A Fast Reconfiguration Algorithm for Modular Robots under Uncertainty using Block Partitioning

Ayan Dutta¹, Prithviraj Dasgupta¹, José Baca¹, Carl Nelson²

¹Computer Science Department, University of Nebraska at Omaha, NE 68182, USA
²Mechanical Engineering Department, University of Nebraska-Lincoln
Email: {adutta, pdasgupta, jbacagarica}@unomaha.edu, cnelson5@unl.edu

Abstract. We consider the problem of reconfiguration planning in modular self-reconfigurable robots (MSRs) in the presence of uncertainty due to operational limitations of the robot and dynamic nature of the environment. We employ a coalition game theory-based technique to address this problem by representing a configuration of the modules as a coalition structure and finding the ‘best’ configuration by searching for the optimal coalition structure. The key contribution of this paper is to speed up the search by making the observation that the maximum size a group or coalition of modules can have is limited to a maximum value, \( n_{\text{max}} \), which is determined by the operational and mobility constraints of the modules. We then propose an anytime algorithm called BlockPartitioning that uses integer partitioning along with an intelligent pruning technique to reject coalitions that are infeasible either due to their excessive size or due to excessive cost of coming together under current operational conditions. We have provided analytical results for our algorithm and also simulated it with different number of modules and different values of the maximum coalition size on an MSR, and, shown that our algorithm takes nominal time to find the optimal solution (less than 1 second for up to 12 modules).

Introduction

We consider the problem of reconfiguration planning for modular self reconfigurable robots (MSRs) - given a set of modules in a certain configuration how to reconfigure those modules to achieve a desired configuration while reducing the time and cost expended to achieve the new configuration. This problem becomes non-trivial due to two reasons. First, in dynamic environments the target configuration might not be known \textit{a priori}, and, consequently, the set of all possible configurations has to be explored on-the-fly to find the best possible configuration under the current condition. Secondly, the operation of the robot modules are characterized by uncertainty depending on the conditions in the environment and the capabilities of the modules related to maneuvering and docking/undocking with each other, which requires integrating an uncertainty model into the reconfiguration process that increases the complexity of the problem. Previous researchers [4, 14] have attempted to address this problem by using coalition game theory-based techniques to solve the MSR reconfiguration planning problem by searching for the optimal coalition structure in a coalition structure graph (CSG). In this paper, we propose an algorithm to improve the computational time of the MSR
reconfiguration problem based on the insight that it is infeasible to maneuver a set of connected modules beyond a certain maximum size, denoted by $n_{max}$. We describe an algorithm called BlockPartitioning (BP) that restricts the search space in the CSG to coalition structures that contain coalitions of size up to $n_{max}$. To further reduce the search space, the BP algorithm employs an intelligent pruning technique based on the expected utility of the current coalition structure. We provide analytical results on the completeness, convergence, anytime nature and complexity of our algorithm. We have also simulated the BP algorithm with different number of modules and different values of the maximum coalition size on an MSR within the Webots simulator and shown that it scales well with the number of modules, and, achieves significantly better speed-up than a CSG-based search algorithm for a similar problem.

![Fig. 1. MSR used for our experiments: single module (left), two module doing inchworm motion (right)](image)

**Related Work**

MSRs are a type of self-reconfigurable robots that are composed of several small modules; an excellent overview of the state of the art MSRs is given in [13]. Out of the three types of MSRs — chain, lattice and hybrid - we have used a chain-type MSR to illustrate the experiments in this paper although our techniques could be used for other MSR-types. The self-reconfiguration problem in MSRs has been solved using search-based [1, 2] and control-based techniques [10]. However, both these techniques require the goal configuration to be determined before the reconfiguration process starts. In contrast, our work in this paper is targeted towards task-based reconfiguration techniques [5] where the goal configuration is not determined *a priori*, but is determined as the configuration that helps the MSR continue performing its task efficiently. We do not explicitly specify a goal configuration but allow the reconfiguration algorithm to select a new configuration that reduces the reconfiguration cost.

