An Improved Dynamic Programming Algorithm for Coalition Structure Generation

(Short Paper)

Talal Rahwan and Nicholas R. Jennings
School of Electronics and Computer Science, University of Southampton, Southampton, UK.
{tr, nrj}@ecs.soton.ac.uk

ABSTRACT

Forming effective coalitions is a major research challenge in the field of multi-agent systems. Central to this endeavour is the problem of partitioning the set of agents into exhaustive and disjoint coalitions such that the social welfare is maximized. This *coalition structure generation problem* is extremely challenging due to the exponential number of partitions that need to be examined. Specifically, given n agents, there are $O(n^n)$ possible partitions. To date, the only algorithm that can find an optimal solution in $O(3^n)$ is the Dynamic Programming (DP) algorithm, due to Rothkopf et al. However, one of the main limitations of DP is that it requires a significant amount of memory. In this paper, we devise an Improved Dynamic Programming algorithm (IDP) that is proved to perform fewer operations than DP (e.g. 38.7% of the operations given 25 agents), and is shown to use only 33.3% of the memory in the best case, and 66.6% in the worst.

1. INTRODUCTION

Coalition formation, the process by which a group of agents come together and agree to coordinate and cooperate in the performance of a set of tasks, is an important form of interaction in multi-agent systems.To date, a number of coalition formation algorithms have been developed to determine which of the many potential coalitions should actually be formed. To do so, they typically calculate a value for each coalition, known as the *coalition value*, which provides an indication of the expected outcome that could be derived if that coalition was formed. Then, having computed all the coalition values, the decision about the optimal coalition(s) to form can be taken.

One of the main bottlenecks that arise in this formation process is that of *coalition structure generation (CSG)*. Specifically, given the value of every possible coalition, the CSG problem involves partitioning the set of agents into exhaustive and disjoint coalitions so as to maximize the social welfare. Such a partition is called a *coalition structure*. In this context, it is usually assumed that every coalition performs equally well, given any coalition structure containing it (i.e. the value of a coalition does not depend on the actions of non-members). Such settings are known as *characteristic function games (CFGs)*, where the value of a coalition is given by a *characteristic function*. Many (but clearly not all) real-world multi-agent problems happen to be CFGs [4, 5]. In this context, a number of algorithms have been developed to solve the CSG problem, which we classify into the following broad classes:

Cite as: An Improved Dynamic Programming Algorithm for Coalition Structure Generation (Short Paper), T. Rahwan and N. R. Jennings, *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. XXX-XXX.

1. **Dynamic programming:** Algorithms in this class have been developed for solving the complete set partitioning problem [8] and the winner determination problem [3], both of which can be viewed as being analogous to the CSG problem.[1] Note that both of these algorithms use basically the same technique and, therefore, have the same computational complexity. Thus, throughout this paper, we do not distinguish between them, and consider both to be the state-of-the-art dynamic programming (DP) algorithm. The main advantage of this algorithm is that, given n agents, it guarantees to find an optimal solution in $O(3^n)$ time. However, the downside is that this algorithm only produces a solution when it has completed its entire execution. Another limitation of this approach is that it requires maintaining three tables in memory, including the input, that each have $2^n - 1$ entries.

2. **Heuristics:** A number of heuristics have been developed to solve the CSG problem. Shehory and Kraus, for example, proposed a greedy algorithm that puts constraints on the size of the coalitions that are taken into consideration [7]. Sen and Dutta, on the other hand, use an order-based genetic algorithm [6]. Although these algorithms return solutions relatively quickly, they do not provide any guarantees on finding the optimal. In fact, their solutions can always be arbitrarily worse than the optimal, and even if they happen to find an optimal solution, it is not possible to verify this fact.

3. **Anytime optimal algorithms:** These algorithms work is by generating initial solutions that are guaranteed to be within a bound from the optimal, and then improve on the quality of their solutions, and establish progressively better bounds, as they search more of the search space, until an optimal solution is found. A number of algorithms have been developed to generate solutions anytime [4, 1], but these can only find an optimal solution once the entire space has been exhaustively searched. More recently, however, Rahwan et al. developed a state-of-the-art anytime algorithm that is significantly faster than the other anytime algorithms [2]. However, the algorithm is still $O(n^n)$, meaning that it might, in the worst case, end up searching the entire space.

