Coalition Structure Formation Using Anytime Dynamic Programming

Narayan Changder[1](✉), Animesh Dutta[1], and Aditya K. Ghose[2]

[1] National Institute of Technology, Durgapur, West Bengal, India
narayan.changder@gmail.com, animeshnit@gmail.com
[2] University of Wollongong, Wollongong, NSW 2522, Australia
aditya@uow.edu.au

Abstract. The optimal coalition structure generation is an important problem in multi-agent systems that remains difficult to solve. This paper presents a novel anytime dynamic programming algorithm to compute the optimal coalition structure. The proposed algorithm can be interrupted, and upon interruption, uses heuristic to select the largest valued coalition from each subproblem of size x and picks the rest of the unassigned agent from other subproblem of size $n - x$, where n is the total number of agents. We compared the performance of our algorithm against the only existing proposal in the literature for the optimal coalition structure problem that uses anytime dynamic programming using 9 distinct datasets (each corresponding to a different distribution). The empirical evaluation shows that our algorithm always generates better or, at least, as good a solution as the previous anytime dynamic programming algorithm.

Keywords: Multi agent system · Optimization · Coalition formation

1 Introduction

The optimal coalition structure generation is an interesting research problem in Multi-Agent Systems (MAS). This problem is interesting to MAS community due to its important applications and its computational challenges. The problem is challenging because of exponential growth of coalition structures when number of agents grows linearly. It is proved that optimal coalition structure generation problem is **NP-** complete [15]. Agents cooperate on issues of their common interest. Given a set of autonomous agents and a value to each subset of agents. One of the main challenges is to create disjoint groups of autonomous agents that cooperate in order to achieve their individual goal or to maximize the total payoff of the system. This complex research process is known as *Coalition Structure Formation* (CSF) process.

Coalition Structure Formation is important in many real world applications such as in e-commerce, customers can form a group/coalition to buy some product in bulk and can get a price discounts for bulk purchasing [18]. In distributed sensor network, sensors are grouped to make a coalition and work together to track targets of interest [2]. Several delivery companies may agree together and can form coalition to make profit by reducing the transportation costs [14]. To determine an optimal way in which agents must co-operate to get the maximum payoff from the system is a computationally hard problem. In simplest terms, a coalition is a group of agents with a common interest who agree to work together towards a common goal. A cooperative game is best choice to model such a scenario. Here, cooperative game is defined by n agents (or players), where the set of agents is denoted as $A = \{a_1, a_2, \ldots, a_n\}$. Any non empty subset of A is called as coalition, where value of each coalition C is given by a characteristic function $v(.)$. Furthermore, a collection of pairwise disjoint coalitions is called a "*coalition structure*" provided that all the agents are present in coalition structure. Formally, this complex procedure of coalition structure formation is defined as follows:

Definition 1. *Given a set of agents $A = \{a_1, a_2, \ldots, a_n\}$, a* Coalition Structure (CS) *over A is a partitioning of the agents into different coalitions $\{C_1, C_2, \ldots, C_k\}$, where k is called size of coalition structure i.e. $k = |CS|$. Such that it satisfies the following constraints:*

1. $C_j \neq \emptyset\,,\ j = \{1, 2, \ldots, k\}$
2. $C_i \cap C_j = \emptyset$ *for all* $i \neq j$ *and*
3. $\bigcup_{i=1}^{k} C_i = A$

For example, in a multi-agent system consisting of three agents $A = \{a_1, a_2, a_3\}$, we have total seven possible coalitions:

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

The set of all coalitions structures over A is denoted as Π^A

$$\Pi^A = \{\{a_1\}, \{a_2, a_3\}\}, \{\{a_3\}, \{a_1, a_2\}\}, \{\{a_2\}, \{a_1, a_3\}\}\{\{a_1\}, \{a_2\}, \{a_3\}\},$$
$$\{\{a_1, a_2, a_3\}\}$$

Now it is observed that the optimal coalition structure and complete partition of a set of agents are same. We are now ready to state our optimization problem formally.

Definition 2. *The value of any coalition structure CS is defined by*

$$V(CS) = \sum_{C_i \in CS} (v(C_i))$$

Generally, the goal of the coalition structure formation problem is to find the coalition structure which maximizes social welfare by finding an optimal coalition structure $CS^* \in \Pi^A$.

$$CS^* = arg\ max_{CS \in \Pi^A} V(CS)$$

The number of coalition structure increases exponentially as the number of participating agent increases linearly (for example, using 25 agents, there are total 4638590332330743949 coalition structures). The total number of coalition structure for n agents is also known as n^{th} Bell number and denoted as $\mathfrak{B}_\mathfrak{n}$ [5], satisfies $\alpha n^{n/2} \leq \mathfrak{B}_\mathfrak{n} \leq n^n$ for some positive constant α. Hence, we can not directly enumerate all the coalition structure in polynomial time.

In Multi-agent systems there are often time limits to get the solution of the problem, and after deadline is over the result becomes useless. The applicability of coalition structure formation problem in multi-agent settings with hard time constraint requires that the result must comes before the time limit is over.

