A Fast Coalition Structure Search Algorithm for Modular Robot Reconfiguration Planning under Uncertainty

Ayan Dutta[1], Prithviraj Dasgupta[1], Jose Baca[1], Carl Nelson[2]

Abstract We consider the problem of reconfiguration planning in modular robots. Current techniques for reconfiguration planning usually specify the destination configuration for a modular robot explicitly. We posit that in uncertain environments the desirable configuration for a modular robot is not known beforehand and has to be determined dynamically. In this paper, we consider this problem of how to identify a new 'best' configuration when a modular robot is unable to continue operating efficiently in its current configuration. We build on a technique that enumerates all the possible partitions of a set of modules requiring reconfiguring as a coalition structure graph (CSG) and finds the 'best' node in that graph. We propose a new data structure called an uncertain CSG (UCSG) that augments the CSG to handle uncertainty originating from the motion and performance of the robot. We then propose a new search algorithm called searchUCSG that intelligently prunes nodes from the UCSG using a modified branch and bound technique. Experimental results show that our algorithm is able to find a node that is within a worst bound of 80% of the optimal or best node in the UCSG while exploring only half the nodes in the UCSG. The time taken by our algorithm in terms of the number of nodes explored is also consistently lower than existing algorithms (that do not model uncertainty) for searching a CSG.

1 Introduction

Over the past few years, modular robots have been proposed as an attractive paradigm for building highly dexterous robots that are capable of maneuvering in environments that are difficult to move in. The major advantage offered by a modular robot is that it consists of individual modules which can be dynamically configured into a shape or configuration that enables the robot to perform tasks under

Ayan Dutta, Jose Baca, Prithviraj Dasgupta
Computer Science Department, University of Nebraska at Omaha, NE 68182, USA e-mail: adutta, jbaca, pdasgupta@unomaha.edu. Carl Nelson
Mechanical Engineering Department, University of Nebraska-Lincoln e-mail: cnelson5@unl.edu

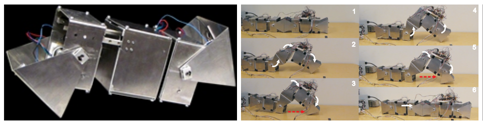
its current conditions. One of the principal computational challenges in designing modular robots is to solve the problem of *reconfiguration planning* - given a set of modules in a certain configuration how to reconfigure those modules to achieve a desired configuration while reducing the time and cost expended to achieve the new configuration. This problem becomes non-trivial if the target configuration is not known *a priori*, and, consequently, the set of all possible configurations has to be explored on-the-fly to find the best possible configuration. Because the space of possible configurations is exponential in the number of modules, conventional search algorithms are unsuitable to solve the reconfiguration planning problem within a reasonable amount of computation time and space. The problem becomes further complicated if we include uncertainty in the mobility and connections of modules, which are practical considerations for any physical robot. In this paper, we address this reconfiguration planning problem for modular robots under uncertainty, using a representation from coalition game theory called coalition structure graph (CSG). We augment the basic CSG to handle uncertainty in modules' movement and in the environment using parameters derived from physical characteristics of a modular robot called ModRED. We formulate the reconfiguration planning problem as an uninformed search problem on the UCSG and propose a modified branch-and-bound algorithm called *searchUCSG* to solve it. We have simulated our algorithm for reconfiguration planning of ModRED and shown that it explores much fewer nodes (about 50%) and its solution quality is within 80% of the optimal node in UCSG. And also the runtime of our algorithm is relatively less than existing algorithms(that do not include uncertainty) to find the optimal coalition structure.

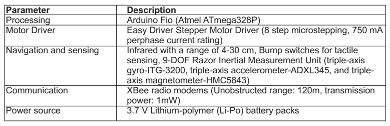
2 Related Work

Modular self-reconfigurable robots (MSRs) are a type of self-reconfigurable robots that are composed of several modules. These modules can change their connections with each other to manifest different shapes of the MSR and select a shape that enables the MSR to perform its assigned task efficiently [12]. An excellent overview of the state of the art MSRs and related techniques 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 types too. 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 initial and goal configuration to be determined before the reconfiguration process starts. A third technique called task-based reconfiguration has recently shown considerable success [5]. Here the goal configuration of an MSR doing reconfiguration is not determined *a priori*, but is determined as the configuration that helps the MSR perform its task efficiently. Our work in this paper is targeted towards task-based reconfiguration techniques; we do not explicitly specify a goal configuration but allow the reconfiguration algorithm to select a new configuration that reduces the reconfiguration cost i.e. cost for going from one configuration to another which includes operations like docking, undocking, aligning, crawling, etc.

and thus selects the best configuration. 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]. A coalition can be loosely defined as a set of agents that remain together with the intention of cooperating with each other, possibly to perform a task. In terms of MSRs a coalition represents a set of MSR-modules that are connected together. Within coalition games, the coalition structure generation problem that deals with partitioning the agents has received significant attention. This problem is NP-complete, and Sandholm [11] and Rahwan [8] have proposed anytime algorithms to find near-optimal solutions. In contrast to these works, we incorporate uncertainty into the CSG and propose a new algorithm with branch and bound -based pruning to find the best coalition structure.