In this paper, we are particularly interested in the first approach because it is the one with the lowest worst case complexity that is guaranteed to find the optimal solution. In contrast, the second approach might never find the optimal, and the third approach might still end up searching the entire space.

Against this background, our contribution lies in developing an Improved Dynamic Programming algorithm (called IDP) that performs fewer operations, and requires less memory, compared to the

[1] This is because they both involve partitioning a set of elements into subsets based on the weights that are associated to every subset.

current state of the art DP algorithm. For example, given 25 agents, IDP performs only 38.7% of the operations, and requires 33.3% of the memory in the best case and 66.6% in the worst.

Throughout this paper, we will denote by A the set of agents, n the number of agents, v the input table (i.e. $v[C]$ is the value of coalition C), and $V(CS)$ the value of coalition structure CS. Moreover, the terms *"coalition structure"* and *"solution"* will be used interchangeably. The remainder of this paper is structured as follows. Section 2 provides a detailed analysis of how DP works. Section 3 presents IDP, and compares it with DP. Finally, section 4 concludes.

2. THE DP ALGORITHM

The DP algorithm is shown in figure 1. The way this algorithm works is by computing two tables, namely f_1 and f_2, each with an entry for every possible coalition. In more detail, for every coalition $C \subseteq A$, the algorithm computes $f_1[C]$ and $f_2[C]$ as follows. First, it evaluates all the possible ways of splitting C in two, and compares the highest evaluation with $v[C]$ to determine whether it is beneficial to split C. If so, then the best splitting (i.e. the one with the highest evaluation) is stored in $f_1[C]$, and its evaluation is stored in $f_2[C]$. Otherwise, the coalition itself is stored in $f_1[C]$ and its value is stored in $f_2[C]$ (see step 2).[2] Note that every splitting $\{C', C''\}$ is evaluated as follows: $f_2[C'] + f_2[C'']$. This implies that, in order to evaluate the possible splittings of a coalition C, the algorithm first needs to compute f_2 for the possible subsets of C. Based on this, the algorithm does not evaluate the splittings of the coalitions of size s until it has finished computing f_2 for the coalitions of sizes $1, \ldots, s-1$ (see step 1 and the first line of step 2). Figure 2 shows an example of how f_1 and f_2 are computed given $A = \{1, 2, 3, 4\}$.[3]

Input: $v[C]$ for all $C \subseteq A$. If no $v[C]$ is specified then $v[C] = 0$.
Output: the optimal coalition structure, CS^*.
1. For all $i \in \{1, \ldots, n\}$, set $f_1[\{a_i\}] := \{a_i\}$, $f_2[a_i] := v[\{a_i\}]$
2. For $s := 2$ to n, do:
  For all $C \subseteq A$ such that $|C| = s$, do:
  2.1. $f_2[C] := max\{f_2[C'] + f_2[C \backslash C'] : C' \subset C$ and $1 \leq |C'| \leq 1/2\,|C|\}$
  2.2. If $f_2[C] \geq v[C]$, then set $f_1[C] := C^*$ where C^* maximizes the right hand side of the equation in step 2.1.
  2.3. If $f_2[C] < v[C]$, then set $f_1[C] := C$, and $f_2[C] := v[C]$.
3. Set $CS^* := \{A\}$.
4. For every $C \in CS^*$, do:
  If $f_1[C] \neq C$, then:
  4.1. Set $CS^* := (CS^* \backslash \{C\}) \cup \{f_1[C], S \backslash f_1[C]\}$.
  4.2. Goto 4 and start with new CS^*.

Figure 1: The DP algorithm for coalition structure generation.