There are two popular techniques available for coalition structure formation including dynamic programming [19] and anytime search algorithms [8,12,13,16]. The advantage of dynamic programming is that it gives optimal result without enumerating all the coalition structures. However, the biggest disadvantage is that it needs to be run to completion to provide optimal solution. Hence, this method is not a good choice when the time required to produce optimal solution is larger than the time available to the agents. In multi-agent settings without hard time limits, dynamic programming algorithm is efficient to solve many real life problem instances. However, in other circumstances with strict deadline and short execution time, we need an alternative approaches.

Against the research aims outlined above, this paper makes the following contributions to the coalition structure formation problem.

– We proposed anytime dynamic programming algorithm for coalition structure formation. Our anytime dynamic programming is an extension of basic dynamic programming [19] for coalition structure formation.
– Anytime dynamic programming needs a good heuristic to solve the problem. The proposed algorithm uses an inexpensive greedy approach to choose a good answer from the remaining possible solutions. The experimental result shows that our greedy strategy works well.
– We compared our algorithm empirically with the existing anytime dynamic program [16] for 9 different data distributions and result shows that our algorithm occasionally fails to produce good result for certain data distributions but for most of the distribution it always generates better or, at least, as good solution as previous algorithm [16]. We experiments our algorithm for 16 agents and averaged the runtime over 40 runs for each experiment.

Our anytime dynamic programming is a shifted paradigm of traditional dynamic programming for coalition structure formation problem. To help the reader for understanding how our anytime dynamic programming algorithm works, we explain the traditional dynamic programming algorithm [19] followed by our novel anytime dynamic programming algorithm to compute the optimal coalition structure.

2 The DP Algorithm

The first dynamic program to solve coalition structure formulation problem is proposed by Yin Yeh [19]. The approach used is shown in Algorithm 1.

Algorithm 1. Dynamic Programming algorithm

```
Input: Set of all possible non- empty subsets of $n$ agents $(2^n - 1)$ . The value of any
coalition $C$ is $v(C)$. If no $v(C)$ is specified then $v(C) = 0$
Output: Optimal coalition structure $CS^*(n)$
 1: for $i = 1$ to $n$ do
 2:     for $C \subseteq A$, where $|C| = i$ do                    ▷ $A$ is set of $n$ agents
 3:         $V_t(C) \leftarrow v(C)$
 4:         $P_t(C) \leftarrow \{C\}$
 5:         for $C' \subset C$ do      ▷ for every possible way of splitting $C$ into two halves
 6:             if $V_t(C') + V_t(C \setminus C') > v(C)$ then
 7:                 $V_t(C) \leftarrow V_t(C') + V_t(C \setminus C')$
 8:                 $P_t(C) \leftarrow \{C', C \setminus C'\}$
 9:             end if
10:         end for
11:     end for
12: end for
13: $CS^* \leftarrow \{A\}$
14: for $C \in CS^*$ do
15:     if $P_t(C) \neq \{C\}$ then
16:         $CS^* \leftarrow (CS^*/C, P_t(C))$
17:         Go to line 14 and start with the new $CS^*$
18:     end if
19: end for
20: Return  $CS^*(n)$
```

The way dynamic programming works is by manipulating two tables — partition table $P_t[C]$ and value table $V_t[C]$. For example in Table 1, $C = \{1, 2\}$, in this case $P_t[C] = P_t[\{1, 2\}] = \{1\}\{2\}$ and $V_t[C] = V_t[\{1, 2\}] = 70$. For any coalition $C \subseteq A$ it calculates value of $P_t[C]$ and $V_t[C]$ as follows. First, coalition C is split into two halves in all possible ways and computes the highest welfare with the original welfare $v(C)$ of coalition C. If it finds best splitting which gives highest welfare, stores the splitting into $P_t[C]$ otherwise stores coalition C into $P_t[C]$ without splitting[1] coalition C. Suppose coalition C split into two coalitions as $\{C', C''\}$ then it is evaluated as $V_t[C'] + V_t[C'']$. In other words it check

$$v(C) < V_t[C'] + V_t[C'']$$

Note that $v(C)$ is the original input values to all coalitions whereas $V_t[C']$ and $V_t[C'']$ is previously computed value of coalition C' and C''. To compute $V_t[C]$ the algorithm must first evaluates all the $V_t[\,]$ values of the subsets of C. Below Table 1 shows an example how to compute P_t and V_t with 4 agents $A = \{1, 2, 3, 4\}$

[1] If the coalition contains single agent, we do not need to split it anymore.