Coalition game theory gives a set of techniques that can be used by a group of agents to form teams or coalitions with each other [7]. In terms of MSRs a coalition represents a set of MSR-modules that are connected together. Within coalition games, the coalition structure generation problem deals with partitioning the agents into a partition that gives the highest value. This problem is NP-complete, and researchers have proposed a coalition structure graph (CSG) based anytime algorithm to find a near-optimal solution. 
and an integer partitioning approach to find the optimal coalition structure [8]. Recently, researchers [4] applied the CSG problem to MSR reconfiguration and solved it using a graph partitioning technique. In contrast to this paper, all the above mentioned works do not consider the uncertainty involved in forming a coalition structure using MSR modules, which is a very practical consideration in a physical environment. In [14], the authors considered the MSR reconfiguration problem under uncertainty, and proposed a branch and bound-based algorithm that searches for the expected best coalition structure within a CSG. Our work also considers an uncertainty model based on physical parameters, but instead of searching in a CSG, we use an integer partitioning approach to generate and check coalitions up to a certain size $n_{\text{max}}$ and find the optimal configuration for the modules more quickly.

**MSR Reconfiguration as Coalition Formation**

We have used the MSR shown in Figure 1 for testing the techniques in this paper. For the simulated version of each module, we have used a GPS node that gives global coordinates on each robot\(^1\), an accelerometer to determine the alignment of the robot with the ground; each module is capable of wireless communication with other modules within a certain range. The movement of the MSR in fixed configuration is enabled through gait tables [12]. While moving in a fixed configuration, if the MSR’s motion gets impeded by an obstacle or an occlusion in its path, it needs to reconfigure into a new configuration so that it can continue its movement efficiently. The MSR reconfiguration problem is formalized below.

Let $A$ be the set of modules or agents that have been deployed in the environment. Let $\Pi(A)$ be the set of all partitions of $A$ and let $CS(A) = \{A_1, A_2, ..., A_k\} \in \Pi(A)$ denote a specific partition of $A$. We call $CS(A)$ a configuration or a coalition structure, and $A_i$ as a coalition. In terms of the MSR, $A_i$ is the $i$-th MSR in the configuration $CS(A)$. $A_i = \{a_{i1}, a_{i2}, a_{i3}, ..., a_{i|A_i|}\}$ where $a_{i1}$ and $a_{i|A_i|}$ are the leading and trailing modules of $A_i$ respectively and $\{a_{ij}, a_{ij+1}\}, j = 1, ..., |A_i| - 1$ is the set of physically coupled modules in $A_i$. When $|A_i| = 1$ the MSR is a single module. Let $Val: A \rightarrow \mathbb{R}$ denote the value function given below:

$$Val(A_i) = \begin{cases} |A_i|^2, & \text{if } |A_i| \leq n_{\text{max}} \\ 0, & \text{otherwise} \end{cases}$$

(1)

The value function assigns to each coalition $A_i$ a real number corresponding to a virtual reward that the coalition can obtain for performing its assigned task. We assume that larger coalitions are able to perform their task better and obtain higher rewards, up to a size of $n_{\text{max}}$. $n_{\text{max}}$ then denotes the maximum allowable size of a coalition; it is determined from the physical characteristics of the modules (e.g., battery, maximum weight of other modules that can be lifted by the modules), the features in the environment such as the amount of free space, the task that the robot has to perform, past performance of coalition sizes that have been used to perform similar tasks, etc.

\(^1\)In the physical MSR, relative positioning is planned to be calculated by combining IMU and IR sensors; communication is implemented using XBee modules.
The value of configuration $CS(A)$ is given by the summation of the values of coalitions comprising it, i.e., $Val(CS) = \sum_{A_i \in CS(A)} Val(A_i)$.