Once f_1 and f_2 are computed for every coalition, the optimal coalition structure, denoted CS^*, can be computed recursively (steps 3 and 4 in figure 1). In our example, this is done by first setting $CS^* = \{1, 2, 3, 4\}$. Then, by looking at $f_1[\{1, 2, 3, 4\}]$, we find that it is more beneficial to split $\{1, 2, 3, 4\}$ into $\{1, 2\}$ and $\{3, 4\}$. Similarly, by looking at $f_1[\{1, 2\}]$, we find that it is more beneficial to split $\{1, 2\}$ into $\{1\}$ and $\{2\}$, and by looking at $f_1[\{3, 4\}]$, we find that it is more beneficial to keep $\{3, 4\}$ as it is. The optimal coalition structure is, therefore, $\{\{1\}, \{2\}, \{3, 4\}\}$.

Having described how DP operates, we will now analyse its performance. To this end, based on our previous experience in this domain, we have observed that the elimination of any redundancy (if it exists) can lead to significant performance improvements. With this in mind, we looked at the *coalition structure graph* [4], which consists of a number of nodes and a number of edges connecting these nodes. Specifically, every node in the graph represents a coalition structure, and every edge represents the merging of two coalitions into one (when followed downwards) and the splitting of one coalition into two (when followed upwards). The nodes are categorized, based on the number of coalitions in each node, into levels $L_i : i \in \{1, 2, \ldots, n\}$ such that L_i contains the nodes (i.e. coalition structures) that contain i coalitions. Figure 3 shows an example of 4 agents.[4]

size	Coalition	The evaluations that are performed before setting f_1 and f_2	f_1	f_2
1	{1}	v[{1}]=30	{1}	30
	{2}	v[{2}]=40	{2}	40
	{3}	v[{3}]=25	{3}	25
	{4}	v[{4}]=45	{4}	45
2	{1,2}	v[{1,2}]=50 f2[{1}]+f2[{2}]=70	{1} {2}	70
	{1,3}	v[{1,3}]=60 f2[{1}]+f2[{3}]=55	{1,3}	60
	{1,4}	v[{1,4}]=80 f2[{1}]+f2[{4}]=75	{1,4}	80
	{2,3}	v[{2,3}]=55 f2[{2}]+f2[{3}]=65	{2} {3}	65
	{2,4}	v[{2,4}]=70 f2[{2}]+f2[{4}]=85	{2} {4}	85
	{3,4}	v[{3,4}]=80 f2[{3}]+f2[{4}]=70	{3,4}	80
3	{1,2,3}	v[{1,2,3}]=90 f2[{1}]+f2[{2,3}]=95 f2[{2}]+f2[{1,3}]=100 f2[{3}]+f2[{1,2}]=95	{2} {1,3}	100
	{1,2,4}	v[{1,2,4}]=120 f2[{1}]+f2[{2,4}]=115 f2[{2}]+f2[{1,4}]=110 f2[{4}]+f2[{1,2}]=115	{1,2,4}	120
	{1,3,4}	v[{1,3,4}]=100 f2[{1}]+f2[{3,4}]=110 f2[{3}]+f2[{1,4}]=105 f2[{4}]+f2[{1,3}]=105	{1} {3,4}	110
	{2,3,4}	v[{2,3,4}]=115 f2[{2}]+f2[{3,4}]=120 f2[{3}]+f2[{2,4}]=110 f2[{4}]+f2[{2,3}]=110	{2} {3,4}	120
4	{1,2,3,4}	v[{1,2,3,4}]=140 f2[{1}]+f2[{2,3,4}]=150 f2[{2}]+f2[{1,3,4}]=150 f2[{3}]+f2[{1,2,4}]=145 f2[{4}]+f2[{1,2,3}]=145 f2[{1,2}]+f2[{3,4}]=150 f2[{1,3}]+f2[{2,4}]=145 f2[{1,4}]+f2[{2,3}]=145	{1,2} {3,4}	150

Figure 2: Example of how the DP algorithm computes f_1 and f_2, given $A = \{1, 2, 3, 4\}$.