3 ModRED MSR





Parameter	Description
Processing	Arduino Fio (Atmel ATmega328P)
Motor Driver	Easy Driver Stepper Motor Driver (8 step microstepping, 750 mA perphase current rating)
Navigation and sensing	Infrared with a range of 4-30 cm, Bump switches for tactile sensing, 9-DOF Razor Inertial Measurement Unit (triple-axis gyro-ITG-3200, triple-axis accelerometer-ADXL345, and triple-axis magnetometer-HMC5843)
Communication	XBee radio modems (Unobstructed range: 120m, transmission power: 1mW)
Power source	3.7 V Lithium-polymer (Li-Po) battery packs

Fig. 1 (a) Single module of the MSR. (b) Two modules doing inchworm motion (c) Major components of the MSR.

We have used an MSR called ModRED [3] that is currently being developed by us, for implementing and testing the techniques in this paper. Unlike most other MSRs, it has 4 DOF (3 rotational and 1 translational); this allows each module to rotate along its long axis as well as extend along that same axis. Single ModRED module is shown in Figure 1(a). This combination of DOF enables the MSR to achieve a greater variety of gaits to possibly maneuver itself out of tight spaces. 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, in addition to the Xbee modules in the physical robot. Two ModRED modules performing an inchworm motion and its major components are shown in Figure 1[2]. The movement of the MSR in fixed configuration is enabled through gait tables [12]. Videos showing the movement of the MSR in different configurations (e.g., chain, ring) using gait tables are available at http://cmantic.unomaha.edu/projects/modred/. 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. In the next section, we formalize the MSR self-reconfiguration problem

[1] In the physical MSR, relative positioning is planned to be calculated by combining IMU and IR sensors.

[2] The order of connectivity does not alter the operation of modules.

and then provide a coalition structure graph-based approach for finding the best configuration.

4 Dynamic Self-Reconfiguration in MSRs

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, and A_i as the i-th MSR in that configuration. $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}, ..., a_{i_{|A_i|}}\}$ where a_{i_1} and $a_{i_{|A_i|}}$ are the leading and trailing modules of A_i respectively and $\{a_{i_j}, a_{i_{j+1}}\}, j = 1...|A_i| - 1$ is the set of physically coupled modules in A_i. When $| A_i |= 1$ the MSR is a single module. We define $Val : \Pi(A) \rightarrow \mathbb{R}$, a value function that assigns each partition $CS(A) \in \Pi(A)$ a real number. $Val(CS(A))$ is a metric that gives a virtual reward or benefit obtained by the robots when they perform their assigned tasks while in the configuration $CS(A)$. Evidently, the most suitable configuration for a set of modules is the one that maximizes $Val(CS(A))$. The problem that we study in this paper is 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:*

$$\max_{CS_{new}(A) \in \Pi(A)} Val(CS_{new}(A))$$

Note that k and k' in the above definition may be different. Such reconfigurations can happen, for example, when a set of modules is deployed into the environment from an aircraft and the modules need to get into a configuration that maximizes their value. Another instance of reconfiguration could happen when an MSR gets stuck at an obstacle while navigating during an exploration task and needs to get into a new configuration so that it can continue performing its navigation. The objective of the MSR reconfiguration problem is to get the MSR into a new configuration that allows it to continue its task while giving the highest value over its current partitions.

In our previous work [9, 4], we have shown that a systematic way to go about analysing the configurations in $\Pi(A)$ is provided by a hierarchical graph structure called a coalition structure graph(CSG) [7]. In a CSG, each partition $CS(A) \in \Pi(A)$ is called a coalition structure and appears as a node in the CSG. The parts or subsets of a partition are called coalitions, denoted by S. A CSG with 4 agents is shown in Figure 2. CSG nodes are organized into levels. *Level l* indicates that every node in level l in CSG has exactly l subsets or coalitions as its members. CSGs offer a structured way of exploring coalition structures because a node at level $l - 1$ can be generated by combining a pair of coalitions from a node at level l. Let $succ : \Pi(A) \rightarrow \Pi(A)$ denote a successor function that takes a node at level l and generates a node at level $l - 1$. $succ^{(k)}(CS(A))$ denotes the node that is reached from $CS(A)$ by applying the $succ(\cdot)$ function k times. Each node or coalition structure $CS(A)$ is associated with a value $Val(CS(A))$ that corresponds to the value of the partition

