Novel Heuristics for Coalition Structure Generation in Multi-agent Systems

Mustansar Ali Ghazanfar and Adam Prugel-Bennett

***Abstract*—A coalition is a set of self-interested agents that agree to cooperate for achieving a set of goals. Coalition formation is an active area of research in multi-agent systems nowadays. Central to this endeavour is the problem of determining which of the many possible coalitions to form in order to achieve some goal, which is called coalition structure generation. Coalition structure generation problem is extremely challenging due to the number of possible solutions that need to be examined, which grows exponentially with the number of agents involved. Generally, agents would enumerate all possible coalitions, store them in memory, and then try to construct the coalition structure that maximizes the sum of the values of the coalitions. However, this is not feasible when we have a large number of agents, and other constraints on execution time, and memory. Hence, there is a need to develop an algorithm that can generate solutions rapidly for large number of agents while can provide bounds on the value of solution as well. With this in mind, we propose two new heuristics, namely LocalSearch and GreedySearch, for generating the coalition structure, which satisfy these properties. We empirically show that these heuristics are able to return 'good-enough' solutions in very short time. They enhance the performance of state of the art algorithm, IP (proposed by [12]) in terms of increased lower bound, anytime property, and solution quality. Furthermore, we implemented different heuristics for selecting a sub-space in the IP algorithm and show how the time required to find a good-enough solution depends on the selection of a sub-space in the IP algorithm**.

***Index Terms*—Multi-agent systems, Coalition formation, Coalition structure generation, Heuristics**

I. INTRODUCTION

Cooperation among agents is an important keystone in Multi-Agent Systems (MAS), which enables them to solve a problem efficiently. Agents cooperate in many economic milieus on issues of common interest, which results in the formation of coalition [1]. For this purpose, agents need to determine the optimal set of agents with whom to enter into a coalition (i.e. the best grouping of agents). This problem is formally referred to as the Coalition Structure Generation (CSG) problem.

Manuscript received March 10, 2010. The work reported in this paper has formed part of the Instant Knowledge Research Programme of Mobile VCE, (the Virtual Centre of Excellence in Mobile & Personal Communications), www.mobilevce.com. The programme is co-funded by the UK Technology Strategy Board's Collaborative Research and Development programme. Detailed technical reports on this research are available to all Industrial Members of Mobile VCE. The authors would like to thank Sarvapali D. Ramchurn and Talal Rahwan for their support.

Both authors are with School of Electronics and Computer Science, University of Southampton, Highfield Campus, SO17 1BJ, United Kingdom (phone: +44 (023) 80594473; fax: +44 (023) 80594498; email: {mag208r,adp}@ecs.soton.ac.uk).

Suppose that we are given set of agents $1, 2, \dots \dots n \in A$, and the value of a coalition , s, is specified by a characteristic function v(.). Then the value of the coalition structure (CS) is:

$$V(CS) = \sum_{S \in CS} (v_s)$$

Generally, the goal is to maximize the social welfare by discovering the optimal coalition structure [2].

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

Finding the optimal coalition structure is very challenging as the computational complexity of finding the optimal coalition structure is exponential[1] in the number of agents and is shown to be NP-hard [3]. To date, a number of algorithms have been proposed to solve CSG problem, but there has been less work on algorithms that can generate good-enough solutions quickly. In this paper, we propose new heuristics to solve this problem and show how good-enough solution can be generated, while balancing the properties, such as execution time and memory.

II. RELATED WORK

Existing literature defines various CSG algorithms that can be classified into three main classes: Dynamic Programming (DP) based algorithms, heuristic based algorithms, and anytime algorithms [4]. Dynamic programming algorithms generate optimal solution (i.e. optimal coalition structure) with minimal computational complexity. They provide a guarantee on the performance of the algorithm in the worst-case scenarios. [5], [6], [7] develop DP based algorithms but these algorithms can not be used for large number of agents (>20). Heuristic based algorithms are not designed to find the optimal solution; rather their focus is on finding good solutions. In this context, [8] employ an order-based genetic algorithm (OBGA) as a stochastic search process to discover the optimal coalition structure. The main limitation of this algorithm is that, it provides no guarantee about finding the optimal CS, and it

[1] The number of coalition structure grows in $O(n^n)$ with the number of agents [3].

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

cannot specify any bounds on the quality of the optimal CS. [9] developed a greedy algorithm, which takes only coalitions up to a certain size into consideration. Its limitation is that it provides no guarantee on the quality of its solutions compared to the actual optimal. Anytime algorithms return an initial solution, and then improve on the quality (and establish better bound gradually) of the solution as they search more of the space. In this context, [3] proposed an anytime algorithm that can establish a bound on quality of the solution, however, the algorithm has to search entire search space, to generate a guaranteed optimal solution[2] and the bounds provided by the algorithm are not valuable for practical use. Based on this concept, [10] proposed another anytime algorithm that can also establish a bound on the quality of solution but employ different searching mechanism and have the same demerits. [11, 12] proposed a state of the art anytime algorithm, IP, but again it has to search entire space in order to generate an optimal solution.

III. BACKGROUND: INTEGER PARTITION GRAPH AND IP ALGORITHM

In [12] the authors proposed an efficient search space representation that can be used for finding the optimal solution efficiently. They called this representation Integer Partition Graph. In this representation, they partitioned the search space p by defining sub-spaces that contain coalition structures that are similar according to the 'integer partitions'[3] of the number of agents. This can be defined by a function F: $\mathfrak{p} \rightarrow \boldsymbol{G}$, where $\boldsymbol{G}$ is the set of integer partition of n. Then they defined a pre-image (or inverse image) of an integer partition G as follows:

$$P_G = F^{-1}[\{G\}].$$

Each pre-image, which represents a sub-space in the integer partition graph, encloses all the coalition structures corresponding to the same integer partition G.

