A Best-First Anytime Algorithm for Computing Optimal Coalition Structures

(Short Paper)

Chattrakul Sombattheera
Decision Systems Lab
School of Computer Sience and Software Engineering
University of Wollongong, NSW 2500
Australia
cs50@uow.edu.au

Aditya Ghose
Decision Systems Lab
School of Computer Sience and Software Engineering
University of Wollongong, NSW 2500
Australia
aditya@uow.edu.au

ABSTRACT

This work presents a best-first anytime algorithm for computing optimal coalition structures. The approach is novel in that it generates coalition structures based on coalition values, while existing algorithms base their generation on the structure (members and configurations) of coalitions. With our algorithm, coalition structures are generated by repeatedly choosing the best coalition, as determined using a novel metric called *agent's contribution to coalition structure* that we define. We have compared the performance of our algorithm against that of Rahwan et al [5] using 20 data distributions. Our results show that our algorithm almost always converges on an optimal coalition structure faster (although it terminates later in some cases). Empirically, our algorithm almost always yields better than or as good as Rahwan et al's results at any point in time.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithm

Keywords

Multiagent Systems, Coalition Formation, Optimal Coalition Structure

1. INTRODUCTION

Computing optimal coalition structures in multi-agent systems is an important research problem both from theoretical [2] and practical perspectives. The optimal coalition structure problem seeks to identify, given a set of agents and a value to each subset, the optimal partitioning of that set of agents (i.e., a partitioning for which the sum of the coalition values is maximized). The optimal coalition structure problem finds application in a variety of real world settings, including logistics and supply chains [8, 9, 10], virtual organizations, team formation,

Formally, given a set N of n agents, a coalition is a non-empty subset, $S \subseteq N$. A coalition where $S = N$ is called the *grand coalition*. We will use $|S|$ to refer to the *cardinality* (i.e., number of agents) of a coalition S. We assume that there is a *characteristic function*, $V : N \rightarrow \mathbb{Z}^+$, which associates each S with a *coalition value*, V_S. The number of distinct coalitions that can be formed with n agents is $2^n - 1$. Formally, a *coalition structure* is a *partitioning* of the set of agents. For example, the following are all the coalition structures that can be obatined from a set of 3 agents $\{1, 2, 3\}$: $\{\{1\}, \{2\}, \{3\}\}, \{\{1, 2\}, \{3\}\}, \{\{1, 3\}, \{2\}\}, \{\{2, 3\}, \{1\}\}$, $\{\{1\}, \{2\}, \{3\}\}$. The number of all coalition structures can be determined by *Bell Number*, B_n [3]. Given a coalition structure CS, we define its value, $V(CS) = \sum_{S \in CS} V_S$, which denotes the utility that accrues to the systems from that partitioning. An optimal coalition structure is denoted by CS^* such that $CS^* = argmax_{CS \in L} V(CS)$. In general, multiple such optimal coalition structures might exist. This problem is proved to be NP-Hard [7].

This work presents a best-first anytime algorithm for computing optimal coalition structures (that we shall refer to as CH). The approach is novel in that it generates coalition structures based on coalition values, while existing algorithms base their generation on the structure (members and configurations) of coalitions. With our algorithm, coalition structures are generated by repeatedly choosing the best coalition, as determined using a novel metric called *agent's contribution to coalition structure* that we define. We have compared the performance of our algorithm against that of Rahwan et al [5] using 20 data distributions. Our results show that our algorithm almost always converges on an optimal coalition structure faster.

2. THE CH ALGORITHM

This work generates coalition structures based on coalition values rather than their coalition members. We consider generating coalition structures as a process of repeatedly choosing the best coalition (i.e., one that contributes the best value to the coalition structure) from available candidates such that for each generated coalition structure $i) \bigcup S_i =$

Cite as: A Best-First Anytime Algorithm for Computing Optimal Coalition Structures (Short Paper), Chattrakul Sombattheera and Aditya Ghose, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1425-1428.

N (the *exhaustiveness* condition [7, 4, 1]) and $ii) S_i \cap S_j = \emptyset$ for $i \neq j$ (the *disjointness* condition [6, 5]). The algorithm must be *systematic*, i.e., it must not generate/evaluate the same coalition structure more than once.