**Uncertainty Model.** The movement of an MSR during reconfiguration is characterized by uncertainty originating from various factors such as inaccurate sensor readings, friction with the substrate while moving, and limited perception range of sensors. Because the principal operation in MSR reconfiguration is the docking and aligning between modules, we have used the probability of a pair of MSRs to dock successfully with each other to represent uncertainty. Following [14], we consider three sources of uncertainty for the MSR: (i) **Distance uncertainty** is the uncertainty arising out of the distance required to be traversed by a pair of MSRs before docking with each other. It is modeled as a half-Gaussian distribution $\mathcal{N}(\mu_{du}, \sigma_{du})$, with the maximum value of the distance set to the communication range $r_{com}$ of a module - the farthest distance at which two modules are able to discover each other (Figure 2(a)). (ii) **Alignment uncertainty** is the uncertainty arising from the angle that the MSR modules need to rotate before they can align with each other before docking. It is modeled as a Gaussian distribution $\mathcal{N}(\mu_{au}, \sigma_{au})$ (Figure 2(b)) based on the premise that it is easier for a pair of modules to dock with each other when they are aligned (angular difference near 0° or 180°). (iii) **Environment uncertainty** is the uncertainty contributed by the operational conditions in the environment due to factors such as obstacles, terrain conditions, surface friction, etc. The operational conditions are characterized as a noise variable in $[0, 1]$. The uncertainty is modeled as a half-Gaussian distribution $\mathcal{N}(\mu_{eu}, \sigma_{eu})$ over the environment noise variable (Figure 2(c)). We associate with each Gaussian a weight that denotes the Gaussian’s effect on the total motion uncertainty of the robot denoted by $w_{du}$, $w_{au}$, and $w_{eu}$ respectively; each weight is given by the inverse of the corresponding Gaussian’s variance. The weighted mean of the three Gaussians then gives the total motion uncertainty [6], expressed as a probability, when two MSRs $A_i$ and $A_j$ attempt to connect with each other, as given below:

$$prob(A_i, A_j) = \frac{w_{du} \cdot p_{du} + w_{au} \cdot p_{au} + w_{eu} \cdot p_{eu}}{w_{du} + w_{au} + w_{eu}},$$

where $p_{du} \in \mathcal{N}(\mu_{du}, \sigma_{du})$, $p_{au} \in \mathcal{N}(\mu_{au}, \sigma_{au})$, and $p_{eu} \in \mathcal{N}(\mu_{eu}, \sigma_{eu})$, and, $w_{du} = \frac{1}{\sigma_{du}}$, $w_{au} = \frac{1}{\sigma_{au}}$, $w_{eu} = \frac{1}{\sigma_{eu}}$. Figure 2. Probability of a pair of MSRs to dock successfully with each other for (a) different distances between the MSRs, (b) different angular orientation between the MSRs, and, (c) different environment noise values.
Expected Cost functions. Let $A_i$ and $A_i'$ denote two distinct partitions of the same set of modules $\{a_i\}$. Reconfiguring the modules from $A_i$ to $A_i'$ involves undocking the appropriate modules in $A_i$, moving the relevant modules to the vicinity of each other, and aligning and docking the appropriate set of modules to get to $A_i'$. We denote the combined cost of these operations as $\text{cost}(A_i, A_i')$. Including motion uncertainty described above, we denote the expected cost in reconfiguring from coalition $A_i$ to $A_i'$ as: $\text{cost}(A_i, A_i') = \text{cost}(A_i, A_i') \times (1 - \text{prob}(A_i, A_i'))$. Going further, we denote the expected reconfiguration cost from configuration $A_{old}$ to configuration $A_{new}$ as $\text{cost}(A_{old}, A_{new}) = \sum_{A_i \in CS(A_{old}), A_i' \in CS(A_{new})} \text{cost}(A_i, A_i')$. Based on the above formulation, we can now formally define the MSR reconfiguration problem as the following:

**Definition 1** MSR Reconfiguration Problem. Given a set of modules $A$ and an arbitrary configuration $CS_{old}(A) = \{A_1^{old}, A_2^{old}, ..., A_k^{old}\}$ in which they are deployed, find a new configuration $CS_{new}(A) = \{A_1^{new}, A_2^{new}, ..., A_k^{new}\}$ such that the following constraint is satisfied:

$$\max_{CS_{new}(A) \in \Pi(A)} (Val(CS_{new}(A)) - \text{cost}(A_{old}, A_{new}))$$

For notational convenience, we denote expected utility of a configuration $CS(A_{new})$ as $\mathcal{U}(CS)$ where $\mathcal{U}(CS) = Val(CS(A_{new})) - \text{cost}(A_{old}, A_{new})$, while assuming that $A_{new}$ and $A_{old}$ are understood from context. Then, the optimal coalition structure or best configuration is given by $CS^* = \arg \max_{CS \in \Pi(A)} \mathcal{U}(CS)$. Also, in the rest of the paper, for the sake of legibility, we will slightly abuse notations by referring to expected utility and expected cost as utility and cost respectively.