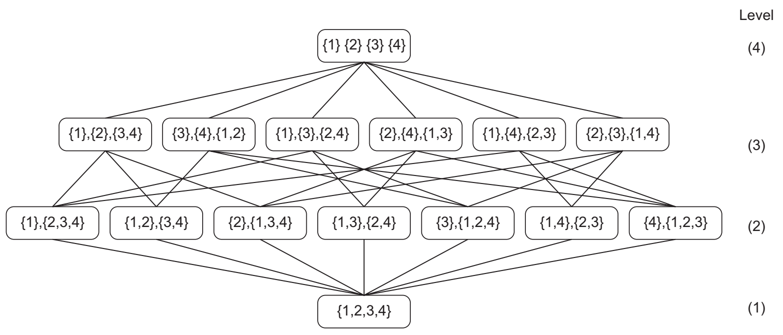


Fig. 2 Coalition structure graph with 4 agents

or coalition structure $CS(A)$. For the context of MSR reconfiguration, an agent a_i corresponds to a single MSR-module, a coalition S corresponds to an MSR A_j, while a coalition structure $CS(A)$ corresponds to a set of MSRs or a configuration of A. The $succ(CS(A))$ operation corresponds to a pair of MSRs which are part of the configuration $CS(A)$ moving close to each other, aligning and docking with each other to give a new configuration $CS'(A)$. To solve the MSR reconfiguration problem given in Definition 1, we have to find the coalition structure in the CSG that corresponds to the maximum value, i.e., find $CS^*(A) = arg \max_{CS(A) \in \Pi(a)} Val(CS(A))$. In the rest of the paper, for the sake of legibility, we have dropped the argument A from CS^* and CS, assuming that it is appropriately defined based on the context. For convenience with the CSG traversal algorithm in Section 4.2, we define the depth of a node at level l as $d = \mid A \mid -l$. The depth of the root node of the CSG is 1 and that of the bottommost node is $\mid A \mid$.

4.1 Uncertainty in Reconfiguration of Modular Robots

Uncertainty in the operations required to reconfigure modules in ModRED is an important consideration to extract the desired behavior of the robot. Uncertainty is caused by inexact or unexpected operation by the robot, which cannot be calculated accurately *a priori*. We have considered two sources of uncertainty in ModRED, *viz.,* (a) motion uncertainty from robot physics and environment, and, (b) performance uncertainty from robot operation, which are discussed below.

4.1.1 Motion Uncertainty

Unexpected motion and alignment of the robot modules can cause ModRED's behavior to deviate from ideal operation. We have considered three major sources of uncertainty under this category that could affect the reconfiguration process:

(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 $\mathcal{N}(\mu_{du}, \sigma_{du})$.

(ii) Alignment uncertainty is the uncertainty arising out of the angle each MSR in a pair of MSRs needs to rotate before they can align with each other prior to docking. It is modeled as a Gaussian distribution, $\mathcal{N}(\mu_{au}, \sigma_{au})$.

(iii) Environment uncertainty is the uncertainty arising from the operational conditions in the environment due to factors such as obstacles, terrain conditions, surface friction, etc. that affect the movement of a pair of MSRs while moving towards and docking with each other. We consider three discrete values for environment uncertainty, $e_u = \{hi, med, lo\}$. The uncertainty is modeled as a multi-variate half-Gaussian distribution $\mathcal{N}(\mu_{eu}, \sigma_{eu})$. Modeling it as a half-Gaussian distribution allows us to represent it as a folded standard normal distribution where the fold occurs at a cumulative probability of 0.5 i.e. at the mean μ_{eu}.

To combine the Gaussian representing the motion uncertainty, we consider their weighted mean with variance [6]. We associate with each Gaussian a weight that denotes the Gaussian effect on the total motion uncertainty of the robot. These weights are denoted by w_{du}, w_{au}, and w_{eu} respectively, and each weight is given by the inverse of the corresponding Gaussian variance. The weighted mean of the three Gaussian then gives the total motion uncertainty, expressed as a probability, when two MSRs A_i and A_j attempt to connect with each other, as given below:

$$prob(A_i, A_j) = \frac{1}{w_{du} + w_{au} + w_{eu}}(w_{du} \cdot p_{du} + w_{au} \cdot p_{au} + w_{eu} \cdot p_{eu}), \quad (1)$$

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, $w_{du} = \frac{1}{\sigma_{du}^2}$, $w_{au} = \frac{1}{\sigma_{au}^2}$, $w_{eu} = \frac{1}{\sigma_{eu}^2}$.

4.1.2 Performance Uncertainty

When a robot moves in a certain configuration, its performance might vary depending on several factors such as how well the modules are physically connected, how well the modules can lift each others weight, how much battery the modules have, how well multiple MSRs coordinate with each other while operating, etc. To model these operational uncertainties, we assume that the utility received by the set of MSRs has a certain variance around the ideal value given by the value $Val(CS)$. We denote this variance as a lower and upper bound on $Val(CS)$ denoted by $EV_{lb}(CS)$ and $EV_{ub}(CS)$, where $EV_{lb}(CS) = (1 - p_l) \cdot Val(CS)$, $EV_{ub}(CS) = (1 + p_u) \cdot Val(CS)$ and $p_l, p_u \in [0, 1]$. For legibility, from now on we use the term value to refer to EV, without loss of generality.