Fig. 1 shows an integer partition graph for 4 agents. We observe that sub-spaces have been categorized into levels, based on the number of parts within the integer partitions. In general, we have n levels, where n is the number of agents. Each level, $\mathfrak{p}_i$ comprises of all the sub-spaces that correspond to an integer partition with i parts.

Given this representation, they computed the Upper Bound[4] (UB) and Lower Bound[5] (LB) in each sub-space P_G as follows: Let L_s be the list of coalitions of size s, and let

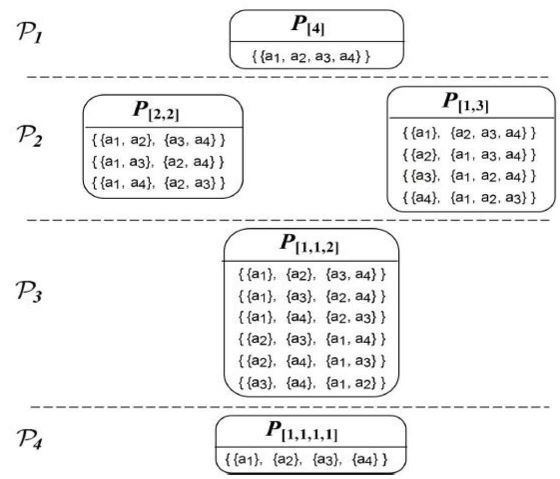

Fig. 1: Example of the integer partition graph for 4 agents [12].

max_s, min_s and avg_s be the maximum, minimum, and average value of the coalition in L_s respectively.
Now given an integer partition G, let T_G be the Cartesian product of the lists $L_s : s \in G$, i.e.

$$T_G = \prod_{s \in G} (L_s)^{G(s)},$$

where G(s) is the multiplicity of s in G.

Now consider the value MAX_G obtained by adding the maximum value of each element (i.e. coalition list) in T_G. Formally, it can be represented as follows:

$$MAX_G = \sum_{s \in G} max_s \times G(s).$$

This value is an upper bound on the best coalition structure in P_G. Now the average value of all the solutions in P_G, denoted by AVG_G, can be computed immediately after scanning the input, by adding the averages of the coalition lists in P_G. If we consider G= $\{g_1, g_2, \ldots, g_{|G|}\}$ as an integer partition, and avg_{gi} as the average of the values of all coalition in L_{gi}, then it can be computed as follows[6]:

$$AVG_G = \sum_{i=1}^{G} avg_{gi}.$$

Furthermore, they argued that it is better to specify AVG_G as lower bound. The reason behind this is that one can prune a lot of search space by improving the LB[7] and average value of a sub-space is usually better than the minimum value.

Two main steps that IP requires in order to search the space using this representation are,

a. Scanning the input in order to compute the bounds (i.e. MAX_G and AVG_G) for every subspace P_G.

[2] i.e. bound=1
[3] Integer partition of n is a multiset of positive integers that add up to exactly n.
[4] UB places an upper limit on the value of the optimal solution, i.e. no coalition structure in a sub-space can have value greater than its UB.
[5] LB places a lower limit on the value of the optimal solution, i.e. the solution at worse will be greater than or equal to this LB.
[6] For proof of this theorem, refer to [12].
[7] Our heuristic (LocalSearch) improves the LB of IP drastically.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

b. Selecting and searching within the remaining sub-spaces—we can apply different selection functions within this step (discussed in next section).

To get the unbiased performance evaluation of IP with other state of the art algorithms, they tested it under different distributions. They used the normal, uniform, and NDCS[8] (Normally Distributed Coalition Structures) input distribution, and benchmarked it against the other state-of-the-art algorithm IDP. The results are shown in fig. 2.

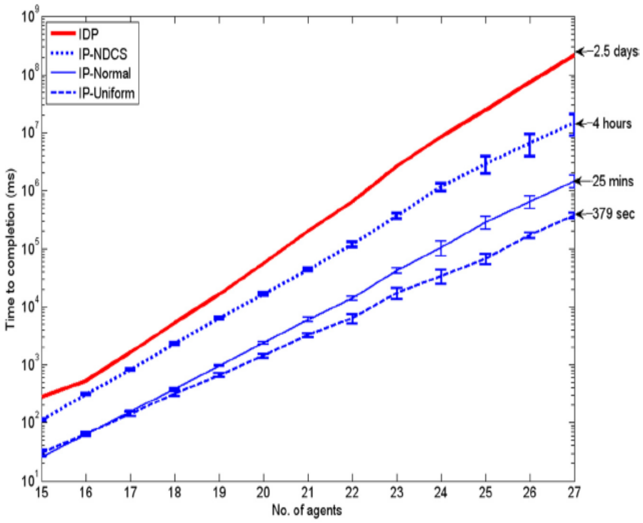

Fig. 2: The time required to find the optimal solution for IDP and IP under NDCS, Normal, and Uniform Distributions [12]. IDP was found to be 15 times slower than IP.

They showed that IP was faster than IDP in finding the optimal coalition structure. Furthermore, they noted that IP was slower in finding the solution in the case of NDCS[9].

