steps:

I. **Partitioning of the search space.**
To discard unnecessary parts of the search space that only contain suboptimal solutions, we first partition the search space into disjoint subspaces.

II. **Calculation of the upper and lower bounds for the partitions.**
We cannot know whether a subspace can be discarded if we don't have a way to deduce whether the best possible solution in that subspace can be discarded (if we want to be able to guarantee the optimality of our solutions).

III. **Searching for the optimal solution.**
We search for the best solution by sequentially searching the partitions, and discarding unnecessary suboptimal subspaces using branch-and-bound.

3.1 Partitioning of the Search Space

Before we describe how our partitioning scheme works, note that an integer partition of $k \in \mathbb{N}$ is a way of writing k as a sum of positive integers [1]. Now, given a set of agents $A = \{a_1, ..., a_n\}$, and a set of tasks $T = \{t_1, ..., t_m\}$, we use the following three steps to partition the search space:

1. First, generate sets from all of the possible distinct integer partitions of the number $|A| = n$ that has $|T| = m$ or fewer addends. For example, if we have that $|A| = 4$ and $|T| = 3$, we generate $\{4\}, \{3, 1\}, \{2, 2\}$ and $\{2, 1, 1\}$.

2. Insert zeros to the sets that we generated during *step 1* until they have as many members as there are tasks. For example, given the sets from the example in *step 1*, we generate $\{4,0,0\}, \{3,1,0\}, \{2,2,0\}$, and $\{2,1,1\}$.
3. Let each possible multiset permutation of each of the sets generated during *step 2* represent a partition (subspace) of the search space by letting each number represent a task. For example, the permutation $\langle 4,0,0\rangle$ corresponds to assigning 4 agents to t_1, 0 agents to t_2, and 0 agents to t_3, while $\langle 0,4,0\rangle$ corresponds to assigning 0 agents to t_1, 4 agents to t_2, and 0 agents to t_3.

The multiset permutations in *step 3* can efficiently be generated using the algorithm based on tree-traversal proposed by Takaoka [10], or the algorithm based on loopless generation proposed by Williams [11].

The reason to why the generated partitions cover the whole search space is the fact that any coalition structure with n agents can be directly mapped to one of the possible distinct integer partitions of the integer n (for proof, see [8]). For instance, $\{\{a_i, a_j\}, \{a_k\}\}$ can be mapped to $\{2,1\}$, and $\{\{a_i, a_j, a_k\}\}$ to $\{3\}$. In *step 1*, we generate the partitions that correspond to these mappings. We then remove unnecessary coalition structures in *step 2*, so that we only look at coalition structures that can represent valid solutions. Finally, in *step 3*, we refine the representation of the search space generated by *step 2*, by taking advantage of the fact that we are only interested in bijections of coalitions to tasks.

3.2 Calculation of the Upper and Lower Bounds for Partitions

To calculate the bounds for partitions, let $\mathbb{A}_p = (X \subseteq A : |X| = p)$, and define:

- $M(p,t) = \max\{v(C,t) : C \in \mathbb{A}_p\}$
- $Avg(p,t) = \frac{1}{|\mathbb{A}_p|} \sum \{v(C,t) : C \in \mathbb{A}_p\}$

Now, given a multiset permutation $P = \langle p_1, ..., p_m\rangle$ that represents a partition (subspace) of the search space, we can calculate an upper bound U_P for the partition that corresponds to P as the sum $U_P = \sum_{i=1}^{m} M(p_i, t_i)$. This is a valid upper bound for the partition that corresponds to P, since given a set of tasks $T = \{t_1, ..., t_m\}$, and any possible solution $S_P = \{C_1, ..., C_m\}$ induced by P with the performance measure $V(S_P) = \sum_{i=1}^{m} v(C_i, t_i)$, then $v(C_i, t_i) \leq M(|C_i|, t_i)$, of which $V(S_P) \leq U_P$ follows.

Similarly, we can calculate a lower bound L_P for the partition that corresponds to P as the sum $L_P = \sum_{i=1}^{m} Avg(p_i, t_i)$, with the intuition that a solution that has a value that is as good as the arithmetic mean of the solutions induced by P is always worse than or equal to the optimal solution induced by P.

3.3 Searching for the Optimal Solution

To search for the optimal solution, we expand one partition at a time, and base the precedence order for expanding partitions on the upper bound of the partitions: $U_{P_i} > U_{P_j} \implies P_i \prec P_j$, where $P_i \prec P_j$ denotes that partition P_i

should be expanded before partition P_j. If two partitions have the same upper bound, we use a second ordering criterion based on the lower bound of the partitions: $U_{P_i} = U_{P_j}$ and $L_{P_i} > L_{P_j} \implies P_i \prec P_j$.