Table 1. Example of DP program with 4 agents

Size	C	v(C)	All splitting by DP	P_t	V_t
1	{1}	30	$v[\{1\}] = 30$	$\{1\}$	30
	{2}	40	$v[\{2\}] = 40$	$\{2\}$	40
	{3}	25	$v[\{3\}] = 25$	$\{3\}$	25
	{4}	45	$v[\{4\}] = 45$	$\{4\}$	45
2	{1,2}	50	$v[\{1,2\}] = 50, v\{1\} + v\{2\} = 70$	$\{1\}\{2\}$	70
	{1,3}	60	$v[\{1,3\}] = 60, v\{1\} + v\{3\} = 55$	$\{1,3\}$	60
	{1,4}	80	$v[\{1,4\}] = 80, v\{1\} + v\{4\} = 75$	$\{1,4\}$	80
	{2,3}	55	$v[\{2,3\}] = 55, v\{2\} + v\{3\} = 65$	$\{2\}\{3\}$	65
	{2,4}	70	$v[\{2,4\}] = 70, v\{2\} + v\{4\} = 85$	$\{2\}\{4\}$	85
	{3,4}	80	$v[\{3,4\}] = 80, v\{3\} + v\{4\} = 70$	$\{3,4\}$	80
3	{1,2,3}	90	$v[\{1,2,3\}] = 90, v\{1\} + v\{2,3\} = 95$ $v\{2\} + v\{1,3\} = 100, v\{3\} + v\{1,2\} = 95$	$\{2\}\{1,3\}$	100
	{1,2,4}	120	$v[\{1,2,4\}] = 120, v\{1\} + v\{2,4\} = 115$ $v\{2\} + v\{1,4\} = 110, v\{4\} + v\{1,2\} = 115$	$\{1,2,4\}$	120
	{1,3,4}	100	$v[\{1,3,4\}] = 100, v\{1\} + v\{3,4\} = 110$ $v\{3\} + v\{1,4\} = 105, v\{4\} + v\{1,3\} = 105$	$\{1\}\{3,4\}$	110
	{2,3,4}	115	$v[\{2,3,4\}] = 115, v\{2\} + v\{3,4\} = 120$ $v\{3\} + v\{2,4\} = 110, v\{4\} + v\{2,3\} = 110$	$\{2\}\{3,4\}$	120
4	{1,2,3,4}	140	$v[\{1,2,3,4\}] = 140, v\{1\} + v\{2,3,4\} = 150$ $v\{2\} + v\{1,3,4\} = 150, v\{3\} + v\{1,2,4\} = 145$ $v\{4\} + v\{1,2,3\} = 145, v\{1,2\} + v\{3,4\} = 120$ $v\{1,3\} + v\{2,4\} = 145, v\{1,4\} + v\{2,3\} = 145$	$\{1\}\{2,3,4\}$	150

To compute the value V_t for coalition of size x, algorithm need to calculates V_t values for all the coalition of size $1, 2, \ldots, x-1$. Whenever the algorithm determines all the entries of P_t and V_t the optimal coalition structure CS^* can be computed recursively as shown in Table 1. Algorithm looks for grand coalition structure $\{1,2,3,4\}$ and checks it is more beneficial to split $\{1,2,3,4\}$ into $\{1\}$ and $\{2,3,4\}$. Similarly by looking at coalition $\{2,3,4\}$ algorithm finds it is more beneficial to split $\{2,3,4\}$ into $\{2\}$ and $\{3,4\}$. As a result, the optimal solution is {{1},{2},{3,4}}. The running time of algorithm is calculated as follows. there are total $\binom{n}{k}$ coalition of size k over the n agents and each of them requires $O(2^k)$ time

$$\sum_{k=1}^{n} \binom{n}{k} O(2^k)$$

According to the binomial theorem, we have

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Now, take $x = 1$ and $y = 2$

$$(1+2)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Hence we get bound as:

$$\sum_{k=1}^{n} \binom{n}{k} O(2^k) = O(3^n)$$

Having described how DP operates, we will now describes how to get a good solution after DP completes k^{th} iteration.

3 Anytime Dynamic Programming

An anytime algorithm is an algorithm that can return a valid solution to a problem even if it is interrupted at any time before it ends. Our algorithm uses two heuristics.

1. At runtime it calculates coalition structures by greedily adding all the singleton coalition i.e. $\{(a_1), (a_2), \dots (a_n)\}$ and grand coalition $\{(a_1, a_2, a_3, \dots, a_n)\}$, then it selects better of two as initial solution.
2. After algorithm completes k^{th} iteration[2], it chooses all the largest valued coalition from each of the subproblem of sizes $n, n-1, n-2 \dots 3, 2, 1$[3] and rest of the unassigned agent/agents are picked from the subproblem of sizes $0, 1, 2, \dots, n-3, n-2, n-1$. Note that whatever be the coalition of sizes $1, 2, \dots k$, it has already been stored in optimal way. The intuition behind this greedy strategy is as follows: Since the algorithm is stopped before the completion, for example, if it stops after k^{th} iteration and it might be the case that there is some large valued coalition in rest of the coalition with sizes $k+1, k+2, \dots n$

Now, the DP algorithm starts and it solves all the incremental subproblems of size $1, 2, \dots k$. If run to completion, it returns optimal solution. If the algorithm stops prematurely, the better of the initial solution and current iterative solution using above heuristic is returned. The pseudo-code of algorithm is given in Algorithm 2. The algorithm works as follows: In line 1, algorithm creates an initial solution using grand coalition and singleton coalitions, then it picks one amongst them which gives maximum social welfare. The algorithm chooses singleton coalition because it is not clear about the relationships among agents. If the domain happens to be super-additive, the optimal coalition structure is obviously grand coalition.

[2] After k^{th} iteration algorithm solves all the subproblem of size $1, 2, \dots, k$.

[3] We pick the largest valued coalition of size n, because if $C_{max} = n$, then it contains optimal solution.