IV. PROPOSED HEURISTICS

A. LocalSearch Heuristic

We assume that the input to coalition structure generation algorithm is the value associated to each coalition, v(C), where $C \epsilon 2^A/\{\phi\}$. We further assume that input is given as follows: $C(L_s)\ \forall s \epsilon \{1,2,\ldots..n\}$ and $v(L_s)\ \forall s \epsilon \{1,2,\ldots..n\}$, where $C(L_s)$ is a list containing the coalitions and $v(L_s)$ is a list containing the values of all the coalitions of size s.

Now we define some notations. Let $\max(v(L_s))$ be the maximum value present in a list of value $v(L_s)$. Let MAX consists of memory locations[10], that contain the maximum values (i.e. $\max(v(L_s))$) from each list of values present in G. Furthermore, let V_{Max} be the maximum value present in MAX (i.e. $V_{Max} = \max(MAX)$), L_{Max} be the list of coalition that contains this value V_{Max}, and C_{Max} be the coalition that corresponds to the value V_{Max}.

Like IP algorithm we first scan the value of coalition of size n (called grand coalition), scan the values of coalitions of size 1 (called singleton coalition), and search the level 2 (i.e. $\mathfrak{p}_2$). At this point, we can compute the best solution found so for. Then we run LocalSearch heuristic that computes a good-enough solution.

The pseudo code of the LocalSearch heuristic can be outlined as follows:

Algorithm: LocalSearch()—Scans input, generates CS, and improves the LB of IP.

Input: $C(\boldsymbol{L_s})\ \forall \boldsymbol{s}\ \boldsymbol{\epsilon}\ \{\boldsymbol{1},\boldsymbol{2},\ldots..\boldsymbol{n}\}$, $v(\boldsymbol{L_s})\ \forall \boldsymbol{s}\ \boldsymbol{\epsilon}\ \{\boldsymbol{1},\boldsymbol{2},\ldots..\boldsymbol{n}\}$, set of agents (A = $\{\boldsymbol{a_1},\ldots,\boldsymbol{a_n}\}$), an integer partition (G= $\{\boldsymbol{g_1},\boldsymbol{g_2},\ldots,\boldsymbol{g_{|G|}}\}$).
Output: coalition structure, value of the coalition structure, time required to generate the coalition structure.

1. Set solution= “”, value=0
2. end= |G| *// Size of G*
3. t1=start timer;

   //Loop until we finish finding a valid solution. In each iteration, we pick the maximum possible coalition value from all available coalitions in that sub-space
4. While (end>=1)

   //From step 5 to 7 we load lists into memory, pick maximum value of each list, and store these maximum values in an array MAX
5. Get lists of coalitions, $C(\boldsymbol{L_G})$, from A
6. Get lists of values, $v(\boldsymbol{L_G})$, corresponding to $C(\boldsymbol{L_G})$
7. Get the maximum value present in each list of value and store them in an array MAX, i.e. MAX= [max $(\boldsymbol{v}(\boldsymbol{L_{g1}}))$, max$(\boldsymbol{v}(\boldsymbol{L_{g2}}))$,….. max $(\boldsymbol{v}(\boldsymbol{L_{g|G|}}))$] //pick maximum value from each list of values in G

   //From step 8 to 10, we find the maximum value $\boldsymbol{V_{Max}}$, from MAX array and pick the coalition $\boldsymbol{C_{Max}}$ which corresponds to this value
8. Get element, $\boldsymbol{V_{Max}}$, which has the maximum value in MAX, i.e. $\boldsymbol{V_{Max}}$= max(MAX)
9. Find index of $\boldsymbol{V_{Max}}$ in MAX and find corresponding list, $\boldsymbol{L_{Max}}$ from G, which contains this element $\boldsymbol{V_{Max}}$ *//find out the list which contains this maximum value, $\boldsymbol{V_{Max}}$*
10. Search for the coalition, $\boldsymbol{C_{Max}}$, which has value $\boldsymbol{V_{Max}}$, in corresponding list $\boldsymbol{L_{Max}}$

    //In step 11 and 12, we add $\boldsymbol{V_{Max}}$ and $\boldsymbol{C_{Max}}$ in solution value and solution respectively
11. Value = value+ $\boldsymbol{V_{Max}}$ *//add coalition value*
12. Solution = solution + $\boldsymbol{C_{Max}}$ *//add coalition*

    // From step 13 to 17, we update (except in last iteration) MAX, G, and A
13. If !(end ==1)
14. Update MAX: set all element of MAX to zero, and set |MAX|= |MAX| -1
15. Update G: delete $\boldsymbol{L_{Max}}$ from G, and set |G| = |G| -1
16. Update A: A= A\$\boldsymbol{C_{Max}}$

[8] See appendix A.
[9] Our heuristics are more successful in this case.
[10] Its size is equal to the size of corresponding integer partition that we want to search, i.e., |MAX|= |G|.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

17. End if
18. end =end -1; *//update loop counter*
19. End while
20. t2=stop timer;
21. Return (solution, value, t2-t1)

At start, we pick up the integer partition G, and load its list of coalitions and values in memory (step 5 and 6). Then we find the maximum value from each list and store these values in an array, MAX (step 7). Now we find the maximum value, V_{Max}, from this array and get the coalition list, $\boldsymbol{L}_{Max}$,that contains this value. From this list, we find the coalition, $\boldsymbol{C}_{Max}$ that corresponds to this maximum value (step 8 to 10). Then, we store this value, and the corresponding coalition (step 11 and 12). At end, we update MAX array by decreasing its dimension by one and initializing by zeros, update our agent set[11] which ensure that we generate only the valid coalition structure, and update G by deleting the list, $\boldsymbol{L}_{Max}$, from memory (step 13 to 17). We repeat this process until we finish searching the possible maximum values from all the lists in G, and then return our solution, corresponding value, and searching time.