Best Coalition: Since the data (coalitions and their values) distribution varies, we need a metric that would permit us to pick the next coalition to add to an (incrementally constructed) coalition structure. We define a metric called *agent contribution to coalition*, $\overline{a}_S$, i.e., the average value for each agent in S:

$$\overline{a}_S = \frac{V_S}{|S|}$$

and use this as a basis for our best-first search. We note in passing that the *agent contribution to coalition structure*, $\overline{a}_{CS}$, i.e., the average value for each agent in CS:

$$\overline{a}_{CS} = \frac{\sum V_{S_i}}{n}$$

will always be maximal for any optimal coalition structure.

In order to be efficient in random environments, we define the best coalition, S^*, is the one whose $\overline{a}$ is the highest and . The smaller coalition is better than the larger one whose $\overline{a}$ is equal to the former.

Data Structures: We define $\mathcal{CS}$ a one-dimensional array of coalitions, whose size is $n = |N|$, the maximal number of coalitions the coalition structure may contains. Each element is for a coalition chosen so far. $\mathcal{CS}$ represents CS and will be used interchangeably. We shall refer to the index of $\mathcal{CS}$ as the coalition *layer*, l-th. Element $\mathcal{CS}[1]$ and $\mathcal{CS}[2]$ are the coalitions at the 1^{st} and the 2^{nd} layer of the coalition structure respectively. We store all coalitions in a two-dimensioned array, $\mathcal{C}$. The first dimension refers to cardinalities of coalitions. The second dimension refers to the position of coalitions of a given cardinality. Coalitions in each cardinality will be sorted by their value in descending order, i.e., the 1^{st} position is the highest value or the best coalition, the 2^{nd} position is second highest value or the second best coalition, and so on. We shall refer to the second dimension as the *order* of coalitions within each cardinality. Element $\mathcal{C}[1][1]$, for example, refers to the 1^{st} order (coalition) of cardinality 1 while $\mathcal{C}[2][1]$ refers to the 1^{st} order (coalition) of cardinality 2, and so on.

In order to maintain the *disjoint* condition, we define $\mathcal{R}$ a set of available agents from which a coalition can be chosen for generating CS. A new CS is generated once R is empty, i.e., the *exhaustive* condition is satisfied. We define $\mathcal{B}$ a two-dimensional array of integer for indexing coalitions in $\mathcal{C}$, in order to guarantee the correctness, the algorithm needs to keep track of what coalitions have been used at each layer so far. The first dimension refers to the layer of $\mathcal{CS}$. The second dimension refers to the cardinality in each layer. The value of each element of $\mathcal{B}$ indexes a coalition in $\mathcal{C}$. Such a coalition, $\mathcal{S} \subseteq \mathcal{R}$ and $\mathcal{S} \neq \emptyset$, in each cardinality is the first available order of a given cardinality in $\mathcal{C}$ that might be chosen for layer l of $\mathcal{CS}$. We shall refer to this coalition as the *candidate* coalition of its cardinality. Element $\mathcal{B}[1][1] = 1$ in Figure **??**, for example, implies that the candidate coalition from cardinality 1 for layer 1 of $\mathcal{CS}$ is $\mathcal{C}[1][\mathcal{B}[1][1]]) = \mathcal{B}[1][1] = \{1\}$, i.e., the 1^{st} coalition of cardinality 1. Element $\mathcal{B}[3][2] = 6$ implies that the candidate coalition for layer 3 from cardinality 2 is $\mathcal{C}[2][\mathcal{B}[3][2]] = \mathcal{C}[2][6] = \{3, 4\}$, i.e., the 6^{th} coalition of cardinality 2. The value 0 of an element implies there is no candidate coalition for the specified layer and cardinality. Hence, the value of elements in $\mathcal{B}$ simply tells what is the candidate coalition of each cardinality in each layer of $\mathcal{CS}$.

Algorithm 1 Construct coalition structures by adding the next available coalition into the existing structure

```
1: l ← 1
2: S ← chooseNextS(l)
3: while S! = ∅ do
4:     CS[l] ← S
5:     R ← R \ S
6:     S ← ∅
7:     if R = ∅ then                    ▷ a coalition structure is made
8:         print "***** "+CS;
9:     end if                           ▷ attempt to extend layer
10:    S ← Extend()
11:    if S != null then                ▷ Extend to the next layer
12:        l ← l + 1;
13:    else      ▷ cannot extend then attempt for altering
14:        S ← Alter()
                                        ▷ cannot alter
15:        if S = ∅ then                ▷ attempt to shrink
16:            S ← Shrink()
17:        end if
18:    end if
19: end while
```