Uncertain CSG. To integrate the reconfiguration uncertainties into the CSG, we propose an uncertainty augmented structure called the *Uncertain Coalition Structure Graph*(UCSG). Formally, $UCSG = (V, E, w)$, where $V = \Pi \ni CS$ is the set of vertices of the UCSG corresponding to the set of all partitions of the agents or modules, $e_{i,j} \in E : e_{ij} = (v_i, v_j), v_i, v_j \in V, v_j = succ(v_i)$ is the set of edges in the UCSG, and, $w : E \rightarrow [0,1]$ is a weight associated with edge e_{ij}. We have taken $w_{ij} = w(e_{ij}) = prob(A_i, A_j)$ so that it represents the motion uncertainty involved in forming the coalition structure corresponding to v_j starting from the coalition structure corresponding to v_i. Within the context of the UCSG, we formulate the MSR reconfiguration problem as finding the configuration or coalition structure CS^* that has the maximum expected worst-case (lower-bound) value, that is, $CS^* = \arg\max_{CS \in V} EV_{lb}(CS)$.

Because the number of nodes in the UCSG is unchanged from that in the CSG, and is still exponential, exhaustive search techniques are computationally expensive to implement. To address this problem, we describe an intelligent pruning technique to find the node CS^* in the UCSG, as described in the next section.

4.2 Generating and pruning the UCSG

Our pruning strategy for the UCSG is based on the insight that a pair of MSRs that are likely to incur a high amount of motion uncertainty to get connected with each other will lead to a low-value configuration when included as part of any other configuration. Our objective then is to identify such inefficient pairings as soon as their expected value is calculated and prevent further exploration of the nodes of the UCSG that include those pairings. Starting from singleton modules at the root of the UCSG, the earliest such inefficient pairings can be determined is at depth 2, when two singletons come together to form a two-module MSR. We mark such inefficient 2-module configurations as a *bad coalition (BC)*, as defined below:

Definition 2 Bad Coalition. *Let v_1 denote the root of the UCSG and $\mathbf{v_2}$ denote the children of the root node. Then, any $v \in \mathbf{v_2}$ is marked as a bad coalition iff $prob(a_i, a_j) \leq bc_{thr} : (a_i, a_j) \subset v$, where $bc_{thr} \in [0,1]$.*

Nodes generated while exploring the UCSG which include any bad coalitions are pruned immediately. We assume that all modules are within communication range of each other, and, at the beginning of the reconfiguration process, modules communicate their position and angle to each other within a global reference frame.

The basic algorithm for searching the UCSG for CS^* is a uniform cost search as shown in Algorithm 1. The search starts at the root node, where every module is a singleton, and checks its children for bad coalitions. Bad coalitions, if any, are stored in the set BC. The nodes that do not contain any bad coalitions are placed in the set called $OPEN$. The nodes in $OPEN$ are partitioned into two sets, called *unpromising* and *promising* nodes ($\bar{v}^{unprom}$ and $\bar{v}^{prom}$ respectively) based on whether the nodes upper bound lies below or above the highest value of the lower bound, EV^*_{lb} seen thus far. Nodes in each of these partitions are sorted according to their upper bound value EV_{ub} to ensure that within each partition, less promising nodes are inspected and possibly pruned earlier. During each iteration, the algorithm partially prunes the nodes in $\bar{v}^{unprom}$ and then recursively expands the un-pruned nodes in $OPEN$. The best expected lower and upper bound values in the entire UCSG, EV^*_{lb} and EV^*_{ub}, are updated at the end of each iteration. The algorithm explores UCSG nodes up to

Algorithm 1: Algorithm to search for best coalition structure in UCSG

```
searchUCSG(root)
Input: root // set of singleton agents represented as root of UCSG
Output: $CS^*$: coalition structure in UCSG with max. expected value
Calculate $EV_{lb}(root)$ and $EV_{ub}(root)$;
$EV_{lb}^* \leftarrow EV_{lb}(root)$;
$\bar{v}_{root}^{children} \leftarrow$ Generate children of root;
$BC \leftarrow IdentifyBadCoalition(v_{root}^{children})$;
$OPEN \leftarrow \bar{v}_{root}^{children} \setminus BC$;
Sort nodes in $OPEN$ based on $EV_{ub}$;
Partition $OPEN$ into $\bar{v}^{uprom}$ and $\bar{v}^{prom}$ given by:
    $\bar{v}^{unprom} = \{v \in OPEN : EV_{ub}(v) \leq EV_{lb}^*\}$
    $\bar{v}^{prom} = \{v \in OPEN : EV_{ub}(v) > EV_{lb}^*\}$
Prune$(\bar{v}^{uprom} \cup \arg\min_{\bar{v}^{prom}}(EV_{lb}(\bar{v}^{prom})))$;
$d = 2$;
while $d < targetdepth$ do
    $EV_{lb}^* \leftarrow \max(EV_{lb}^*, \max(EV_{lb}(OPEN))$;
    for every node $v \in OPEN$ do
        $\bar{v}^{children} \leftarrow$ Generate children of $v$
            after removing any children nodes with coalitions in $BC$
        $OPEN \leftarrow OPEN \setminus \{v\}$;
        $OPEN \leftarrow OPEN \cup \bar{v}^{children}$;
    Sort nodes in $OPEN$ based on $EV_{ub}$;
    Partition $OPEN$ into $\bar{v}^{uprom}$ and $\bar{v}^{prom}$ given by:
        $\bar{v}^{unprom} = \{v \in OPEN : EV_{ub}(v) \leq EV_{lb}^*\}$
        $\bar{v}^{prom} = \{v \in OPEN : EV_{ub}(v) > EV_{lb}^*\}$
    Prune$(\bar{v}^{uprom} \cup \arg\min_{\bar{v}^{prom}}(EV_{lb}(\bar{v}^{prom})))$;
    $d \leftarrow d + 1$;
$EV_{lb}^* \leftarrow \max(EV_{lb}^*, \max(EV_{lb}(OPEN))$;
$CS^* \leftarrow$ node with $EV_{lb}^*$;
return $CS^*$;


IdentifyBadCoalition($\bar{v}$)
Input: $\bar{v}$: set of nodes belonging to UCSG
Output: $BC$: set of bad coalitions
//$v$ only contains nodes at depth 2 of the UCSG which consist of
//exactly one 2-agent coalition and remaining singleton coalitions.
$BC \leftarrow \{\emptyset\}$;
for every $v \in \bar{v}$ do
    if $\exists(i,j) \in v : u(i,j) << u_2$ then
        $BC \leftarrow BC \cup (i,j)$;
return $BC$;
```