B. *GreedySearch Heuristic*

This heuristic is greedy because it starts by discovering the coalition that has the highest value among all the input coalitions. Then it finds all possible integer partitions that can go with this value. Afterwards, it chooses integer partition according to the following selection criteria: chooses integer partition that has the highest average utility, chooses integer partition that has the highest UB, and chooses integer partition that has the highest sum of average and UB. Next, we feed these integer partitions to the LocalSearch heuristic. In this way, we guarantee that we can come up with a good solution at low cost.

Now we define some notations. Let $V_{SpaceMax}$ be the highest value among all the input values, $C_{SpaceMax}$ be the coalition which corresponds to the value $V_{SpaceMax}$, $Partition_{SpaceMax}$ encloses all the integer partitions, which contain $|C_{SpaceMax}|$ as an element, and IP_{size} (Where size$\leq |Partition_{SpaceMax}|$) is such an integer partition.

The pseudo code of GreedySearch heuristic can be outlined as follows:

Algorithm: GreedySearch()— Generate solution quickly.

Input: $C(\boldsymbol{L_s})\ \forall \boldsymbol{s} \in \{\boldsymbol{1}, \boldsymbol{2}, \ldots \boldsymbol{n}\}$, $v(\boldsymbol{L_s})\ \forall \boldsymbol{s} \in \{\boldsymbol{1}, \boldsymbol{2}, \ldots \boldsymbol{n}\}$, set of agents ($A = \{\boldsymbol{a_1}, \ldots, \boldsymbol{a_n}\}$), Set of possible Integer Partition ($G = \{\boldsymbol{G_1}, \boldsymbol{G_2}, \ldots, \boldsymbol{G_n}\}$).
Output: solution, value of solution, time required to generate the solution.

1. Set solution[]= "", value[]=0.0, utility[]=0.0 , conspicuousNode[]=0, time[]=0; *//creates 3 instances: [0] for the highest UB, [1] for highest average, and [2] for highest (UB and average)*
2. t1=start timer;

//From step 3 to 7, we find the maximum value in the space and determine all the sub-spaces which contain this value

3. Get lists of coalitions, $C(\boldsymbol{L_g})$, from A
4. Get lists of values, $v(\boldsymbol{L_g})$ corresponding to $C(\boldsymbol{L_g})$
5. Find value, $\boldsymbol{V_{SpaceMax}}$, which is the maximum value among all the values in $v(\boldsymbol{L_g})$
6. Get coalition, $\boldsymbol{C_{SpaceMax}}$, corresponding to $\boldsymbol{V_{SpaceMax}}$
7. Get all integer partitions which can go with $|\boldsymbol{C_{SpaceMax}}|$ as first element and store them in $\boldsymbol{Partition_{SpaceMax}}$, i.e. $\boldsymbol{Partition_{SpaceMax}}$={ [$|\boldsymbol{C_{SpaceMax}}|$, …],…, [$|\boldsymbol{C_{SpaceMax}}|$, ….]} = { [$\boldsymbol{IP_1}$], [$\boldsymbol{IP_2}$], …. }, where $\boldsymbol{partition_{SpaceMax}} \in G$

//From step 8 to 23, we discover the sub-space which can at expectation give us good enough solution

8. end= $|\boldsymbol{Partition_{SpaceMax}}|$, size=1
9. Set conspicuousNode [0]= conspicuousNode [1]=conspicuousNode [2]=$\boldsymbol{IP_1}$;
10. while (size <= end)
11. Iterate through, $\boldsymbol{IP_{size}}$, from second to last element

//we skip first element, as we know that it will be there in every solution

12. compute UB, LB, and UB + LB
13. If utility[0] < UB
14. utility[0] = UB, conspicuousNode[0] = $\boldsymbol{IP_{size}}$
15. end if

//Update the IP which contains highest sum

16. If utility[1] < LB
17. utility[1] = LB, conspicuousNode[1] = $\boldsymbol{IP_{size}}$
18. end if

//Update the IP which contains highest average

19. If utility[2] < UB+LB
20. utility[2] = UB+LB, conspicuousNode[2] = $\boldsymbol{IP_{size}}$
21. end if

//Update the IP which contains highest sum

22. size++; *//Update loop counter*
23. end while

//From step 24 to 30, we call the LocalSearch algorithm with the selected integer partition

24. (solution[0], value[0], time[0]) := **LocalSearch** ($v(\boldsymbol{L_G})$, $C(\boldsymbol{L_G}), \boldsymbol{A}, \textbf{conspicuousNode}[\textbf{0}]$)
25. If (conspicuousNode[1] != conspicuousNode[0])
26. (solution[1], value[1], time[1]) := **LocalSearch** ($v(\boldsymbol{L_G})$, $C(\boldsymbol{L_G}), \boldsymbol{A}, \textbf{conspicuousNode}[\textbf{1}]$)
27. End if *//This step ensures that we are not going through same integer partition twice*
28. If ((conspicuousNode[2] != conspicuousNode[0]) && (conspicuousNode[2] != conspicuousNode[1]))
29. (solution[2], value[2], time[2]):= **LocalSearch** ($v(\boldsymbol{L_G})$, $C(\boldsymbol{L_G}), \boldsymbol{A}, \textbf{conspicuousNode}[\textbf{2}]$)
30. End if *//This step ensures that we are not going through same integer partition twice*
31. t2=stop timer;
32. Return (solution[], value[], t2-t1)

[11] This step is very crucial and is required to save resources. Further details can be found in [4].