Now, given this order of precedence for the expansion of partitions, we sequentially search through each expanded partition using branch-and-bound. When a partition has an upper bound that is lower than or equal to the value of the best solution that we have found so far, simply discard the entire partition and terminate the search. Due to using our order of precedence, it is possible to terminate the search and still guarantee optimality.

To address the high memory requirements for generating and storing many multiset permutations (required for generating the precedence order), we can generate and store multiset permutations into blocks. These blocks can sequentially be generated and searched during partitioning. The more blocks we use, the less memory is required. In our case, we use each set generated in *step 2* during the partitioning phase to represent a block. In other words, each possible group of multiset permutations that has the same members is searched in sequence according to the aforementioned order of precedence.

4 Evaluation

A common approach to evaluating the performance of search algorithms is to use standardized problem instances for benchmarking. In the case of simultaneous coalition structure generation and task assignment, no such standardized problem instances exist. Therefore, we look at standardized problem instances from a similar domain. More specifically, we translate standardized problem instances used for benchmarking coalition structure generation algorithms to the domain of simultaneous coalition structure generation and task assignment.

Larson and Sandholm [5] provided standardized synthetic problem sets for the optimal coalition structure generation problem by using normal and uniform probability distributions to provide randomized coalition (utility) values. Following Rahwan et al. [8], we denote these NPD and UPD, respectively. Since our algorithm's performance depends on its ability to discard suboptimal subspaces of the search space, it is important that we benchmark it using problem sets with different characteristics. As such, we suggest using NPD and UPD for benchmarking our algorithm. In addition to NPD and UPD, we also use NDCS, a probability distribution that was proposed by Rahwan et al. [8] for benchmarking coalition structure generation algorithms, since both NPD and UPD generate biased results. Translating these probability distributions to our domain is simple, and the translations are presented below, where $v(C, t)$ denotes the performance measure of assigning a coalition C to a task t:

- **NPD:** $v(C, t) \sim |C| \times \mathcal{N}(\mu, \sigma^2)$, where $\sigma = 0.1$ and $\mu = 1$.
- **NDCS:** $v(C, t) \sim \mathcal{N}(\mu, \sigma^2)$, where $\sigma = \sqrt{|C|}$ and $\mu = |C|$.
- **UPD:** $v(C, t) \sim |C| \times \mathcal{U}(a, b)$, where $a = 0$ and $b = 1$.

Furthermore, the result of each experiment is produced by calculating the average of the resulting values (e.g. time measures or utility values) from 100 gener-

ated problem sets per probability distribution and experiment. We deem this to be sufficient to give a clear indication of the behavior of the algorithm. Finally, we compare our algorithm to simple brute-force and branch-and-bound implementations, since there are no existing algorithms that solves the problem under our assumptions that we can compare to. However, this is not possible when there are many agents and tasks, since simple algorithms based on brute-force and branch-and-bound are too slow.

4.1 Implementation and Equipment

The algorithms were implemented in C++11 using the C++ standard library. All probability distributions were generated using the random number distribution generator `std::random::normal_distribution<double>` for NDCS and NPD, and `std::random::uniform_real_distribution<double>` for UPD. The tests were conducted using a computer with Windows 10 (x64), an Intel 7700K 4200MHz CPU, and 16GB of DDR4 memory (3000MHz CL15).

4.2 Results

The execution time to find an optimal solution for the fixed number of 8 tasks is plotted using a logarithmic scale in *Figure 1*. Plots of the search times for the plain branch-and-bound (denoted **pBNB**) and brute-force algorithms are used as a comparison to the presented algorithm (denoted **iBNB**). In *Figure 2*, we fix the number of agents to 10, and look at how the number of tasks affect performance. Finally, in *Figure 3*, we look at the quality of the anytime solutions generated by our algorithm. We used 12 agents and 8 tasks for this purpose, and interrupted the algorithm during search by only allowing it to evaluate a fixed number of solutions. The total number of solutions for 12 agents and 8 tasks is $8^{12} \approx 7 \times 10^{10}$. On the y-axis, we show the performance measure (utility value) U of the solutions that our algorithm had found on interruption, divided by the value U^*_{opt} of an optimal solution.