Algorithm 2. Anytime Dynamic Programming algorithm

```
Input: Set of all possible non-empty subsets of $n$ agents ($2^n - 1$). The value of any coalition $C$ is $v(C)$. If no $v(C)$ is specified then $v(C) = 0$ and $C_{max}$ is the maximum number of iteration
Output: Good coalition structure $CS^*(n)$
 1: $CS^*_{initial}$= max$\{\sum_{i=1}^{n} v(a_i),\ v(a_1, a_2, \ldots, a_n)\}$   ▷ Calculate initial solution by
        using grand coalition and singleton coalitions.
 2: for $i = 1$ to $C_{max}$ do
 3:     for $C \subseteq A$, where $|C| = i$ do                ▷ $A$ is set of $n$ agents
 4:         $V_t(C) \leftarrow v(C)$
 5:         $P_t(C) \leftarrow \{C\}$
 6:         for $C' \subset C$ do    ▷ for every possible way of splitting $C$ into two halves
 7:             if $V_t(C') + V_t(C \setminus C') > v(C)$ then
 8:                 $V_t(C) \leftarrow V_t(C') + V_t(C \setminus C')$
 9:                 $P_t(C) \leftarrow \{C', C \setminus C'\}$
10:             end if
11:         end for
12:     end for
13: end for
14: Maximum $\leftarrow 0$
15: for $i = n$ to 1 do    ▷ Heuristic is used to pick largest valued coalition for each
        subproblem
16:     $X \leftarrow C|\ \max_{C \in C^i}(V_t(C))$            ▷ $C^i$ is the coalitions of size $i$
17:     $Y \leftarrow U \setminus X$[a]      ▷ $U$ is the set of all agents and $Y$ is the unassigned agents
            belongs to coalition of sizes $n - i$.
18:     Temp$_{value} \leftarrow V_t(X) + V_t(Y)$  ▷ for any coalition $C \in \{X, Y\}$, if $|C| > C_{max}$,
            then it uses $v(C)$ value.
19:     if Temp$_{value} > Maximum$ then
20:         $Maximum \leftarrow$ Temp$_{value}$
21:         $CS_{Temp} \leftarrow \{X, Y\}$
22:     end if
23: end for
24: $CS^*(n) \leftarrow$ Best of $CS^*_{initial}$ and $CS_{Temp}$
```

[a] Each time X is considered, Y is the complement.

Line 2–13 is same as dynamic programming. C_{max} is the iteration limit of proposed algorithm. The incremental subproblem of size 1 to C_{max} are solved exactly with dynamic programming. After the iteration limit C_{max} reached, all the coalition of size 1 to C_{max} are already solved and results are stored in table P_t and V_t. Note that if $C_{max} = n$, it will return optimal solution.

Line 15–18 is used for greedy heuristic after the iteration limit C_{max} is reached. The heuristic used here is to pick up the largest valued coalition with size $n, n-1, n-2, \ldots, 1$ and pick the rest of the unassigned agents with sizes $0, 1, 2, \ldots n-1$[4]

[4] If size of coalition X is 0 then the value of the coalition $X = 0$.

Line 19–22 is to keep track of highest valued coalition structure found till now. Line 24 compares the best of initial solution and the solution using greedy approach used after the iteration limit C_{max}.

Example 1. *Consider the example used in Table 2. Suppose that the iteration limit $C_{max} = 2$. All the coalition of size $1, \ldots 2$ are solved and stored in the table after the iteration limit $C_{max} = 2$ is reached.*

At first the algorithm creates initial solution as follows:

$$CS^*_{initial} = \text{Max}\{\underbrace{\{v(1,2,3,4)\}}_{\text{Value=140}}, \underbrace{\{v(1) + v(2) + v(3) + v(4)\}}_{\text{value=30+40+25+45=140}}\}$$

Table 2. Example of DP program with 4 agents with $C_{max} = 2$

Size	C	v(C)	All splitting by DP	P_t	V_t
1	{1}	30	$v[\{1\}] = 30$	{1}	30
	{2}	40	$v[\{2\}] = 40$	{2}	40
	{3}	25	$v[\{3\}] = 25$	{3}	25
	{4}	45	$v[\{4\}] = 45$	{4}	45
2	{1,2}	50	$v[\{1,2\}] = 50, v\{1\} + v\{2\} = 70$	{1}{2}	70
	{1,3}	60	$v[\{1,3\}] = 60, v\{1\} + v\{3\} = 55$	{1, 3}	60
	{1,4}	80	$v[\{1,4\}] = 80, v\{1\} + v\{4\} = 75$	{1, 4}	80
	{2,3}	55	$v[\{2,3\}] = 55, v\{2\} + v\{3\} = 65$	{2}{3}	65
	{2,4}	70	$v[\{2,4\}] = 70, v\{2\} + v\{4\} = 85$	{2}{4}	85
	{3,4}	80	$v[\{3,4\}] = 80, v\{3\} + v\{4\} = 70$	{3, 4}	80
3	{1,2,3}	90			
	{1,2,4}	120			
	{1,3,4}	100			
	{2,3,4}	115			
4	{1,2,3,4}	140			

Hence, our initial solution $CS^*_{initial}$ is any of them because they gives same value.

Next, algorithm picks the maximum valued coalition with sizes 4 and it is the grand coalition $\{1, 2, 3, 4\}$ with value 140.

Now, algorithm picks the maximum valued coalition of size 3, which is the coalition $\{1, 2, 4\}$ with value 120. Then algorithm chooses the rest of unassigned agent from coalition of size 1, which is the coalition $\{3\}$ with value 25. Total value of coalition structure $\{1, 2, 4\}\{3\}$ is $120 + 25 = 145$.