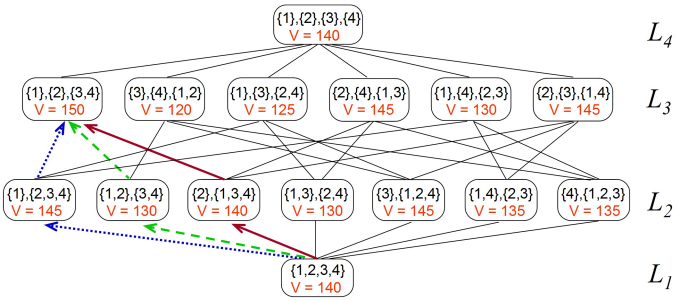

Figure 3: The coalition structure graph of 4 agents.

Looking at this graph enables us to visualize how DP works. Basically, given a coalition structure CS, the splitting of a coalition $C \in CS$ into $\{C', C''\}$ can be seen as a movement from the node that contains CS to the node that contains $(CS \backslash C) \cup \{C', C''\}$. The way DP works is by evaluating every possible movement in the graph (steps 1 and 2 in figure 1), and then, starting at the bottom node and moving upwards through a series of connected nodes (which we call a *"path"*) until an optimal node is reached, after which no movement is beneficial (steps 3 and 4 in figure 1). The dashed path in figure 3 shows how DP found the optimal in our previous example (which was shown in figure 2).

To be still more precise, the way DP works can be described on the coalition structure graph as follows. For every coalition C of size

[2]The special case in which C contains only one agent is dealt with seperately (see step 1); in this case, we always have: $f_1[C] = C$ and $f_2[C] = v[C]$.

[3]Note that, in order to store a splitting C', C'' of a coalition C, it is sufficient to store either C' or C'' in $f_1[C]$ (because knowing one of them is sufficient to know the other). In figure 2, however, both of them are shown in f_1 to make the example easier to understand.

[4]For illustrative purposes, the figure also shows the value of every coalition structure, taken from our previous example which is shown in figure 2.

$s \in \{2, \ldots, n\}$, the algorithm determines whether it is beneficial to make a movement that involves the splitting of C. This is done by first evaluating the possible splittings of C (using f_2) and then comparing the best one, denoted $\{C^*, C \backslash C^*\}$, with $v[C]$. Now if it is more beneficial to split C, then C^* is stored in $f_1[C]$. This indicates that, whenever a node CS is reached that contains C, the best movement (out of all the movements that involve splitting C) is the one that leads to $(CS \backslash C) \cup \{C^*, C \backslash C^*\}$. On the other hand, if it is more beneficial not to split C at all, then C is stored in $f_1[C]$. This indicates that, whenever a node is reached that contains C, it is not beneficial to make any movement that involves splitting C. A key point to note, here, is that the evaluation of a particular movement is done by taking into consideration the best movements that will follow this one. This is captured by the way $f_2[C]$ is calculated, which uses $f_2[C']$ for all $C' \subset C$. In figure 3, for example, DP did not move from the node containing $\{1, 2, 3, 4\}$ to that containing $\{\{3\}, \{1, 2, 4\}\}$ (which has a value of 145). Instead, it moved to the one containing $\{\{1, 2\}, \{3, 4\}\}$ (which has a value of 130). The way DP made this decision was based on the movements that will follow each of the aforementioned ones. As can be seen, the movement that DP took led to the optimal coalition structure (which has a value of 150), while the other movement would not lead to any coalition structure with a value greater than 145.

From this visualization, it is apparent that, any coalition structure containing more than two coalitions has more than one path leading to it. For example, starting from the bottom node, one could reach $\{\{1\}, \{2\}, \{3, 4\}\}$ through three different paths, which are shown in figure 3 using dotted, dashed, and bold lines. To this end, we note that, whenever there are multiple paths leading to the optimal, DP does not have any preference on which of these is taken. This can be seen in step 2.2 in figure 1, where $f_1[C]$ is set to the splitting that maximizes the right hand side of the equation in step 2.1, (i.e. the one that has the maximum evaluation). This implies that, whenever multiple splits have the same evaluation, it doesn't matter which one of them is chosen, as long as it maximizes the right hand side of 2.1. In figure 2, for example, $f_1[\{1, 2, 3, 4\}]$ was set to the splitting $\{\{1, 2\}, \{3, 4\}\}$ because this had *one of* the highest evaluations (which is 150). However, we could have set it to $\{\{1\}, \{2, 3, 4\}\}$ instead (since this splitting also has an evaluation of 150), and from $f_1[\{2, 3, 4\}]$, we would have found that it is more beneficial to split $\{2, 3, 4\}$ into $\{2\}$ and $\{3, 4\}$. As a result, the same optimal (i.e. $\{\{1\}, \{2\}, \{3, 4\}\}$) would have been found.