**Block Partitioning(BP) Algorithm**

From the definition of the value function in Equation 1, the maximum value of a configuration happens when each coalition has a size $n_{max}$. Based on this observation, the main idea behind our proposed approach to solve the MSR reconfiguration problem is to start inspecting the partitions of $A$ where the most number of coalitions have size $n_{max}$. To proceed with generating partitions in a systematic manner, we use integer partitions called blocks that are defined below.

Let $IP(n)$ denote a function that returns an ordered set of integer partitions of a positive integer $n$, starting with $(n, 0)$ and ending with $(1, 1, 1, ... 1)$. Let $p(n) = |IP(n)|$. Given a set of modules $A$ and an integer $n_{max} \leq |A|$, let $IP_{n_{max}} \in IP(|A|)$ denote an integer partition of $A$ into $\frac{|A|}{n_{max}}$ modules of size $n_{max}$ and at most one module of size $|A| - \lfloor \frac{|A|}{n_{max}} \rfloor$. Let $p_{n_{max}} = |IP_{n_{max}}|$.  

**Definition 2** Block and Coalition Set. Given a set of modules $A$, maximum coalition size $n_{max}$, and $IP_{n_{max}}$ as defined above, a block is defined as $B_i = \{IP(ip_i) : ip_i \in IP_{n_{max}}\}$. Let $B_{i,j}$ denote the $j$-th element (an integer partition) in block $B_i$, and $B_{i,j,k}$ denote the $k$-th part of $B_{i,j}$. A coalition set is defined as $\beta_{i,j} = \{A_k : A_k \subseteq A, A_k \cap B_{i,j,k} \neq \emptyset, k = 1, 2, ..., |B_{i,j}|\}$.
As an example, with $|A|=10$ and $n_{\text{max}}=4$, $IP_{n_{\text{max}}}=(4,4,2)$. Then, $ip_1=ip_2=4$ and $ip_3=2$. And, $B_1 = B_2 = \{(4,0),(3,1),(2,2),(2,1,1),(1,1,1,1)\}$, $B_3 = \{(2,0)(1,1)\}$. Finally, a coalition set, e.g., $\beta_{1,2}$ corresponds to integer partition $B_{1,2} = (3,1)$ in block $B_1$, and consists of sets of coalitions whose size are 3 and 1, e.g., \{\{a_8,a_9,a_{10}\}\{a_1\}\}, \{\{a_9,a_{10},a_1\}\{a_8\}\}, and so on. Note that for integer partitions, the order of the parts does not change the coalitions, i.e., $(1,3) = (3,1)$ give the same coalition sets.

Algorithm 1: searchBlock

\textbf{searchBlock}(Integer Partition $B_{i,j}$, Coalition Structure $CS$)

\textbf{Input}: $B_{i,j}$: current integer partition of $B_{i}$ being inspected, $CS$: currently generated coalition structure

\textbf{Output}: 'Best' coalition structure $CS$.

\textbf{while} (not all elements of $B_{i,j}$ inspected) \textbf{do}

//first check fitness of $B_{i,j}$ before generating coals.

\textbf{if} all coalitions of sizes $< n_{\text{max}}$ have been seen before \textbf{then}

//go to next int. partition $B_{i,j+1}$

\textbf{continue;}

\textbf{while} (not all coalitions corr. to $B_{i,j}$ generated) \textbf{do}

$\beta_{i,j,k}$ $\leftarrow$ generate next element of coal. set $\beta_{i,j}$

//Next check for bad coalition.