Next, maximum valued coalition $\{2, 4\}$ of size 2 is picked up and rest of unassigned agents is picked up from coalition of size 2. Total value of coalition structure is $\{2, 4\}\{1, 3\}$ is $85 + 60 = 145$.

At last maximum valued coalition $\{4\}$ of size 1 is picked up and rest of unassigned agents form coalition of size 3 is picked up. Total coalition value is $\{4\}\{1,2,3\} = 45 + 90 = 135$.

Now, compare initial solution $CS^*_{initial}$ with present greedy solution and finds that the maximum value it gives is 145 with coalition structure $\{2,4\}\{1,3\}$. The final coalition structure is $\{2\}\{4\}\{1,3\}$ because we see that $\{2,4\}$ is stored as $\{2\}\{4\}$. Note that algorithm could also choose $\{1,2,4\}\{3\}$ as final coalition structure because it also gives value 145.

4 Anytime Property of Proposed Algorithm

The anytime property is also satisfied by proposed algorithm

i) **Monotonicity**— the quality of the result is a nondecreasing function of computation time. In general proposed algorithm is monotonic.
Proof— The algorithm is clearly monotonic. Suppose we have n agents in the system. For this scenario we have problem sizes of $1, 2, \ldots, n$. With problem size i, all the coalition contains i number agents. Suppose, maximum valued coalition in problem size i is M_i. Let algorithm interrupted after k^{th} iteration. Now, all the coalition of sizes $1, 2, \ldots k$ are already solved. In this case the maximum valued coalition structure is computed as follows:

$$max_{\forall i \in [1,2,\ldots,k,k+1,\ldots n]}\{v(M_i) + v(U \setminus M_i)\}$$

where U is the set of all agents. The largest valued coalition in problem size k is M_k. If maximum valued coalition structure contains any of the coalition with problem sizes $1, 2 \ldots, k$, then in $(k+1)^{th}$ iteration this value must be the same as the value generated in k^{th} iteration or greater because the values generated in $(k+1)^{th}$ iteration is depends on the values generated in $1, 2, \ldots, k$ iteration.

ii) **Preemptability**— the algorithm can be suspended and resumed with minimal overhead. Proposed algorithm is clearly preempt-able. After the iteration limit is reached, algorithm needs to check the largest valued coalition from each subproblem of size x, where $x \in [1, n]$ and fetch the remaining unassigned agents from subproblem of sizes $n - x$, where n is the total number of agents. This procedure takes $O(n)$ time.

5 Evaluating Proposed Algorithm

In this section we describe the environment on which algorithms have been tested.

5.1 Experimental Setup

To calculate the time performance, we repeat each experiment 40 times and averaged the runtime. The algorithms are implemented in Python (Version:3.4), compiled in IDE Pycharm, and the experiments were run on a Intel(R) Core(TM) i5-4690 CPU, running at 3.50 GHz under Windows 7 operating system (64 bit).

5.2 Dataset Generation

The **NP**-complete problems are intractable but there is no conclusive proof. We cannot deny the possibility that **NP**-complete problem is solvable in polynomial time. Intractable in the sense that there is no polynomial time algorithm for that problem which gives correct result. That means every algorithm for coalition structure formation is imperfect because we know this is an **NP**-complete problem. One way to validate or compare imperfect algorithm for **NP** hard combinatorial optimization problem is to run them on typical problem instances and see how often they fail. The word imperfect means that there is some input for which the algorithm fails to give the correct result. Any imperfect algorithm is usefull if they do not fail too often. With this in mind, we compare proposed algorithm with existing anytime dynamic programming [16] using different value distributions. Specifically, we consider the following distributions.

i) **Uniform**— as studied by Larson and Sandholm [6]: for all coalition $C \in 2^A - 1$, $v(C) \sim U(a, b)$, where $a = 0$ and $b = |C|$
ii) **Modified Uniform**— as proposed by Service and Adams [17]. The value of each coalition C is drawn uniformly $v(C) \sim U(a, b)$, where $a = 0$ and $b = 10 \times |C|$., next a random number r is generated $r \sim U(0, 50)$ and is added to the coalition value $v(C)$ with probability 0.2.
iii) **Normal**— as studied by Rahwan et al. [12] every coalition value is drawn from $v(C) \sim N(\mu, \sigma^2)$, where $\mu = 10 \times |C|$ and $\sigma = 0.1$
iv) **Modified Normal**— as proposed by Rahwan et al. [10]. The value of each coalition C is first drawn $v(C) \sim N(a, b)$, where $a = 10 \times |C|$ and $b = 0.01$, next a random number r is generated $r \sim U(0, 50)$ and is added to the coalition value $v(C)$ with probability 0.2.
v) **Beta** — The value of each coalition C is drawn as $v(C) \sim |C| \times$ Beta (α, β), where $\alpha = \beta = 0.5$.
vi) **Gamma**— The value of each coalition C is drawn as $v(C) \sim |C| \times$ Gamma (x, θ), where $x = \theta = 2$
vii) **Agent-based Uniform**— as proposed by Rahwan et al. [10], each of the agent a_i is assigned a random power $p_i \sim U(0, 10)$, reflecting its average performance over all coalitions. Then for all coalition C in which agent a_i appears, the actual power of a_i in C is determined as $p_i^C \sim U(0, 2 \times p_i)$ and the coalition value is calculated as the sum of all the members power in that coalition. That is, $\forall C$, $v(C) = \sum_{a_i \in C} p_i^C$.
viii) **Agent-Based Normal**— as proposed by Tomasz Michalakn et al. [7], each of the agent a_i is assigned a random power $p_i \sim N(10, 0.01)$. Then for all coalition C in which agent a_i appears, the actual power of a_i in C is determined as $p_i^C \sim N(p_i, 0.01)$ and the coalition value is calculated as the sum of all the member's power in that coalition. That is, $\forall C$, $v(C) = \sum_{a_i \in C} p_i^C$.
ix) **Normally Distributed Coalition Structures (NDSC)**— as proposed by Rahwan et al. [13], value of each coalition C is drawn as $v(C) \sim N(\mu, \sigma^2)$, where $\mu = |C|$ and $\sigma = \sqrt{|C|}$.