2.1 Main Function

There are two parts in the main function: populating data and the main loop. **Populating Data:** At the beginning, $\mathcal{C}$ is populated. We follow Sandholm et. al. [7, 11] that a $1/n$ bound can be established at the cost of 2^{n-1}. Its elements in each cardinality are sorted by their values in descending order. Sorting coalitions in each cardinality can be done in parallel using any efficient sort algorithm. In our implementation, we use *Merge* sort algorithm, which is very robust and efficient because its worst case time complexity is among the best, i.e., $m \log m$ where m is the size of input. Since sorting is done in parallel, the largest value of m is ${}^nC_{\lfloor n/2 \rfloor}$ which is slightly less than $2^n - 1$, the time complexity of Rahwan et. al. The space complexity is also reasonable that it is $O(m)$. (One may argue that sorting can be costly to the performance of the algorithm. The empirical results show that, taking into account the sorting time, our algorithm still converges much quicker in all data distributions.) All the coalitions in $\mathcal{C}$ are sorted by their values rather than their members as in Rahwan et. al [6, 5]. The first coalition of each cardinality is the best, i.e., highest value, and so on. Then, both $\mathcal{B}$ and $\mathcal{CS}$ are initialized with 0 and *null* for all elements respectively. Then, l is set to 1 indicating that the coalition structure is being built at layer 1 as the starting layer. At each layer l, the algorithm will determine what are the candidate coalitions. At the beginning, the first coalition in each cardinality is its candidate. Thus, $\mathcal{B}[1][j]$ is set to 1 indicating that any first coalition can be chosen for layer 1 of $\mathcal{CS}$. $\mathcal{R}$ is set to N since a coalition is yet to be used in $\mathcal{CS}$. Just before the main loop, the first coalition $\mathcal{S}$ is chosen by function $ChooseNextS()$. This function is described below.

Main Loop: The logic of the main loop is very simple. It determines if there is anymore candidate coalition to be placed in $\mathcal{CS}$ at the present layer l. Since the first coalition $\mathcal{S}$ has just been chosen, the algorithm enters the main loop in

this first round and places $\mathcal{S}$ at layer $l = 1$ of $\mathcal{CS}$. Remainder agents $\mathcal{R}$ is subtracted by S because the agents who are the members of $\mathcal{S}$ cannot be part of the next coalition. This will guarantee the disjointness of the coalition structure [6, 5]. $\mathcal{S}$ is then reset to null. After that the algorithm determines whether a new coalition structure has been generated by examining if agents are exhaustively used in $\mathcal{CS}$, i.e., to examine whether $\mathcal{R}$ is empty. If that is the case, Then the algorithm outputs the newly generated CS.

The algorithm continues acquiring the next coalition to fill in $\mathcal{CS}$. To satisfy the two conditions mentioned above, the algorithm may do one of the followings in order to accommodate the new coalition into CS: i)*Extend* the body by calling $Extend()$, ii)*Alter* the body by calling $Alter()$, or iii)*Shrink* the body by calling *Shrink*. The algorithm goes through these steps, in which the value of $\mathcal{S}$ as well as the next layer l at which the new coalition can be inserted into $\mathcal{CS}$ will be determined, and reaches the end of the loop . It goes back to the start of the loop again where it examines the value of $\mathcal{S}$ whether it will continue in the loop. The algorithm terminates when it cannot find any more coalition. The 5 working functions that are used in the main algorithm will be discussed below.

2.2 Working Functions

The following are working functions used in the algorithm. **Choosing Next Coalition (CHOOSENEXTS):** Firstly, this function is called to determined the first best coalition before entering the main loop as well as in extending, altering and shrinking CS. The search for the best coalition to fill in $\mathcal{CS}$ is very simple. The algorithm goes through each candidate coalition at layer l by cardinalities in ascending order nd chooses the highest $\overline{a}_S$. In case there is no candidate found, it returns 0.

Extending Coalition Structures: After inserting a new coalition into $\mathcal{CS}$, the algorithm examines whether there will be a new coalition which will extend, or be inserted into layer $l+1$ of, $\mathcal{CS}$. The present $\mathcal{CS}$ might be extended if $l < n$ and for each valid cardinality i, $i \leq |\mathcal{R}|$. Firstly, algorithm search for candidates in cardinalities i at the next level, $l+1$ by calling function $NextS$. The returned value will be assigned to $\mathcal{B}[i][l+1]$ as the candidate coalition. Note that the return value might be 0, indicating that there is no more candidate in this cardinality. The algorithm then chooses the next coalition $\mathcal{S}$ by calling the function $chooseNextS()$, which will return the best candidate at the next layer. The best candidate will be inserted into $mathcalCS$ in the next round. If $\mathcal{S}$ is empty, the algorithm tries the followings functions.