\textbf{if} any coalition in $\beta_{i,j,k}$ is a bad coalition \textbf{then}

//skip to next coalition set in $\beta_{i,j}$

\textbf{continue;}

\textbf{else}

// passed both tests

$CS_{\text{curr}} \leftarrow CS_{\text{curr}} \cup \beta_{i,j,k}$;

\textbf{if} util. of any coal. size $|\beta_{i,j}|$: $\beta_{i,j,k}$ exceeds highest recorded util. for that size \textbf{then}

\textbf{update} $U_{hi}(|\beta_{i,j}|)$;

\textbf{if} current block is last block $B_{p_{\text{max}}}$ \textbf{then}

Calculate $\overline{U}(CS_{\text{curr}})$ and $\alpha_{CS_{\text{curr}}}$;

\textbf{if} first coalition OR ($\alpha_{CS_{\text{curr}}} > \alpha_{\text{best}}$) \textbf{then}

\textbf{update} $\alpha_{\text{best}} \leftarrow \alpha_{CS_{\text{curr}}}$ and $\text{bestCS} \leftarrow CS_{\text{curr}}$;

\textbf{else}

\textbf{searchBlock}($B_{i+1,j}, CS_{\text{curr}}$); \textbf{return; } //go to next coalition in $B_{i-1,j}$

Because our value function (Eqn. 1) assigns the highest value to coalitions of size $n_{\text{max}}$, when $A$ is partitioned into coalitions of sizes $n_{\text{max}}$ given by $IP_{n_{\text{max}}}$, $Val(CS_{n_{\text{max}}}) = \sum_{A_i \in CS(A)} Val(A_i)$ has the highest value. We denote this ideal utility (when there is no cost to configure the modules) as $U_{\text{ideal}} = Val(CS_{n_{\text{max}}})$. For any coalition structure, $CS$, let $\alpha_{CS} = \frac{\overline{U}(CS)}{U_{\text{ideal}}}$. $\alpha_{CS}$ is dependent on the cost of forming coalition structure.
CS and gives a worst case bound. Let \( CS_1 \) denote the first coalition structure generated by the BP algorithm (with coalition sizes given by \( IP_{n_{\text{max}}} \)). \( \alpha_{CS_1} = \frac{U(\text{CS}_1)}{U_{\text{ideal}}} = 1 - \frac{\text{cost}(\text{CS}_1)}{\text{Val}(\text{CS}_1)} \). Note that \( \alpha_{CS_1} \) denotes the initial worst case bound on \( \alpha_{CS} \).

The core of the BP algorithm is the searchBlock method that selectively generates a coalition structure, starting from \( CS_1 \) and admits a new coalition structure \( CS \) only if it improves the worst case bound \( \alpha_{CS} \). In the searchBlock method’s pseudo-code shown in Algorithm 1, we sequentially inspect all the integer partitions in every block \( B_i, i = 1..p_{n_{\text{max}}} \). For every integer partition \( B_{i,j} \in B_i \), we inspect the coalition sets \( \beta_{i,j} \). To prevent generating unpromising coalition sets in \( \beta_{i,j} \), we employ two pruning techniques. First, we check if the coalition sizes or parts of the integer partition \( B_{i,j} \) pass the fitness test given below:

**Definition 3** Fitness function: Given an element (integer partition) \( B_{i,j} \) of block \( B_i \) the fitness function, \( \Phi : \{Z^+\} \rightarrow \{\text{Pass, Fail}\} \), takes as argument the integer partition given in \( B_{i,j} \) and returns a Pass/Fail value that determines whether the coalition set corresponding to \( B_{i,j} \) should be generated.

The fitness function is implemented as:

\[
\Phi(B_{i,j}) = \begin{cases} 
\text{Pass}, & \text{if } \left( \sum_{k \in B_{i,j}} \hat{U}_{hi}(k) + \sum_{l=1,(i-1),(i+1) \ldots p_{n_{\text{max}}}} \sum_{m=1 \ldots |B_l|} \sum_{k \in B_{l,m}} \hat{U}_{hi}(k) \right) > U^*(CS) \\
\text{Fail}, & \text{otherwise,}
\end{cases}
\]  

(3)

where, \( \hat{U}_{hi}(k) \) denotes the maximum utility of a coalition of size \( k \) seen by the algorithm.\(^2\) The fitness test inspects the sum of the maximum utilities \( \hat{U}_{hi}(k) \) of the coalition sizes given by \( B_{i,j} \) together with all possible coalition sizes from every other block, and prevents generating coalition sets if that value does not exceed the highest expected utility of a coalition structure found thus far. Although the fitness function inspects all blocks, it works with coalition sizes instead of coalition sets and significantly reduces the running time of the algorithm by pruning all coalition sets for integer partitions that fail the fitness test. An example of pruning using the fitness function is shown at the bottom left corner of Figure 3. If the fitness test is not passed we continue with the next element (integer partition) \( B_{i,j+1} \) of current block \( B_i \). If the fitness test is passed, we generate the coalitions \( \beta_{i,j,k} \) for the current coalition set \( \beta_{i,j} \).