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

We start by finding the maximum value among all the input values (step 3 to 5). Then we find the sub-spaces that contain this maximum value (step 6 to 7). Afterwards, we find the utility (in terms of highest UB, LB and sum of UB and LB) of each such sub-space and choose the sub-spaces that give us the highest utility (step 8 to 23). Then, we search within these sub-spaces by using the LocalSearch heuristic (step 24 to 30), and return the solution.

C. *Selection of a Sub-space in the IP Algorithm*

We assume that we want to find a good-enough[12] solution and we have constraint on the search time. We can make a reasonable selection according to the requirements, by choosing a sub-space according to its normalized size, UB, and LB. For instance, given constraint on searching time, we can pick a sub-space that has the highest (UB + 1/Size) rather than going for a sub-space that has the highest UB. In the former case, we can generate solution much quickly because it contains the small amount of possible solutions. Whereas, in the latter case, it can contain millions of possible solutions and we might not have enough time to search them. Hence, given such priorities, we can choose sub-spaces that can generate required solution more efficiently than the other ones.

We implemented the following important heuristics for selecting a sub-space: Select sub-space that has the highest UB, highest LB, highest (UB+LB), highest (UB + 1/size[13]), highest(LB+1/size), highest ((UB+LB) +1/size), lowest ((UB+LB) +1/size), highest ((UB-LB) +1/size), lowest ((UB-LB) +1/size), and smallest size.

V. ANALYSIS AND RESULTS

In this section, we empirically evaluate our heuristics. We used Java JDK 1.6 as a development language and an Intel 3.2 GHZ dual core PC with 3GB of RAM for running our experiments.

A. *LocalSearch Heuristic*

We plug-in the code of LocalSearch heuristic in the IP algorithm and recorded the algorithm's performance for different number of agents (from 8 to 22). Furthermore, we used the standard instances of the coalition structure generation problem[14].

In the case of NDCS distribution, the average results obtained by running the algorithm for 50 times are shown in fig. 3. It is clear that LocalSearch heuristic is able to return greater than 80% optimal solutions for 8 to 15 agents, and greater than 75% solutions for 16 to 22 agents. In the lower plot, we observe that the increase in the LB* (optimal LB computed by IP while scanning the input and searching the first two layers) is between 5-10%. Furthermore, the total time taken by the IP algorithm is nearly zero for 8 to 15 agents and is less than 400ms for 16 to 22 agents. It is worthy to note that in case of 22 agents, this heuristic returns a 75% optimal solution (with 8% increase in the LB*) in 300ms which is very small. This is because; we are not exploring all possible solutions of the search space, which reduces the exponential nature of the problem. In fact, the complexity of the LocalSearch heuristic depends on the number of possible integer partition of n, (where n is the number of agents) and is independent of the number of possible solutions in the entire space.

Similar results were obtained in the case of normal distribution (not shown), where it returns greater than 95% optimal solutions for 8 to 15 agents and greater than 92% optimal solutions for 16 to 22 agents. We observe that the increase in the LB* is less than 4%. Furthermore, the time taken to return solutions is the same as in the NDCS case.

The results were not promising (not shown) for uniform distribution. For this distribution, the increase in the LB* is less than 1% when number of agents are less than 14, and is zero when the number of agents increases.

Our heuristic gives better results in the case of NDCS distribution, than normal and uniform distributions. The reason is that, in the NDCS case coalition values have more spread (due to the high sigma value)[15] as compared to normal and uniform cases; and LocalSearch heuristic can easily pick these values. For the normal distribution this spread is small (as sigma value lies between 0 and 1)[16], so increase in LB* is smaller as compared to the NDCS case. The bad performance of LocalSearch heuristic in case of uniform distribution comes from the fact that IP finds 95 to 99% optimal solution in the second level[17], while scanning the input.

Now we show how this heuristic improves the anytime property of the IP algorithm. For this purpose, we observe the behaviour of the heuristic while it visits each sub-space. To this end, we assume that we have 15 agents and values have been drawn from the NDCS distribution. Furthermore, we want to find a solution which is 85% optimal. It is worthy to note that algorithm will stop only if it is successful in finding the required optimal solution or it has visited all the sub-spaces. The behaviour of the heuristic is shown in fig. 4. In fig. 4, the percent increase in LB (in lower plot) refers to $(\frac{LB\_Real_G - AVG_G}{AVG_G}) * 100$, where LB_Real_G is the solution computed by the LocalSearch heuristic in a particular sub-space G. The solid line (in the upper plot) shows that the solution is 77% optimal before the IP algorithm calls the LocalSearch heuristic. We record the results after LocalSearch heuristic visits each sub-space (dotted line in the upper plot). Note that the solution becomes 85% optimal (i.e. $x_2 - x_1 =$

[12] Solution with bound > 1
[13] Size has been normalized with respect to the largest size in the space.
[14] See appendix A.
[15] See appendix A.
[16] We run our heuristic wth $\sigma = 0.1$.
[17] In fact, in uniform distribution, coalitions of larger size have more value as compared to the smaller ones; hence searching the second layer returns the 95 to 99% optimal solution. See appendix A for more information.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