Now suppose that, when determining the best of all the movements that involve splitting $\{2, 3, 4\}$, the following splitting was never taken into consideration: $\{\{2\}, \{3, 4\}\}$. This corresponds to the removal of the edge that connects $\{\{1\}, \{2, 3, 4\}\}$ with $\{\{1\}, \{2\}, \{3, 4\}\}$. In this case, DP will no longer consider the movement from $\{1, 2, 3, 4\}$ to $\{\{1\}, \{2, 3, 4\}\}$ as one of the options that lead to the optimal. This is because, as far as DP can tell, the best coalition structure that can be reached by making this movement is one that has a value of 145. Note that, although DP will no longer be able find the optimal through the dotted path, it can still do so through any of the remaining paths (i.e. the dashed or the bold ones). In fact, as long as there is one path leading to the optimal, DP will be able to find that optimal (because the evaluation of that path is not affected by the removal of the other paths). This implies that finding the optimal might not require evaluating every possible splitting of every possible coalition. What would be desirable, then, is to be able to avoid the evaluation of as many splittings as possible, and still be guaranteed to keep at least one path for every possible solution. In the next subsection, we present an Improved Dynamic Programming algorithm (which we call IDP) which does exactly that. Moreover, we show how the memory requirements can be reduced without affecting its performance.

3. THE IDP ALGORITHM

The main idea behind IDP is to avoid the evaluation of as many splittings as possible, without losing the guarantees of finding the optimal coalition structure. As mentioned above, this corresponds to the removal of as many edges from the coalition structure graph as possible, without removing all the paths that lead to the optimal node (where a path to a node $CS : |CS| = s$ is defined as a series of connected nodes $\langle n_1, n_2, \ldots, n_s \rangle$ such that $\forall i \in \{1, 2, \ldots, s\}, n_i \in L_i$ and $n_s = CS$). In order to do so, we must guarantee that, for every node in the graph, there exists at least one path leading to that node.[5] This can be done by proving that, for every node in $L_s : s \in \{2, \ldots, n\}$, there exists another node in L_{s-1} that is connected to that node. In other words, for every coalition structure CS containing s coalitions (where $s \in \{2, \ldots, n\}$), we need to prove that there exists another coalition structure containing $s - 1$ coalitions that is connected to CS.

To this end, let $E^{s's''}$ be the set of all the edges that involve the splitting of a coalition of size $(s' + s'')$ into two coalitions of sizes s' and s'', where $s' \leq s''$. Moreover, let E^* be a subset of edges that is defined as follows:

$$E^* = (\bigcup_{s'' \leq n-(s'+s'')} E^{s's''}) \cup (\bigcup_{s'+s''=n} E^{s's''}) \qquad (1)$$

With these definitions in place, we can now present the main theorem underpinning IDP.

Theorem 1. For every coalition structure CS containing s coalitions ($s \in \{2, \ldots, n\}$), there exists an edge in E^* that connects CS with another coalition structure containing $s - 1$ coalitions.