Algorithm 2: Algorithm to prune a set of nodes from the OPEN list in a depth first manner.

```
$Prune(\bar{v})$
**Input**: $\bar{v}$: set of nodes belonging to $OPEN$
**Output**: $winner$: node from $\bar{v}$ with highest expected future utility
**for** $l = 1$ to $k$ **do**
    //for $k$ levels expand branch starting from node $v_j$
    **for** every $v_j \in \bar{v}$ **do**
        $\bar{v}_{j,l}^{children} \leftarrow succ^{(l)}(v_j)$ (after removing any bad coalitions);
        $v_{j,l}^{child} \leftarrow \arg\max_{v \in \bar{v}_{j,l}^{children}} EV_{ub}(v)$;
        **if** $\exists j' : v_{j',l}^{child} = v_{j,l}^{child}$ **then**
            $\bar{v} \leftarrow \bar{v} \setminus \{v_j\}$
            $pruned \leftarrow pruned \cup \{v_j\}$;
        **else**
            $grad_j \leftarrow \frac{(EV_{lb}(v_{j,l}^{child}) - EV_{lb}(v_j)) + grad_j \cdot (l-1)}{l}$;
//Project each line up to $targetdepth$ and select winner
$v_k^{max} \leftarrow \max_{v \in v_{j,k}^{child}} EV_{lb}(v)$;
$grad_k^{max} \leftarrow$ gradient corr. to $v_k^{max}$;
$winner \leftarrow v_k^{max}$;
//find intersection depth of $v_k^{max}$ with every unpruned child of $v_j$ at depth $k$
**for** every $v_{j,k}^{child}$ **do**
    $inter_j \leftarrow \frac{EV_{lb}(v_k^{max}) - EV_{lb}(v_{j,k}^{child})}{grad_{j,k} - grad_k^{max}}$;
**if** $\min_j \mid (targetdepth - inter_j) \mid \leq thresh$ **then**
    $winner \leftarrow \arg\min_j \mid targetdepth - inter_j \mid$;
    //break ties in $winner$ by selecting $winner$ with $(targetdepth - inter_j)$
$pruned \leftarrow pruned \cup (\bar{v} \setminus \{winner\}$;
$OPEN \leftarrow OPEN \setminus pruned$;
return $winner$;
```

a certain depth called $targetdepth$ and returns the node with the highest value of lower bound EV_{lb}^* that has been encountered so far.

The pruning mechanism used for nodes in the unpromising partition, $\bar{v}^{unprom}$, requires some insight. Consider a situation where there is only one node in $\bar{v}^{unprom}$ and $\bar{v}^{prom}$ respectively. Denote these nodes by $v_k \in \bar{v}^{unprom}$ and $v_l \in \bar{v}^{prom}$ and their lower bound values by $EV_{lb}(v_k)$ and $EV_{lb}(v_l)$ respectively. By definition of the promising and unpromising partition, $EV_{lb}(v_k) < EV_{lb}(v_l)$. In the conventional branch and bound algorithm, v_k can be pruned right away as the successor nodes in its subtree cannot improve on the values in v_l's subtree. However this might not be the case in the UCSG because of the operational uncertainty between modules. Let $p_{k1}, p_{k2}, p_{k3} ... p_{kd'}$ denote the operational uncertainty denoted by the weights (w_{ij}) encountered starting from v_k up to a node at depth d' away from v_k's depth. Similarly, let $p_{l1}, p_{l2}, p_{l3} ... p_{ld'}$ denote the corresponding operational uncertainty starting from node v_l up to a node at depth d' below v_l. The values p_{km} and

$p_{ln}(m,n = 1...d')$ are probabilities determined by the operational conditions. Consider a case where for every m,n $p_{km} >> p_{ln}$. In such a scenario, we can have $EV_{lb}(succ^{(d')}(v_k)) > EV_{lb}(succ^{(d')}(v_l))$ implying that d' levels below the current depth, v_k's lower bound might be higher than v_l's lower bound. Therefore, it might be incorrect to make a decision about pruning node v_k as soon as it is placed in $\bar{v}^{unprom}$.