In the comparison graph shown in Figs. 1 and 2, we call our algorithm as PADP (Proposed Anytime Dynamic Programming) and the algorithm proposed by Service and Adams [16] is denoted as ADP (anytime dynamic programming).

6 Performance

In order to evaluate the proposed algorithm we implemented it in the Python programming (Version: 3.4) language and tested the behavior of coalition structure formation problem. As our benchmark we use the algorithm presented in [16]. We selects algorithm in [16] because according to our knowledge, it is the only available coalition structure formation algorithm using anytime dynamic programming. Here we present experimental results on the behaviors of PADP and ADP [16], considering in particular solution quality and runtime performances. We tested both the algorithms for 16 agents and compare it with the increasing iteration limit of basic dynamic programming. For each of the above distributions, we plotted the termination times of both algorithms for each iteration with 16 agents. Here, time is measured in seconds, and plotted on a log scale and similarly solution is also plotted on a log scale. Figure 1 shows the resulting performances of both algorithms with respect to solution obtained. The results show that if proposed algorithm is interrupted before running to completion, it may still return a solution with relatively high quality than the algorithm proposed by Service and Adams [16]. Specifically, we find that.

i) Except Normal and Agent based normal distribution our proposed algorithm always produces better solution as compared to previous algorithm [16]. The results are shown in Fig. 1a, 1b, 1e, 1f, 1g, 1i.
ii) With Normal and Agent based normal distribution (shown in Fig. 1c, 1h) both algorithm performances are same in terms of solution quality, because of the fact that, under these distribution, the optimal solution mainly consisted of the singleton coalitions.
iii) With Modified Uniform, Normal, Beta and Agent-based Normal distributions,(shown in Fig. 1b, 1c, 1e, 1i) PADP takes very less time to produce near optimal solution.
iv) With Modified Normal distribution, PADP fails to produce better result for first $n/2$ iterations, after that it always produce better result than ADP.

In terms of runtime, results show that the proposed algorithm runs little bit faster for first few iterations and after that both algorithms running time is same. The runtime comparison is given in Fig. 2.

7 Related Work

The current research work on coalition structure formation can be classified into three categories [8].