For our second pruning technique, we check the coalitions in the current coalition set and remove bad coalitions - coalitions which involve a relatively high cost of getting the modules together, as given by the following definition:

**Definition 4** Bad Coalition: For any coalition \( C \) with two modules only, if \( \text{cost}(C) \geq \text{thres}^* \times \text{Val}(C) \), where \( \text{thres}^* \in [0, 1] \), then \( C \) is considered to be a bad coalition.

\(^2\)The BP algorithm has to generate coalitions of all coalition sizes \( \leq n_{\text{max}} \) at least once so that \( \hat{U}_{hi}(k) \) can be determined for every \( k < n_{\text{max}} \) and the fitness test can be used.
If any bad coalition is detected we discard the current coalitions and continue to the next element in the coalition set $\beta_{i;j}$. For example, as shown in Figure 3, \{$a_8, a_9$\} is determined to be a bad coalition, and, subsequently all coalition structures involving these two agents in the same coalition are pruned.

If both the bad coalition and fitness conditions are satisfied, we check if the block being processed is the last block $B_{n_{\text{max}}}$, If the current block is not the last block, we make a recursive call to searchBlock to process the next block $B_{i+1;j}$. On the other hand, if the current block is the last block, the partitioning of $A$ is complete and we have a coalition structure. We then calculate the expected utility of this coalition structure and update the worst case bound ($\alpha_{\text{best}}$) and best coalition structure ($\text{bestCS}$) variables, if required, and, continue to the next coalition set of the same (last) block. If all coalition sets of the last block have been inspected, we continue to the next coalition set for the previous block(s) $B_{i-1;j}$ (handled by recursion).

If $|A| = 10$, $n_{\text{max}} = 4$, $IP_{n_{\text{max}}} = (4,4,2)$, $P_{n_{\text{max}}} = |IP_{n_{\text{max}}}| = 3$ 
$B_1 = B_2 = ((4,0) (3,1) (2,2) (1,1,1,1)); B_3 = ((2,0) (1,1,1))$

**Fig. 3.** Working procedure of the BP algorithm; in the left figure the blue colored partition is the first coalition structure to be checked and the red bordered partitions are pruned by the fitness function; the right figure shows how the search process continues and the red bordered coalitions are pruned by prune method.
Analytical Evaluation of BP Algorithm

Lemma 1. The fitness function $\Phi()$ does not remove an optimal coalition structure.

Proof: (by contradiction) Suppose that $\Phi(B_{i,j})$ returns $Fail$ for and removes the optimal coalition structure $CS^*(A) = \{A_1^*, A_2^*, ..., A_{\#}^*\}$. From the definition of $\Phi(B_{i,j})$ in Eqn. 3, it follows that $\sum_{A_i^* \subseteq A} \bar{U}_i(|A_i^*|) < \bar{U}(CS^*(A))$. Again, from the definition of optimal coalition structure, $\bar{U}(CS^*(A)) = \sum_{A_i^* \subseteq A} \bar{U}(A_i^*)$ s.t. $\bar{U}(A_i^*) = \bar{U}_i(|A_i^*|)$ for at least one $i$, and, $\bar{U}(A_i^*) \leq \bar{U}_i(|A_i^*|)$, for all other $i$. This means that $\bar{U}(CS^*(A)) \leq \sum_{A_i^* \subseteq A} \bar{U}_i(|A_i^*|)$ for all $i$. Or, $\sum_{A_i^* \subseteq A} \bar{U}_i(|A_i^*|) > \bar{U}(CS^*(A))$, which contradicts the inequality derived above from the definition of $\Phi(B_{i,j})$ in Eqn. 3. Hence proved.

Lemma 2. A bad coalition can not be a part of the optimal coalition structure if $\text{thresh}^* > 0.5$.