To address this problem, we propose a lookahead-based technique for making a decision to prune a node from the unpromising partition. The main idea of the lookahead technique is to check whether the lower bound of any node in the unpromising partition might exceed the lower bound of the worst node (lowest EV_{ub}) in the promising partition at a lower depth in the UCSG called $targetdepth$. However, expanding all the successor nodes of every node in $\bar{v}$ up to $targetdepth$ is a computationally expensive operation as the number of successor nodes grows exponentially. Therefore, we divide the lookahead process into two steps - (i) a *gradient calculation phase* that expands a node for k successive levels from the current level, retains only the best successor node at each level, and, calculates the change in the value or gradient of EV_{lb} from the current level upto k levels below; (ii) a *projection phase*, that calculates the expected value of EV_{lb} at $targetdepth$ by projecting the last calculated value of EV_{lb} using the gradient. As shown in Algorithm 2, in the gradient calculation phase, each node $v_j \in \bar{v}$, is expanded for k successive levels and only, $v_{j,l}^{child}$ the successor node at each level l with highest value of the upper bound EV_{ub} is retained. After removing duplicate nodes, the change in the value of the lower bound EV_{lb} from v_j up to $v_{j,l}^{child}$ is calculated as $grad_j$. In the projection phase, we first identify the best node (with highest value of EV_{lb}) at the last calculated level $(current\,level + k)$. We call this node *winner* and its associated gradient for EV_{lb} as $grad_k^{max}$. We then inspect each of the remaining nodes $v_j \neq winner$ at $(current\,level + k)$ and calculate the level, $inter_j$, at which v_j's EV_{lb} value exceeds *winner*'s EV_{lb} value. If $inter_j$ is at $targetdepth$ or near (*i.e.* within $thresh$ levels of $targetdepth$), the node $v_j \in OPEN$ having the highest positive difference in EV_{lb} at or near $targetdepth$ with *winner* is marked as the new *winner*. All nodes except the final *winner* are then pruned, while *winner* is added to *OPEN* for expansion in the next iteration step.

5 Experimental Results

To verify the performance of our proposed reconfiguration planning technique, we implemented the searchUCSG algorithm in C++ on a desktop PC (Intel Core i7 - 960 3.20GHz, 12GB DDR3 SDRAM).

Experimental Setting. We consider a setting where a set of $n = 4...25$ ModRED modules are placed randomly within a 4×4 m^2 environment. Initially none of the modules are connected with each other. The objective of the modules is to find the configuration that gives the highest value. To do this, each module runs the *searchUCSG* algorithm given in Algorithm 1. The value of an MSR $A_i \in CS$ is defined as:

$$Val(A_i) = \begin{cases} Val(1), \text{ if } |A_i| = 1; \\ Val(1) \times (|A_i| + \frac{|A_i|^2}{10}), \text{ if } |A_i| \le A_{max}; \\ Val(1) \times e^{-(|A_i| - A_{max}) \times 10}, \text{ if } |A_i| > A_{max}. \end{cases} \quad (2)$$

This value function[3] causes value of an MSR to monotonically increases from a singleton up to a certain size A_{max}, beyond which it decreases exponentially. In other words, it gives preference to forming larger coalitions or MSRs up to certain maximum size A_{max}, which is given by the physical limitations of MSR modules to connect with each other and maneuver while remaining connected. The value of a configuration or coalition structure is given by: $Val(CS) = \sum_{A_i \in CS} Val(A_i)$. For our experiments, we have used $Val(1) = 5$. The values of different parameters used by our algorithm are shown in Table 1. The environment uncertainty is set to *medium* for most experiments, unless otherwise stated.

Parameter	Symbol	Values
Number of agents	n	$\{4....25\}$
Maximum desired coalition size	A_{max}	$2(n=4), 4(n=6), 6(n>6)$
Max. depth explored in UCSG	$targetdepth$	$\{\frac{n}{2}, \frac{2n}{3}, \frac{3n}{4}, n-1\}$
Look-ahead depth	k	$\frac{n - current\,depth}{2}$
Mean and std. dev. for distance uncertainty	p_{du}, σ_{du}	0, $\frac{17}{3}$(comm. range of modules = 17cm)
Mean and std. dev. for angle uncertainty	p_{au}, σ_{au}	$\frac{\pi}{2}, \frac{\pi}{6}$
Mean and std. dev. for env. uncertainty	$p_{eu=hi}, \sigma_{eu=hi}$ $p_{eu=med}, \sigma_{eu=med}$ $p_{eu=lo}, \sigma_{eu=lo}$	$0.66, 0.1$ $0.33, 0.1$ $0, 0.1$
Prob. for estimating EV_{lb} from V_{CS}	p_l	0.5 for $n \le 12$ 0.2 for $n > 12$
Prob. for estimating EV_{ub} from $V(CS)$	p_u	0.2
Bad coalition threshold probability	bc_{thr}	0.1