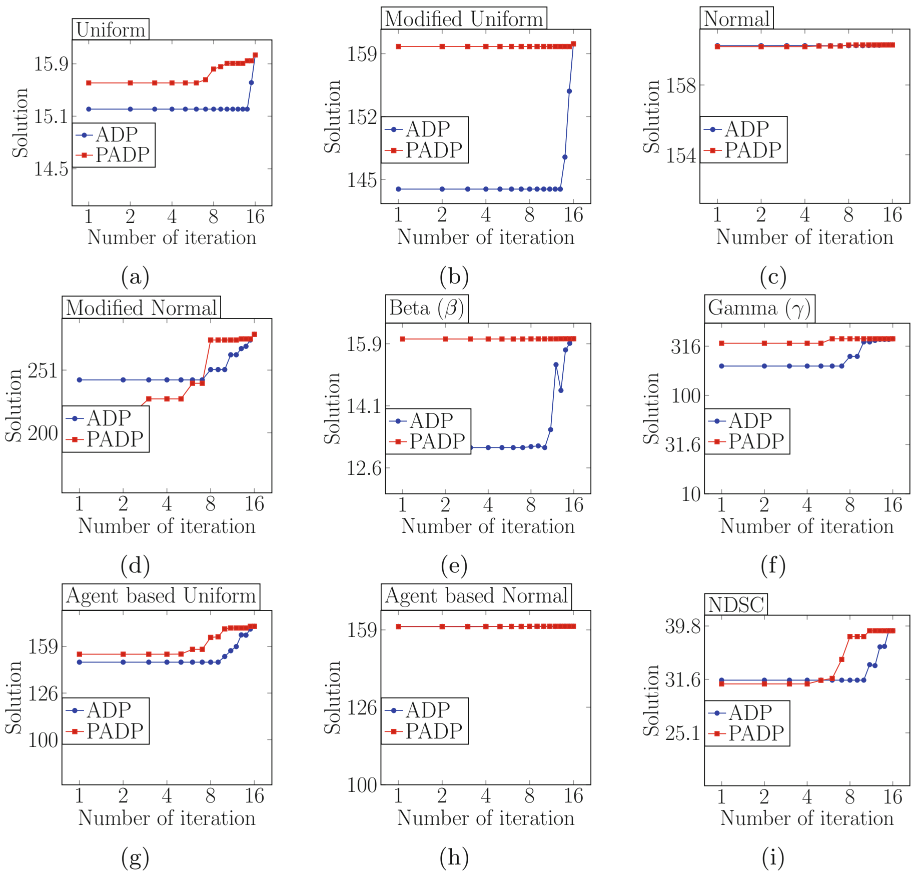

Fig. 1. Solution quality of proposed algorithm and ADP

1. **Anytime algorithms**— It permits premature termination (i.e., before the optimal solution has been found) but at the same time it provides guarantees on the quality of the solution. One of the disadvantage of anytime algorithm in coalition formation mechanism is that they all requires, in the worst case to check all coalition structures. Hence, time required is $O(n^n)$.
2. **Design-to-time algorithms**— This type of algorithm guarantees to return an optimal solution but to do so, it must run on completion. That is they can not produce intermediate result like anytime algorithms.
3. **Heuristics algorithms**—This type of algorithm sacrifices quality guarantees of solution for speed. The main drawback of this type algorithm is that it is impossible to verify the quality of generated solution.

The optimal coalition structure can be generated by using dynamic programming algorithm [19] in $O(3^n)$ time, however it is impractical for moderate size of inputs.

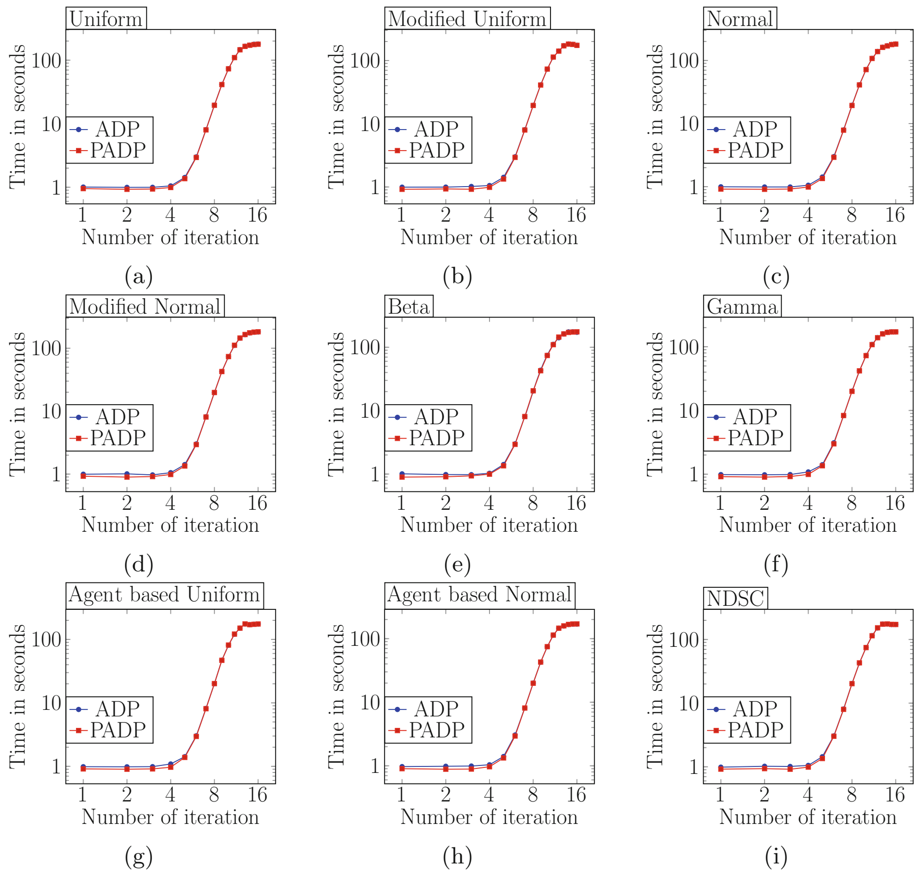

Fig. 2. Time performance of proposed algorithm and ADP

The state-of-the-art *Improved Dynamic Programming* (IDP) [9] algorithm is improved version of [19] it is improved by Rahwan and Jennings [9] to enhance the usability of dynamic programming. Their approach is not to evaluate some unnecessary splitting. The author shows that their approach is empirically faster and uses less memory, but still worst case runtime is $O(3^n)$. The main drawback of this approach is it does not have anytime properties.

Due to the high complexity of dynamic programming many researchers are developing anytime algorithm which allow quality suboptimal solution very faster. Many anytime coalition structure formation algorithm operates on the space of all coalition structures, between $\omega(n^n)$ to $O(n^n)$ [15]. Sandholm et al. [15] proposes first anytime algorithm for coalition structure formation problem. They proved that to provide a bound on the quality of the solution algorithm needs to check at-least 2^{n-1} coalition structures. Their approach uses coalition structure graph, where nodes in the graph are coalition structures and edges between nodes are splitting(or mergers) of coalition structure.

On the other hand the algorithm proposed by Dang and Jenning [3] also used the coalition structure graph representation as Sandholm et al. [15], they define and search a particular graph subsets and shown that their technique empirically generate tighter quality guarantee's than Sandholm et al. [15]. Moreover, several researchers developed anytime algorithms that searches the space of coalition structure graph in different ways [3,8,13].