Altering the Last Coalition: Since it cannot extends $\mathcal{CS}$, the algorithm tries altering the body by replacing its last coalition at $\mathcal{CS}[l]$ with the next best coalition from available candidates at that level. Before it can actually do that, it has to return the member of $\mathcal{CS}[l]$ back to $\mathcal{R}$. Then it searches for the next candidate of the the corresponding cardinality by calling function $nextS()$. The next coalition $\mathcal{S}$ will be acquired by calling function $chooseNextS(l)$. If $\mathcal{S}$ is found, the execution reaches the end of the loop and continues in the next round. Otherwise, it attempts shrinking the body.

Shrinking Coalition Structures: At this point, algorithm cannot find any coalition to extend $\mathcal{CS}$ to the next layer $l+1$ nor any coalition to alter $\mathcal{CS}[l]$. It then shrinks CS and tries if coalition structure can still be generated. This can be done only if the condition $l > 1$ holds. Firstly, it decreases the value of l by 1. The coalition $\mathcal{CS}[l]$ will be replaced by its next available candidate by calling $NextS$. The next coalition S will be acquired by calling function $chooseNextS(l)$. If S is found, the execution reaches the end of the loop and continues in the next round. Otherwise, it will attempt to shrink the body again.

Search for Next Candidate Coalition (NEXTS): After acquiring a new coalition S at any layer l, the algorithm prepares the next candidate coalition in its corresponding cardinality by searching downwards. The next available coalition is the first one, whose members are all in $\mathcal{R}$. The value of each of these candidate coalitions is the *upperbound* of its cardinality and can be used to determine the upperbound of the optimality in order to decide whether the algorithm should terminate. The search might not find a valid candidate. Thus the respective element of $\mathcal{B}$ is assigned the value 0. Hence, the search is needed only if there is a chance to find a candidate coalition, i.e., the value of the respective $\mathcal{B}$ is greater than 0.

2.3 Correctness

Guarantee of Correctness and Systematicity: This algorithm guarantees the correctness condition by the collaboration of these extending, altering and shrinking actions. At the beginning, the algorithm will keep extending CS. At layer $l = 1$, all coalitions are candidates. The number of candidate coalitions across cardinalities in each layer drops while the algorithm proceeds to further levels. This continues until the first coalition structure is generated and the algorithm cannot extend the structure anymore. Hence the algorithm tries altering its last coalition. It discards the last coalition of the coalition structure and search for its substitute as the candidate coalition. Then one of the candidate coalitions in that layer then could be chosen as the new coalition from which the algorithm can further extend the coalition structure.

This cycle of extend and alter repeats until there is no candidate coalitions left in that layer. Once there is no candidate coalition left in each layer, the algorithm shrinks CS by 1 layer. The last coalition in CS will, again, be discarded. The algorithm substitutes it with its valid successor from the same cardinality. The algorithm chooses one of the candidate coalitions in this layer from which the algorithm may extend CS. In the case that there is no candidate coalition in the last layer after shrinking, then the algorithm repeats the shrinking until it can either extend CS or alter the last coalition. If the shrinking repeats until it reaches layer 1, the search for candidate coalitions downward will push the upperbound down as well.

Two coalition structures are identical if they are composed of the same set of coalitions. The repeating coalition structure might have coalitions generated i) in the same order, or ii) in different order of its predecessor. Our algorithm generates coalition structures in a certain order thus guaranteeing systematic search. Since the next coalition is to be chosen from candidate coalitions by a certain criterion, a coalition S_i that is strictly better than S_j will always be chosen before S_j regardless of the layer l and other candidate coalitions. This prevents repetition in different order. Our algorithm easily prevents the first type repetition by its method in the search for next candidate coalition which must be a subset of $\mathcal{R}$ as well as the search always begin at the next coalition