Proof. For the coalition structures that contain two coalitions, we need to show that each one of them is connected with the coalition structure that contains one coalition (which is the grand coalition).[6] This is easily shown since E^* includes all the edges that involve the splitting of the grand coalition in two (i.e. $E^{s's''} : s' + s'' = n$). What is left, then, is to show that, for every coalition structure CS that contains more than two coalitions, there exists two coalitions $C', C'' \in CS$ (where C' contains s' agents and C'' contains s'' agents), such that $E^{s's''} \in E^*$. This can be done by proving that the following condition holds:

$$\exists C', C'' \in CS : |C'| = s', |C''| = s'', s'' \leq n - (s' + s'')$$

Note that $n - (s' + s'')$ is the sum of all the sizes of the remaining coalitions in CS. That is:

$$n - (s' + s'') = \sum_{C \in \{CS \backslash (C' \cup C'')\}} |C|$$

Also note that the size of any of the smallest two coalitions in CS has to be smaller than, or equal to, the sum of the sizes of the remaining coalitions in CS. In other words, by setting C' and C'' to the smallest two coalitions in CS, we have $s'' \leq n - (s' + s'')$. □

Based on theorem 1, what IDP does is avoid the evaluation of all the edges that do not belong to E^*. For instance, given our previous example of four agents (shown in figure 2), IDP does not evaluate the splittings of a coalition of size 3 into two coalitions of sizes 1 and 2. Instead, it simply sets $f_2[C] = v[C] : |C| = 3$. This is because, given: $n = 4, s' = 1, s'' = 2$, we have: $s'' > n - (s' + s'')$.

To analyse the difference between DP and IDP in terms of the

[5] Otherwise, if there is no path leading to a particular node, and if that node happens to be the optimal one, then DP will not be able to find it.

[6] Recall that the grand coalition is the one in which every agent is a member.

number of evaluated splittings, we provide equations for computing the exact number of splittings that are evaluated by each of the algorithms. Generally speaking, the number of possible subsets that contain s out of n elements is: $C_s^n = \frac{n!}{(n-s)!\times s!}$. Based on this, the number of possible splittings of a coalition $C : |C| = s$ into two coalitions of sizes s', s'' is given as follows:

$$N^{s',s''} = \begin{cases} \sum_{s''=\lceil \frac{s}{2} \rceil}^{s-1} \frac{C_{s''}^s}{2} & if \;\; s' = s'' \\ \sum_{s''=\lceil \frac{s}{2} \rceil}^{s-1} C_{s''}^s & otherwise \end{cases}$$

This is because, for every possible subset $C' \subset C$ of size s', there exists another subset $C'' \subset C$ of size s' such that $C' \cup C'' = C$.[7] Against this background, the total number of splittings (denoted t) is given as follows:

$$t = \sum_{s=1}^{n} (C_s^n \times \sum_{s'' \in \{\lceil \frac{s}{2} \rceil, \ldots, s-1\}} N^{n-s'',s''})$$

and the number of splittings that are avoided by IDP (denoted d) is computed as follows:

$$d = \sum_{s=1}^{n} (C_s^n \times \sum_{s'' \in \{\lceil \frac{s}{2} \rceil, \ldots, s-1\}, s'' > n-s} N^{n-s'',s''})$$

Based on this, the number of splittings that are evaluated by DP is t, and the number of those evaluated by IDP is $t - d$. Given 25 agents, figure 4 shows the number of splittings evaluated by both algorithms for coalitions of different sizes.[8] When computing the total number of operations, we found that IDP evaluates only 38.7% of the splittings that are evaluated by DP.

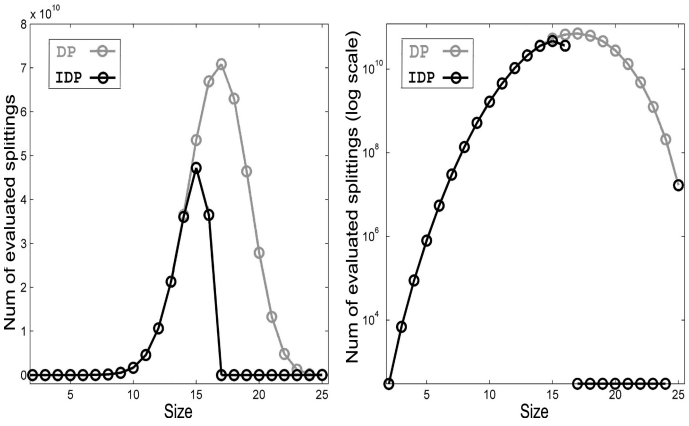

Figure 4: Given 25 agents, the figure shows (on linear and log scales) the number of splittings that are evaluated for the coalitions of size $s \in \{2, 3, \ldots, 25\}$.