Rahwan and Jennings [13] have not used coalition structure graph. They use a totally new representation of coalition structures space based on integer partitions and shown that their approach empirically gives high quality solutions than any other previous anytime algorithms. Their approach is called anytime IP algorithm.

The algorithm in [8], called IDP-IP, it combines positive sides of IDP and anytime IP algorithm and avoids their weakness.

Service and Adams [16] proposed an anytime dynamic programming (ADP) to solve coalition structure formation problem. Their approach is to first create a greedy solution in $O(2^n)$ time by greedily adding the coalition of largest value, out of unassigned agents. Next, dynamic programming is used to solve the problem with sizes $1, 2, \ldots k$. If run to completion, it gives optimal result. Otherwise, it will produce better of greedy solution and current iterative solution.

In this paper, we raised the question of how to achieve high quality solutions to the coalition structure formation problem (see e.g. [11] for a survey of the problem), especially when the search execution times to find solutions are very limited. Motivated by the observations in [1,16], we developed our algorithm and compared it empirically with ADP [16]. In search of creating an anytime algorithm for coalition structure formation we choose dynamic programming as basic tool because it is the algorithm available with lowest worst case time complexity $O(3^n)$.

8 Conclusion

Coalition structure formation is a computationally hard, combinatorial problem in multi-agent systems. This work represents a new anytime dynamic programming for solving coalition structure formation and has been compared with previous anytime dynamic programming algorithm [16]. The backbone of both algorithms are dynamic programming but they used different heuristics to get the solution.

Both the algorithm use preprocessing to make a greedy solution before dynamic program starts. The preprocessing phase of ADP [16] runs in $O(2^n)$ and the preprocessing phase of our proposed algorithm runs in $O(n)$. In most of the input distribution, proposed algorithm performs well than ADP [16].

Acknowledgments. This research work is funded by Visvesvaraya PhD scheme of DeitY (Department of Electronics & Information Technology), Govt. of India.

References

1. Boddy, M.S.: Anytime problem solving using dynamic programming. In: AAAI, pp. 738–743 (1991)
2. Dang, V.D., Dash, R.K., Rogers, A., Jennings, N.R.: Overlapping coalition formation for efficient data fusion inmulti-sensor networks. In: AAAI, vol. 6, pp. 635–640 (2006)
3. Dang, V.D., Jennings, N.R.: Generating coalition structures with finite bound from the optimal guarantees. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 564–571. IEEE Computer Society (2004)
4. Kahan, J.P., Rapoport, A.: Theories of Coalition Formation. Psychology Press, Palo Altom (2014)
5. Kreher, D.L., Stinson, D.R.: Combinatorial Algorithms: Generation, Enumeration, and Search, vol. 7. CRC Press, Boca Raton (1998)
6. Larson, K.S., Sandholm, T.W.: Anytime coalition structure generation: an average case study. J. Exp. Theoret. Artif. Intell. **12**(1), 23–42 (2000)
7. Michalak, T., Rahwan, T., Elkind, E., Wooldridge, M., Jennings, N.R.: A hybrid exact algorithm for complete set partitioning. Artif. Intell. **230**, 14–50 (2015)
8. Rahwan, T., Jennings, N.R.: Coalition structure generation: dynamic programming meets anytime optimization. In: AAAI, vol. 8, pp. 156–161 (2008)
9. 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, vol. 3, pp. 1417–1420. International Foundation for Autonomous Agents and Multiagent Systems (2008)
10. Rahwan, T., Michalak, T., Jennings, N.R.: A hybrid algorithm for coalition structure generation (2012)
11. Rahwan, T., Michalak, T.P., Wooldridge, M., Jennings, N.R.: Coalition structure generation: a survey. Artif. Intell. **229**, 139–174 (2015)
12. Rahwan, T., Ramchurn, S.D., Dang, V.D., Giovannucci, A., Jennings, N.R.: Anytime optimal coalition structure generation. In: AAAI, vol. 7, pp. 1184–1190 (2007)
13. Rahwan, T., Ramchurn, S.D., Jennings, N.R., Giovannucci, A.: An anytime algorithm for optimal coalition structure generation. J. Artif. Intell. Res. **34**, 521–567 (2009)
14. Sandhlom, T.W., Lesser, V.R.: Coalitions among computationally bounded agents. Artif. Intell. **94**(1), 99–137 (1997)
15. Sandholm, T., Larson, K., Andersson, M., Shehory, O., Tohmé, F.: Coalition structure generation with worst case guarantees. Artif. Intell. **111**(1), 209–238 (1999)
16. Service, T.C., Adams, J.A.:Anytime dynamic programming for coalition structure generation. In: Proceedings of the 9th International Conference on AutonomousAgents and Multiagent Systems, vol. 1, pp. 1411–1412. International Foundation for Autonomous Agents and Multiagent Systems (2010)
17. Service, T.C., Adams, J.A.: Approximate coalition structure generation. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (2010)
18. Tsvetovat, M., Sycara, K.: Customer coalitions in the electronic marketplace. In: Proceedings of the Fourth International Conference on Autonomous Agents, pp. 263–264. ACM (2000)
19. Yun Yeh, D.: A dynamic programming approach to the complete set partitioning problem. BIT Numer. Math. **26**(4), 467–474 (1986)