Proof: From its definition, a bad coalition involves two modules. For a two module coalition $C = \{a_{11}, a_{12}\}$, $\bar{U}(C) = Val(C) - \cos(C) \leq (|C|)^2 - \text{thresh}^* \cdot (|C|)^2 = 2^2 - \text{thresh}^* \cdot 2^2$. If the modules were singletons, then $\bar{U}(|a_{11}\}) + \bar{U}(|a_{12}\}) = Val(|a_{11}\}) + Val(|a_{12}\}) = 2$. If $C$ is not part of the optimal coalition structure, it is a sub-optimal coalition, i.e., $\bar{U}(C) < \bar{U}(|a_{11}\}) + \bar{U}(|a_{12}\})$, or, $4 - \text{thresh}^* - 4 < 2$, giving $\text{thresh}^* > 0.5$. In general, if a bad coalition $C = \{a_{11}, a_{12}\}$ is part of a larger coalition $C'$, then $\bar{U}(C') < \bar{U}(C' \setminus \{a_{12}\}) + \bar{U}(|a_{12}\})$. This means that $C'$ cannot be part of the optimal coalition structure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of agents ($</td>
<td>A</td>
</tr>
<tr>
<td>Max. desired coalition size ($n_{max}$)</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>Mean and std. dev. for distance uncertainty</td>
<td>0, 17/3 (in cm ± 17 cm)</td>
</tr>
<tr>
<td>Mean and std. dev. for angle uncertainty</td>
<td>±2, ±6</td>
</tr>
<tr>
<td>Mean and std. dev. for env. uncertainty</td>
<td>0, 0.33</td>
</tr>
<tr>
<td>Bad coalition threshold probability (thresh*)</td>
<td>0.6</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>A</th>
<th>Ratio to opt.</th>
<th>Runtime (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>0.078</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.078</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.109</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.148</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.150</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0.302</td>
</tr>
</tbody>
</table>

(b)

Fig. 4. (a)Parameters used for experiments (b) Ratio of utilities of best coalition structure found by BP algorithm to optimal utility and running times.
Lemma 3. **BP Algorithm is anytime.**

*Proof*: The BP algorithm generates and inspects coalitions in order of decreasing sizes, starting with the integer partition of $A$ given by $IP_{n_{\text{max}}}$. Therefore, the first coalition structure generated will have the highest number, $\lfloor \frac{|A|}{n_{\text{max}}} \rfloor$, of coalitions of size $n_{\text{max}}$ in any coalition structure with $|A|$ modules. The initial value of $\alpha_{\text{best}}$ is established for this coalition structure. Since the searchBlock method only updates $\alpha_{\text{best}}$ when a coalition structure giving a higher value of $\alpha$ than the current $\alpha_{\text{best}}$ is found, this bound will successively improve. This guarantees that after the first generated coalition structure, the utility of any coalition structure that is admitted by the algorithm will be within a certain bound from the optimal coalition structure, (if not the optimal itself). Therefore, we can establish a worst case bound from the very first coalition structure inspected by the algorithm. This shows that the BP algorithm is *anytime*.

Lemma 4. **BP Algorithm finds the optimal coalition structure.**

*Proof*: Due to its anytime property, the BP algorithm only admits coalition structures that have a higher utility than the first inspected coalition structure. The BP algorithm also prunes unpromising coalitions that cannot be part of an optimal coalition structure (by Lemmas 1 and 2). This ensures that the BP algorithm never accepts a coalition structure that has a lower utility than a previously seen coalition structure. Hence, it finds the optimal coalition structure eventually.

The completeness of the BP algorithm also follows from Lemma 4 which guarantees that it always finds the optimal coalition structure.

**Complexity.** The BP algorithm has to generate all coalitions of every size up to $n_{\text{max}}$ at least once before it can use the fitness function to prune unpromising coalitions.

The number of such coalitions is given by

$$
\prod_{k=0}^{\lfloor \frac{|A|}{n_{\text{max}}} \rfloor} \frac{(|A| - k \cdot n_{\text{max}})!}{(n_{\text{max}})!} \times \sum_{k=0}^{n_{\text{max}}} S(n_{\text{max}}, k).
$$

The lower bound on the number of coalitions generated by the BP algorithm is therefore given by $\Omega(n_{\text{max}}^{n_{\text{max}}})$. Note that this bound is independent of the total number of modules $|A|$. It is a significant improvement as the complete search space has $O(|A|^{|A|})$ coalition structures, and without the fitness test, the bound would become $O(n_{\text{max}}^{|A|})$, which is significantly worse than the current bound for $|A| \gg n_{\text{max}}$.