Table 1 Different parameters used for our simulations

5.1 Simulation Results

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 from UCSG with intelligent pruning. For $4 \le n \le 12$, 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 n. The time taken to find the optimal value with our intelligent pruning technique is given in Figure 3(a). For $n > 12$, exhaustive search becomes prohibitive as its complexity is $O(n^n)$. So for more than 12 agents, the highest coalition structure value that our algorithm finds with $targetdepth = n - 1$ (exploring all depths, but with pruning) is used as the *best* value. The ratio between the optimal ($n \le 12$) or best ($n > 12$) value and the value found by our algorithm, for different values of $targetdepth$, are shown in Figures 3 (b) and (c). Figure 3(b) shows that for $n \le 12$ if we vary the exploration

[3] Value or *reward* can be determined from the history of past performances of a coalition.

depth, $targetdepth$ in the UCSG, then for $targetdepth = \frac{n}{2}$, on an average we can get 80% of the optimal value[4]. If we increase the limit to $targetdepth = \frac{2n}{3}$, then the achieved value is almost 90% or above of the optimal value and if we further increase $targetdepth$ to $\frac{3n}{4}$, then it is 95% of the optimal value. Empirically, we can say that our algorithm provides the *worst bound* of 80% if we explore till depth $\frac{n}{2}$ and this *bound* increases as we go explore deeper. For $n = 15...25$ agents, in Figure 3(c), we can see that if we explore upto $targetdepth = \frac{n}{2}$, then the value obtained by our algorithm is 90% of the best value obtained. And for $targetdepth = \frac{2n}{3}$, this ratio increases to almost 95%, whereas for $targetdepth = \frac{3n}{4}$ it is more than 95%. From this set of results, we can say that our algorithm provides the *worst bound* of 90% if we go till depth $\frac{n}{2}$ (for more than 12 agents) and this *bound* increases as we further explore lower depths.

No. of agents	Ratio to opt. value	Runtime (mins)
4	1	0.0006
6	1	0.005
7	1	0.04
8	1	0.2
9	1	1.08
10	1	4.22
12	1	6.35

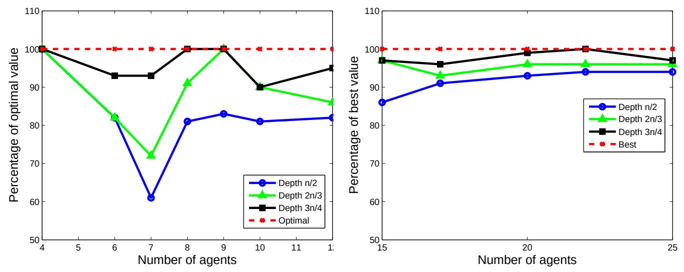
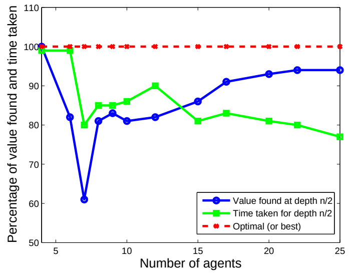


Fig. 3 a) Ratio of values of coalition structure found using searchUCSG algorithm with $targetdepth = n-1$ to the optimal value and corresponding running times taken by searchUCSG algorithm, (b) Ratio of value found by searchUCSG to the optimal found (expressed as percentage) for 4 to 12 agents, (c) Ratio of value found by searchUCSG to the optimal found (expressed as percentage) for 15 to 25 agents

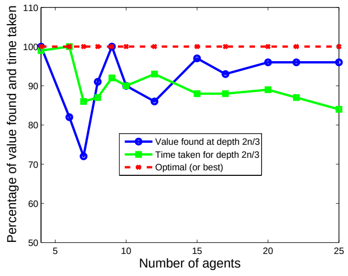
The quality of the solution found by our algorithm is also dependent on the ratio between exploration depth $targetdepth$ and A_{max}. Recall that the value function we have defined, gives the highest value for an MSR (coalition) that is of size A_{max}. The first time a coalition of size A_{max} occurs in the UCSG is at depth A_{max}. This implies that $targetdepth \geq A_{max}$, for our algorithm to be able to find the coalitions with good values, and the higher the value of $\frac{targetdepth}{A_{max}}$ is, the closer the solution is to the optimal or best value. For example, in Figure 3 (a), with $n = 12$ and $targetdepth = \frac{n}{2} = 6$ which is equal to A_{max}, the value found by our algorithm is on an average within 80% of the optimal. But when $n = 25$, and $targetdepth = \frac{n}{2} = 12$ which is greater than A_{max}, the algorithm has already seen several nodes with coalitions of size $A_{max} = 6$ and consequently finds a better solution which is within 94% of the best value.