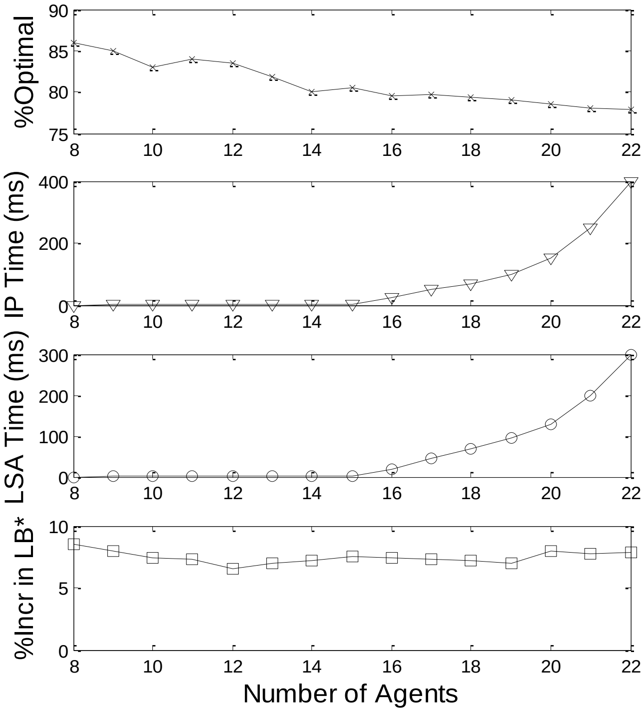

Fig. 3: LocalSearch heuristic for the NDCS distribution. The ‘time (ms) taken by LSA’ refers to the time LocalSearch heuristic took to compute the solution, and the ‘time (ms) taken by IP’ refers to the time IP algorithm took to scan the input, search the second level, and run the LocalSearch or GreedySearch heuristic to completion[18].

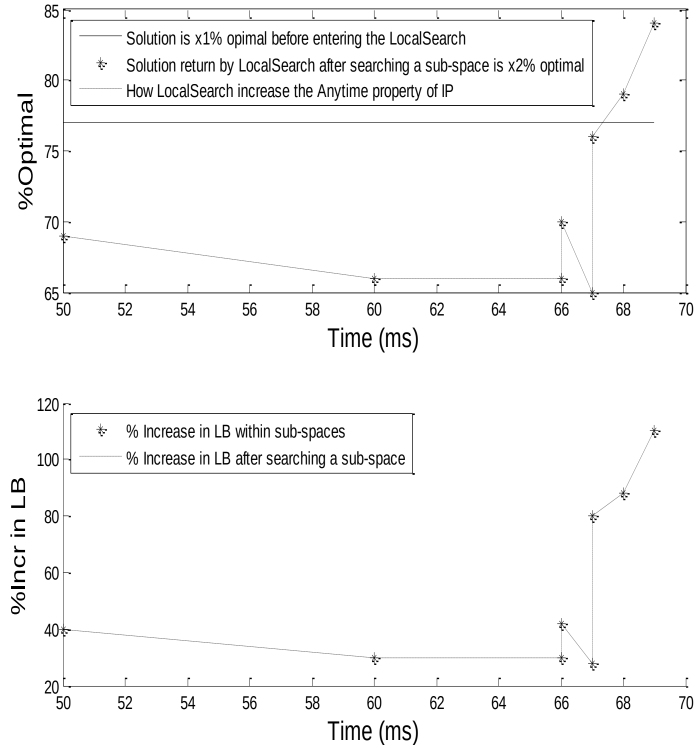

Fig. 4: How LocalSearch heuristic improves the anytime property of the IP algorithm.

8%, which corresponds to the increase in the solution quality) after visiting a few sub-spaces and then algorithm stops and returns the solution. This behaviour shows that the LocalSearch heuristic improve the anytime property of the IP algorithm.

Note that this heuristic increases the solution quality of the IP algorithm as well. Moreover, the percent increase in the solution quality is at least equal to the percent increase in the LB*[19].

B. GreedySearch Heuristic

We plug-in the GreedySearch heuristic in the IP algorithm, run the algorithm for 15 to 27 agents, stop it when the GreedySearch heuristic finishes finding a solution, and record the results. Furthermore, we run our algorithm for 50 times, and reported the average results. The results in the case of NDCS distribution are shown in fig. 5.

Fig. 5 shows that the GreedySearch heuristic is able to find 70 to 75% optimal solutions in less than 400ms. Although the increase in LB* is between 2-4%, but it is a significant improvement, as time taken by it to return a solution is very small. Similar results were observed in the case of normal distribution. Furthermore, for uniform distribution, the results were not statistically significant (The reason is the same, as discussed before).

Note that for 27 agents, this heuristic returns a good-enough solution in 410ms that is 10 times less as compared to the time taken by the IP algorithm (5000ms – 410ms) to scan the input and search the second level. It is worth noting that, for 27 agents and NDCS distribution, finding an optimal solution can take many hours (or days) as shown in fig. 1. Hence, one may prefer a good solution over optimal for setting where one has constraint over time (for instance, in real-time applications).

C. Selection Functions for IP Algorithm

We assume that we want to find a 92% optimal solution[20]. We recorded the performance of the IP algorithm for 15 to 21 agents against uniform, normal, and NDCS distribution. Furthermore, we run our algorithm 70 times for 15 to 19 agents and 50 times for 20 to 21 agents, and recorded the average results. The results in the case of NDCS distribution are shown in fig. 6[21].

[18] The terms ‘time (ms) taken by LSA’ and ‘time (ms) taken by IP’ have the same meaning for all figures.

[19] For proof, refer to appendix B.

[20] We take this value as an example. Any other value less than 100% can be assumed. Furthermore, nearly similar results were observed for 85% optimal solution.