Having shown how the number of evaluated splittings can be reduced without losing optimality, we will now show how the memory requirements can also be reduced. To this end, recall that the movement through a path, from a node CS to another, is done by splitting a coalition $C \in CS$ in two. The way this is done is based on $f_1[C]$, which returns a splitting of C that has an evaluation of $f_2[C]$. Now suppose that we did not use f_1. In this case, we would not have *instance access* to a splitting of C that has an evaluation of $f_2[C]$. Such a splitting, however, can still be found by re-evaluating all the splittings of C that were originally evaluated.[9] Note that the longest possible path contains n nodes, and that the movement from one node to another involves splitting one coalition. This means that we only need to re-evaluate the splittings of at most $n - 1$ coalitions. This is insignificant compared to the total number of evaluated splittings (which is exponential). Based on this, IDP avoids the use of f_1 and, therefore, require only 66.6% of the memory (compared to DP).

The memory requirements can be reduced even further whenever the input is not required after an optimal solution has been found. To this end, note that v is used two times. The first is when computing f_2 for every coalition, and the second is when determining whether it is beneficial to make a movement from a node CS in the graph.[10] Now suppose that, after computing $f_2[C]$, we stored in $v[C]$ the value that we would have normally stored in $f_2[C]$ (This can be seen in figure 2 by replacing every f_2 with v). In this case, determining whether it is beneficial to make a movement from a node CS can no longer be done by comparing $v[C]$ with $f_2[C]$ for all $C \in CS$ (this is because we no longer have the original $v[C]$). This, however, can still be done in a different way as follows. First, we re-evaluate the splittings of the coalitions in CS, but this time, without taking $v[C]$ into consideration. For example, $\{1, 2, 3\}$ is re-evaluated by only comparing $f_2[\{1\}] + f_2[\{2, 3\}]$ with $f_2[\{2\}] + f_2[\{1, 3\}]$ and $f_2[\{3\}] + f_2[\{1, 2\}]$. By not taking $v[C]$ into consideration, we end up with a different value $\hat{f}_2[C]$, which is, then, compared with $f_2[C]$ (which we now store in $v[C]$). Now if $\hat{f}_2[C] < f_2[C]$, then it is not beneficial to split the coalition in two. The process of re-evaluating the splittings is carried out for every coalition $C \in CS$, and if it is not beneficial to split any of the coalitions in two, then it is not beneficial to make a movement from CS. Note that the longest possible path contains n nodes, and that every node $CS_i \in L_i$ in that path contains i coalitions. This means that the maximum number of coalitions in a path is $\sum_{i=1}^{n} i$. This is, again, insignificant compared to the total number of evaluations. Based on this, whenever the input is not required after the optimal has been found, IDP only uses a table with 2^n entries, which is 33.3% of the memory requirements of DP.

4. CONCLUSIONS

This paper has developed a new state-of-the-art dynamic programming algorithm (called IDP) for solving the CSG problem. Specifically, IDP performs less operations, and requires less memory, and is guaranteed to find the optimal solution.

5. REFERENCES

[1] V. D. Dang and N. R. Jennings. Generating coalition structures with finite bound from the optimal guarantees. In *AAMAS-04*, pages 564–571, 2004.

[2] T. Rahwan, S. D. Ramchurn, A. Giovannucci, V. D. Dang, and N. R. Jennings. Anytime optimal coalition structure generation. In *AAAI-07*, pages 1184–1190, 2007.

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

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

[5] T. W. Sandholm and V. R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

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

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

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

[7] Note that, if $s' = s''$, then we only need to count half of the possible subsets of size s''.

[8] Larger numbers of agents show the same broad trends.

[9] This is especially true since DP has no preference on which splitting is chosen as long as it has an evaluation equal to $f_2[C]$.

[10] Recall that a movement from a node CS is only beneficial if there exists a coalition $C \in CS$ such that $v[C] < f_2[C]$.