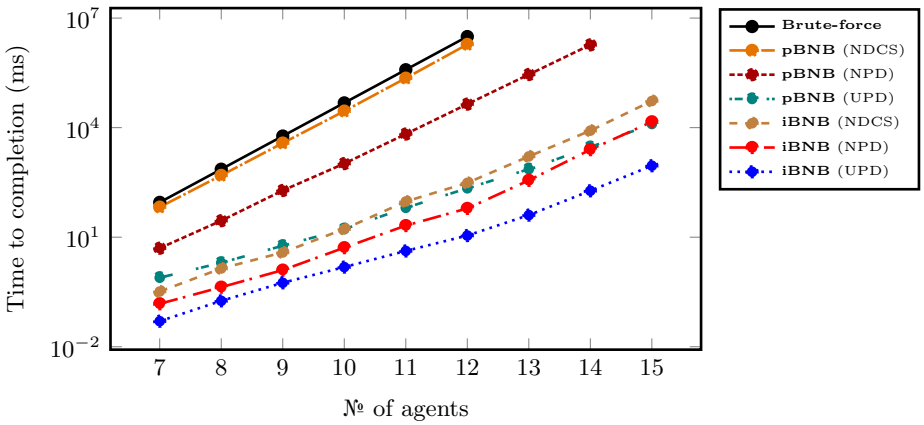

Fig. 1. The execution time to find an optimal solution in problems with 8 tasks.

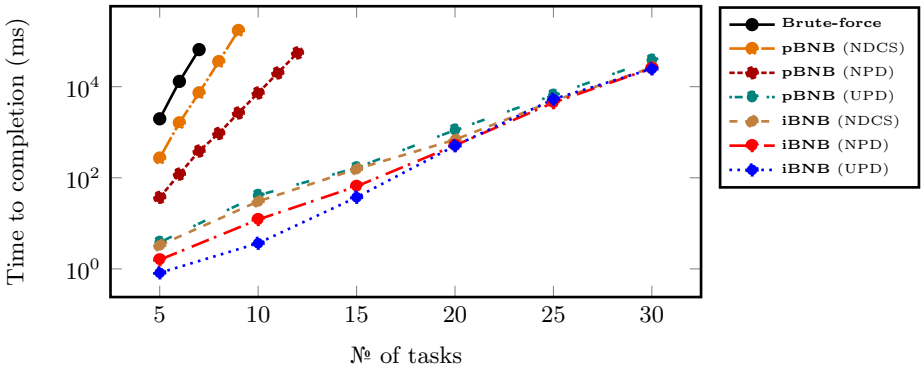

Fig. 2. The execution time to find an optimal solution in problems with 10 agents.

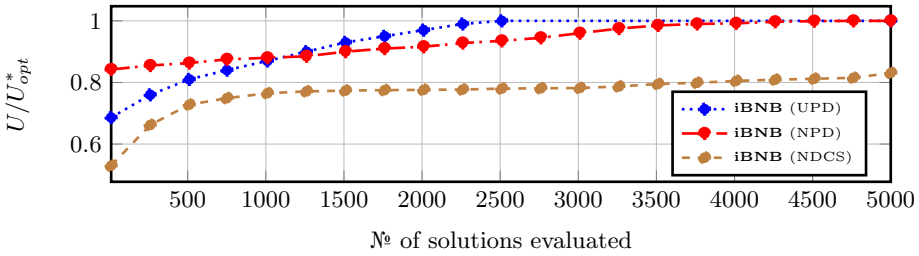

Fig. 3. The quality of anytime solutions when the algorithm is interrupted prior to finishing an exhaustive search when solving synthetic problem sets with $8^{12} \approx 7 \times 10^{10}$ possible solutions.

4.3 Applying the Algorithm to Europa Universalis 4

The algorithm was also applied to Europa Universalis 4 — a commercial strategy game in which agents are required to act and reason in real-time. We used the presented algorithm to solve the problem of assigning agents to regions. By doing so, it was possible to test our algorithm in a real-world multi-agent system with high computational requirements, and compare its performance to a Monte Carlo algorithm that was specifically designed (and previously used) to solve a particular instance of the simultaneous coalition structure generation and task assignment problem.

The problem sets that were generated by Europa Universalis 4 each consisted of up to 8 agents and 35 regions (tasks). As such, the problem sets were rather small, with $|A| \in [2, 8]$ and $|T| \in [2, 35]$. In total, 13922 problem sets were generated, each reflecting a problem instance from the game. With this in mind, our algorithm managed to increase both the quality of the solutions, and the performance of the computer-based players, compared to using the specialized Monte Carlo algorithm that was developed and designed by the developers of the game. The quality of the solutions was increased by, on average, 565%, and the search time for the best possible solution was improved by, on average, 422%.