The upper bound on the number of steps of the fitness function is given by $(p(n_{\text{max}}))^{\lfloor \frac{|A|}{n_{\text{max}}} \rfloor},$ where $p(n)$ denotes the partition function of $n$ [15].

**Experimental Evaluation of the BP algorithm**

To verify the performance of our proposed reconfiguration planning technique, we implemented the BP algorithm on a desktop PC (Intel Core i5 -960 3.20GHz, 6GB DDR3 SDRAM). We consider a setting where a current configuration of the MSR consists of $|A| = 6$ through 12 modules and needs to reconfigure. They are placed within a $4 \times 4$ grid.

---

3 $S(n, k)$ denotes the Stirling number of the second kind

4 $n_{\text{max}}$ is determined from physical capability of modules and does not grow with $|A|$. 

m² environment. Simulations are performed within the Webots robot simulator. The objective of the modules is to reconfigure to a new configuration that gives the highest utility using the BP algorithm. The values of different parameters used by our algorithm are shown in Figure 4(a).

In the first set of experiments, we analyzed the effect of the main concept of our algorithm i.e. finding the best coalition structure possible with intelligent pruning. For $6 \leq |A| \leq 12$, we were able to do an exhaustive search in the space of all coalition structures to find the optimal coalition structure and see that our algorithm is able to find the optimal coalition structure for all values of $|A|$. We kept $n_{max}$ fixed at 4. The time taken to find the optimal value with our intelligent pruning technique is given in Figure 4(b). We see that the time taken by the BP algorithm is of the order of fractions of a second.

We have also tested the number of coalitions generated using BP algorithm. $n_{max}$ was kept fixed at 4 for this case. The results are shown in Figure 5(a). The results show that, to find the optimal coalition structure, our proposed algorithm generates only a fraction of the total number of coalitions possible. For $|A| = 6$, the number of coalitions generated is 21; for $|A| = 12$, it is 1436. In the next set of experiments, we have kept the number of modules fixed at $|A| = 12$ and varied $n_{max}$ through 4, 5 and 6. The number of coalitions generated in each case is shown in Figure 5(b). As can be seen from the figure, as the value of $n_{max}$ increases, the number of integer partitions also increases. So, the number of coalitions generated is more when $n_{max}$ is 6 than when it is 4. But still, using our two types of pruning strategies, we were able to prevent that number from increasing rapidly.

Finally, we have compared the performance our algorithm against a CSG search technique for MSR reconfiguration that was described in [14], where they have proposed the searchUCSG algorithm to find the optimal configuration under uncertainty using a coalition structure graph. We have compared the time taken by these two algorithms to find the optimal coalition structure. The results shown in Figure 5(c). We see that as the number of modules increases, e.g., for $|A| = 12$ modules, the searchUCSG algorithm takes 381 seconds, while the BP algorithm takes 0.302 seconds - an improvement of the order of $10^5$ times. Overall, the BP algorithm finds the optimal coalition

![Fig. 5.](image-url)
structure in considerably lower time by avoiding searching inspecting integer partitions and pruning coalition structures intelligently.

Figure 6 shows an execution of the BP algorithm within Webots to find the optimal coalition structure. Initially, the MSR configuration was \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8\}\}. \(n_{\text{max}}\) was set to 4 (Figure 6.1). The optimal configuration was found to be \{\{1, 2, 3, 8\}, \{4, 5, 6, 7\}\}. Figures 6.2 and 6.3 show the intermediate and final configurations.

**Conclusion and Future Work**

We have proposed a MSR reconfiguration planning technique that models the problem as a block partitioning problem and intelligently searches through these blocks. We are investigating a structured way to learn the value function based on modules’ past performances and task types. We are also investigating reconfiguration algorithms that can be execute quickly within the time and space constraints of physical modules. Finally, we are working on implementing this algorithm on physical modules.

**Acknowledgements**

The research in this paper has been supported through a NASA grant as part of the ModRED project.

**References**