[21] Similar results were observed for normal and uniform distribution.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

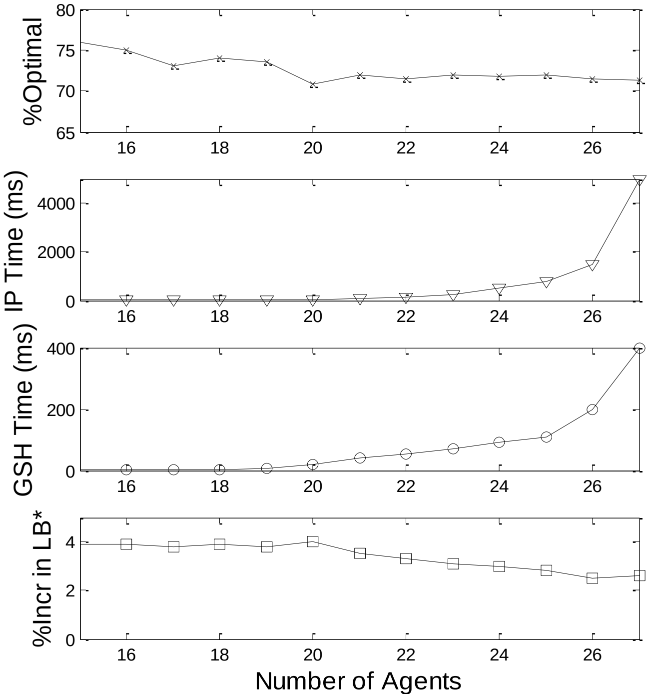

Fig. 5: GreedySearch heuristic for the NDCS distribution[22].

Fig. 6 shows that, the following sub-spaces are found good in generating the required solution:

- Sub-spaces having the highest LB with the smallest size return the solution about 30 to 300% faster than the other ones. The reason is that, they contain overall high values of the coalitions; hence, after searching a few solutions, we may find the desired optimal solution. Furthermore, it is able to return a good solution faster than others due to its smaller size.
- Sub-spaces having the highest UB with the smallest size return the solution about 40 to 200% fasters then the rest ones (excluding the highest LB +1/Size) one. The reason is that the highest UB ensures to generate good solution and smallest size ensures that it can be generated much quickly.
- Sub-spaces having the smallest size show same behaviour as that of the highest (UB+1/Size) one. The reason is that, they can return solution much quickly due to their smallest size.

Furthermore, some sub-spaces, such as the one having lowest ((UB-LB) + 1/Size) are more than 100% slower in generating the solution. The reason is that, they have large size and low values of the coalitions. From the results, we can conclude that the selection of a particular sub-space has significant effect on the time required to find a good solution.

[22] In fig. 5, the 'time (ms) taken by GSA' refers to the time GreedySearch heuristic took to compute the solution

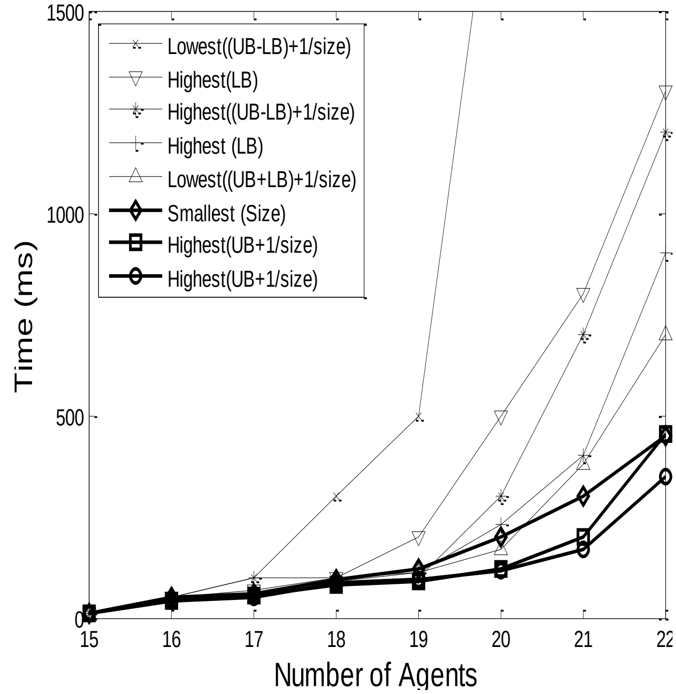

Fig. 6: The time required to generate the 92% optimal solution for different sub-spaces against NDCS distribution.

VI. CONCLUSION AND FUTURE WORK

Coalition formation is an advanced research area within multi-agent systems nowadays. Generally, the goal of the coalition structure generation activity is to maximize the social welfare by finding the optimal coalition structure, but exponential nature of the solution space does not allow making exhaustive search for the optimal solution. Hence, we may prefer a good solution over an optimal one in settings where we have constraints on execution time and memory. From this line of research, we proposed two new heuristics for coalition structure generation.

This paper advances the state of the art in the followings:

- First, we proposed a novel heuristic, namely LocalSearch for coalition structure generation and empirically show that it generates good-enough solution in short time. Furthermore, it improves the anytime property, lower bound, and solution quality of the IP algorithm. The increased lower bound can prune a major portion of the exponential search space without going into the space.
- Second, we proposed a greedy heuristics, namely GreedySearch for finding a good-enough solution, without going fully to any of the sub-space, in settings where we have a large number of agents (>20).
- Third, we implemented different heuristics for selecting a sub-space in the IP algorithm proposed by [12]. We show that, in order to find a good solution (as opposed by optimal), the selection of a particular sub-space in the IP

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

algorithm has significant effect on its performance, in term of the time required to return the solution.