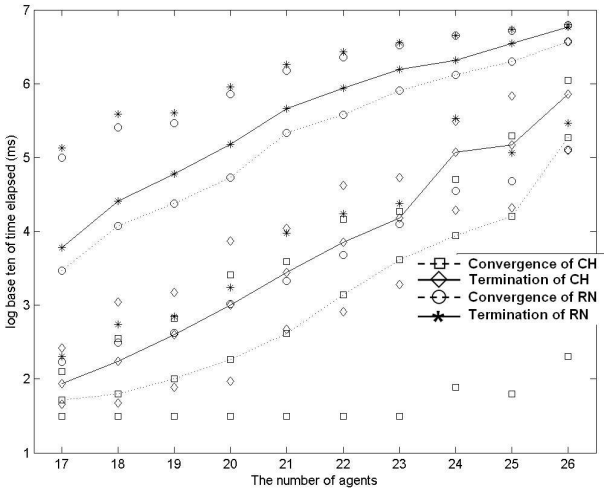

Figure 1: The graph show results on NMD. Each gadget, i.e., square, diamond, circle and star, represent elapsed time of CH Convergence, CH Termination, RN Convergence and RN Termination respectively.

in the same cardinality as the chosen coalition.

3. EXPERIMENTAL RESULTS

Settings end Empirical Results: We actually experiment our algorithm against Rahwan et.al.(RN), s apparently the state of the art, on 20 data distribution (in addition to Sandholm et al 4 distributions). [1] For each of 100 sample data in each distribution, we observe the elapsed time for i) convergence (the solution reaches the highest value but yet to terminate) and ii) termination (the algorithm terminates because either there is no way to improve the solution or it is timeout). For each of these elapsed time, we find the average, highest and lowest elapsed times for $17 \geq n \leq 26$ agents for both algorithms.

Due to limited space, we choose to show outputs of normail distribution. CH converges and terminates earlier than RN in all cases. Both algorithms terminate shortly after convergence across all variations. Empirically, our algorithm always converges earlier than RN. This implies that our algorithm guarantees better or, at least, as good as RN's result at anytime although our algorithm takes longer to terminate in some cases. The further implication of this is that generating coalition structures from best coalitions can help reach optimal coalition faster. This is based on the simple fact that merely generating coalitions alone can be intractable for a centralized system because all the existing algorithms (including ours) requires that all coalitions and their values must be observed before the actual generation– let alone the coalition structure generation. For example, it is impossible to execute any of the existing centralized algorithms for 60 agents because none of the existing single computer systems can offer enough memory. In terms of anytime algorithm which might be more appropriate for multi-agent systems, our algorithm empirically shows that it always generates better or, at least, as good solution as RN. Note that the

[1]Both algorithms are implemented in Java 1.5. Note that we were not given RN implementation, we try our best on several ways to ensure it runs at the fastest speed possible. The representation of the coalitions and their values are the same as in our implementation. The executions are done on Pentium 4 2GHz with 2GB of ram on Windows XP machines.

4. CONCLUSION AND FUTURE WORK

In this work, we have sought to develop an anytime algorithm that provides a guarantee of generating the optimal solution, given it has enough time. The algorithm advances the state of the art, in terms of anytime algorithm, by almost always generating solution better or, at least, as good as RN at any point in time. We present empirical results that support our claims. In the future, we would like to apply more efficient pruning machanism.

5. REFERENCES

[1] V. D. Dang and N. R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS'04), pp. 564-571*, 2004.

[2] J. P. Kahan and A. Rapoport. *Theories of Coalition Formation*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1984.

[3] D. L. Kreher and D. R. Stinson. *Combinatorial Algorithms Generation, Enumeration and Search*. CRC Press, Boca Raton, Florida, 1999.

[4] K. S. Larson and T. W. Sandholm. Anytime coalition structure generation: an average case study. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1):23 – 42, January 2000.

[5] T. Rahwan, S. D. Ramchurn, V. D. Dang, A. Giovannucci, and N. R. Jennings. Anytime optimal coalition structure generation. In *Proceedings of the 22nd Conf. on Artificial Intelligence (AAAI)*, pages 1184–1190, July 2007.

[6] T. Rahwan, S. D. Ramchurn, V. D. Dang, and N. R. Jennings. Near-optimal anytime coalition structure generation. In *Proceedings of the 20th International Joint Conference on AI (IJCAI)*, January 2007.

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

[8] T. Sandholm and V. Lesser. Coalition Formation among Bounded Rational Agents. *14th International Joint Conference on Artificial Intelligence*, pages 662–669, January 1995.

[9] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proc. of IJCAI*, pages 655–661, August 1995.

[10] C. Sombattheera and A. Ghose. Agent-based coalitions in dynamic supply chains. In *The 9th Pacific Asia Conference on Information Systems*, 2005.

[11] C. Sombattheera and A. Ghose. A pruning-based algorithm for computing optimal coalition structures in linear production domains. In *Proceedings of the 19th Canadian Conference on Artificial Intelligence*, 2006.