4.4 Discussion

As empirically shown, our algorithm is considerably faster (by many orders of magnitude) than both brute-force and plain branch-and-bound — for all problem sets and distributions. The reason is that our algorithm discards huge portions of the search space, and almost always terminates searches before it generates unnecessary solutions, even when solving extremely difficult problem sets (NDCS). For the same reasons, and due to our order of precedence for expanding partitions, it doesn't take our algorithm many evaluations before it finds close-to-optimal solutions for any of the three probability distributions.

The presented algorithm solves problem sets generated by UPD the fastest, followed by the sets generated by NPD, and then NDCS. This is not surprising, since it is reasonable to expect that our algorithm exhibits performance characteristics that are similar to those exhibited by similar algorithms used for coalition structure generation. In the case of 8 tasks, 8 agents, and problem sets generated by UPD, our algorithm is, on average, roughly 2416 times better than brute-force (i.e. it takes approximately 0.041% of the time to find the optimal solution). As the number of agents increases, this factor also increases. For example, in the case of 8 tasks and 12 agents and UPD, our algorithm is, on average, 280882 times better than brute-force (i.e. it finds optimal solutions in approximately 0.00035% of the time it takes for the brute-force algorithm). As such, if we increase the number of tasks or agents, the relative gains in performance increases considerably, which is also true in comparison to plain branch-and-bound.

5 Conclusions

In this paper, we presented an anytime algorithm that efficiently solves the simultaneous coalition structure generation and task assignment problem by integrating task assignment into the formation of coalitions. To benchmark our algorithm, we extended established methods for benchmarking coalition structure generation algorithms to our domain, and then used synthetic problem sets to empirically evaluate its performance. We used brute-force and plain branch-and-bound algorithms for comparison, since we didn't find any specialized algorithms that solves the problem under the same assumptions as we do.

Our results clearly demonstrate that our algorithm is far superior to brute-force and plain branch-and-bound, and that our algorithm doesn't have to search for very long before it can find good solutions. This is beneficial in many real-time systems (e.g. real-world multi-agent systems), in which optimal solutions are not always required. Apart from these properties, our algorithm is also able to give worst-case guarantees on anytime solutions due to taking advantage of branch-and-bound. Finally, by using our algorithm to improve agent-to-region assignment in Europa Universalis 4, we demonstrated that our algorithm can be used to efficiently solve a common real-world simultaneous coalition structure generation and task assignment problem.

References

1. Andrews, G., and Eriksson, K. "Integer partitions". Cambridge University Press. (2004)
2. Bryan, H., and Lesser, V. "A survey of multi-agent organizational paradigms". In: *The Knowledge Engineering Review.* (2004)
3. Gerkey, B. P., and Matari, M. J. "A formal analysis and taxonomy of task allocation in multi-robot systems." In: *The International Journal of Robotics Research 23.9*, p. 943. (2004)
4. Kuhn, H. W. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly 2.1-2.* (1955)
5. Larson, K. S., and Sandholm, T. W. "Anytime coalition structure generation: an average case study". In: *Journal of Experimental & Theoretical Artificial Intelligence 12.1.* (2000)
6. Rahwan, T., Jennings, N. R. "An improved dynamic programming algorithm for coalition structure generation". In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3. International Foundation for Autonomous Agents and Multiagent Systems.* (2008)
7. Rahwan, T., Michalak, T. P., Wooldridge, M., Jennings, N. R. "Coalition structure generation: A survey". In: *Artificial Intelligence 229.* (2015)
8. Rahwan, T., Ramchurn S. D., Jennings, N. R., Giovannucci, A. "An anytime algorithm for optimal coalition structure generation". In: *Journal of Artificial Intelligence Research 34.* (2009)
9. Shehory, O., and Kraus, S. "Methods for task allocation via agent coalition formation". In: *Artificial intelligence.* (1998)
10. Takaoka, T. "An *O(1)* time algorithm for generating multiset permutations". In: *International Symposium on Algorithms and Computation. Springer.* (1999)
11. Williams, A. "Loopless generation of multiset permutations using a constant number of variables by prefix shifts". *In: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics.* (2009)
12. Yamada, T., and Nasu, Y. "Heuristic and exact algorithms for the simultaneous assignment problem". In: *European Journal of Operational Research 123.3.* (2000)
13. Yang, J., Luo, Z. "Coalition formation mechanism in multi-agent systems based on genetic algorithms". In: *Applied Soft Computing 7.2.* (2007)