As a future work, we would like to integrate our work with recommender systems [13, 14]. There has been no work in literature that uses coalition formation among agents for solving recommender systems problems. If we divide users (or items) into distinct clusters, then our algorithm can be used in finding the most relevant users (or items). A K nearest neighbour based collaborative filtering algorithm can be used for generating recommendations. Furthermore, proposed algorithm can be helpful in distributed recommender system.

REFERENCES

[1] Caparrós A., Hammoudi A., and Tazdaït T. (2004). On Coalition Formation with Heterogeneous Agents. Working Papers 2004.70, Fondazione Eni Enrico Mattei

[2] Larson H. S., Sandholm T. W. (1999). Anytime coalition structure generation: an average case study, Proceedings of the third annual conference on Autonomous Agents, p.40-47, Seattle, Washington, United States

[3] Sandholm, T. W., Larson, K., Andersson, M., Shehory, O., and Tohme, F. (1999). Coalition structure generation with worst case guarantees. Artificial Intelligence, 111(1–2):209–238.

[4] Rahwan, T. (2007) Algorithms for Coalition Formation in Multi-Agent Systems. PhD thesis, University of Southampton.

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

[6] Rothkopf, M. H., Pekec, A., and Harstad, R. M. (1995). Computationally manageable combinatorial auctions. Management Science, 44(8):1131–1147.

[7] Rahwan T. and Jennings N. R. (2008b). An improved dynamic programming algorithm for coalition structure generation. In Proc 7th Int Conf on Autonomous Agents and Multi-Agent Systems (AAMAS-08), Estoril, Portugal , volume 3, pages 1417–1420.

[8] Sen, S. and Dutta, P. (2000). Searching for optimal coalition structures. In Proceedings of the Fourth International Conference on Multiagent Systems, pages 286–292.

[9] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. Artificial Intelligence, 101(1–2):165–200, 1998.

[10] V. D. Dang and N. R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pages 564–571, 2004.

[11] T. Rahwan, S. D. Ramchurn, V. D. Giovannucci, V. D. Dang, and N. R. Jennings. Anytime optimal coalition structure generation. In Proceedings of the Twenty second Conference on Artificial Intelligence (AAAI-07), pages 1184–1190, 2007.

[12] T. Rahwan, S. D. Ramchurn, A. Giovannucci, and N. R. Jennings (2009). An Anytime Algorithm for Optimal Coalition Structure Generation. *Journal of Artificial Intelligence Research (**JAIR**)*. 34, Pages 521-567.

[13] Mustansar Ali Ghazanfar, and Adam Prugel-Bennett, *"A Scalable, Accurate Hybrid Recommender System"* , In the 3rd International Conference on Knowledge Discovery and Data Mining (WKDD 2010), 9-10 Jan 2010, Thailand.

[14] Mustansar Ali Ghazanfar, and Adam Prugel-Bennett, *"An Improved Switching Hybrid Recommender System using Naive Bayes Classifier and Collaborative Filtering",* In the 2010 IAENG International Conference on DataMining and Applications (ICDMA10), Hong Kong, 17-19 March, 2010

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

APPENDIX A

For benchmarking the coalition structure generation algorithms, the standard instances of the input value distribution have been defined as follows [2]:

Normal Distribution: $v(C) = |C|$ x p where $p \sim N(\mu, \sigma^2)$, $\mu = 1$ *and* $\sigma = 0.1$

Uniform Distribution: $v(C) = |C|$ x p where $p \sim U(a, b)$, $a = 0$ *and* $b = 1$

Sub-additive: $v(C) \leq v(C') + v(C'')$ where $C = C' \cup C''$ and $v(C)$ is uniform as above. (In this case the singleton coalitions form the optimal structure)

Super-additive: $v(C) \geq v(C') + v(C'')$ where C', C'' *and* $v(C)$ are as defined above (In this case the grand coalition is the optimal structure).

The validity of uniform and normal instances has been questioned by [13], where the authors claimed that these instances generate biased results: *"we analytically show that any CSG problem with an input defined according to distributions of coalition values based on the size of the coalitions (such as the Normal and Uniform distributions above) will generate biased results"* [13].

In fact this was the main reason why in the case of uniform and normal distribution, our Heuristics (LocalSearch and GreedySearch) did not showed much improvement in the LB* computed by the IP algorithm.

NDCS (Normally Distributed Coalition Structures): This instance of the input distribution has been defined by [12], and is well suited for the coalition structure generation problems. This instance is defined as follows:

$v(C) \sim N(\mu, \sigma^2)$, where $\mu = |C|, \sigma = \sqrt{|C|}$

In this distribution, the value of every possible coalition structure is independently drawn from the same normal distribution

Furthermore, for this distribution, our heuristics showed significant improvement in LB* computed by the IP algorithm.

APPENDIX B

This comes from the fact that **LB* = max ($\boldsymbol{AVG_G^*}, \boldsymbol{V(CS')}$)** where $V(CS')$ is the best solution found in levels $\mathfrak{p}_1, \mathfrak{p}_2, and\ \mathfrak{p}_n$. Let LB_Real^* be the best solution found by the LocalSearch heuristic. We compute the percent increased in the LB* as follow: % $\boldsymbol{Increase\ in\ the\ LB^*} = \left(\frac{\boldsymbol{LB_{Real}}^* - \boldsymbol{LB}^*}{\boldsymbol{LB}^*}\right)$ ***** 100. We can easily conclude from this equation that the % increase in the solution quality is at least equal to this % increase in the LB* (in case we have LB*= $V(CS')$) and can be greater than this % increase in the LB* (in case we have LB* = AVG_G^*).

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)