We have also compared the percentage of optimal(or *best*) value we are getting till a certain $targetdepth$ using our algorithm and percentage of the time taken to

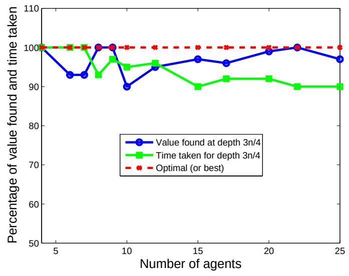
[4] As we have fixed A_{max} at 6, for 7 agents we can find coalitions of size 6 only at depth 6 (i.e. d= n-1). That is why, if we go for $targetdepth = \frac{n}{2}$, then obtained value is only 60% of the optimal value.



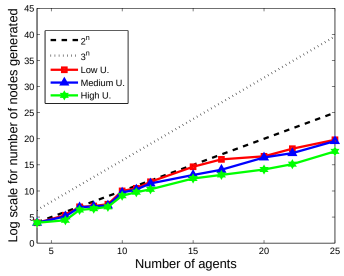
(a)



(b)



(c)



(d)

Fig. 4 Comparison of value found and time taken for different $targetdepth$ values. (a) $targetdepth = \frac{n}{2}$, (b) $targetdepth = \frac{2n}{3}$, (c) $targetdepth = \frac{3n}{4}$; (d) Comparison of number of nodes explored, in log scale, between our algorithm (using low, medium and high environment uncertainty) and 3^n and 2^n times obtained in previous works.

find the optimal value. The results are shown in Figures 4 (a)-(c) for up to 25 agents. We see that as as the number of agents increases, we get values that are closer to the optimal or best value, while taking less time. Also as we explore deeper into the UCSG, this relative difference between the best found value and the time taken increases. As can be seen from the graphs, as $targetdepth$ increases from $\frac{n}{2}$ to $\frac{3n}{4}$, the relative difference between the percentage value found and percentage time taken also increased - the time got successively lower and the best value found got successively higher. This implies that as our algorithm proceeds, the improvement in value of the node found is more than the cost (time) incurred to find the node, that is, our algorithm takes less time to find a better node, as it proceeds further.

We have also compared our result with previously established bounds for the CSG search problem [11, 8], where the complexity of the algorithms were $O(3^n)$ and $O(2^n)$ respectively. As can be seen from Figure 4(d) (shown with log scale on y-axis), using our algorithm, for all three of the environment uncertainty types ($eu = \{lo, med, hi\}$), the number of nodes generated is lower than the other algorithms. For $n < 12$ agents, although the curves for our algorithm's time appear very close to the 2^n line on the log scale, on a linear scale they take an average time of 75% ($eu = lo$), 62.5% ($eu = med$) and 40% ($eu = hi$) of the worst case time of 2^n.

6 Conclusion and Future Work

In this paper we have proposed a MSR reconfiguration planning technique that models the problem as a new data structure called a UCSG and then described an algorithm to intelligently prunes the search within the UCSG to find the best configuration of a set of MSR modules. Currently, our technique starts with all singletons or individual modules and finds the best configuration or partition among them. We are currently extending our algorithm to enable it to start from any configuration of modules and change to the 'best' possible configuration. We are also looking at more structured ways to incorporate the performance uncertainty of modules using agent types within a Bayesian game framework. Yet another direction we are investigating is to automatically determine the optimal $targetdepth$ based on the values of n and A_{max}. Finally, we are working on implementing this algorithm on physical ModRED modules.

References

1. Butler, Z., Brynes, S., and Rus, D. Distributed motion planning for modular robots with unit decompressable modules. In *IEEE/RSJ Intl. Conf. Intell. Rob. and Sys.* (2001), pp. 790–796.
2. Chirikjian, G., Pamecha, A., and Ebert-Uphoff, I. Evaluating efficiency of self reconfiguration in a class of modular robots. *Robotics Systems 13* (1996), 317–338.
3. Chu, K., Hossain, S. G. M., and Nelson, C. Design of a four-dof modular self-reconfigurable robot with novel gaits. *ASME Intl. Design Engg. Tech. Conf. (DETC2011-47746)* (2011).
4. Dasgupta, P., Ufimtsev, V., C.Nelson, and Mamur, S. M. G. Dynamic reconfiguration in modular robots using graph partitioning-based coalitions. In *Intl. Conf. on Auton. Agents and Multi-Agent Systems (AAMAS)* (Valencia, Spain, 2012).
5. Kamimura, A., Yoshida, E., Murata, S., Hurokawa, H., Tomita, K., and Kokaji, S. Distributed Self-Reconfiguration of M-TRAN III Modular Robotic System. In *Intl. J. of Robotics 27(1)* (2008), 373–386.
6. Meier, P. Variance of a weighted mean. *Biometrics 9*, 1 (1953), 59–73.
7. Myerson, R. *Game Theory: Analysis of Conflict*. Cambridge, Massachusetts: Harvard University Press, 1997.
8. Rahwan, T., Ramchurn, S., Jennings, N., and Giovannucci, A. An anytime algorithm for optimal coalition structure generation. *J. Artif. Intell. Res. (JAIR) 34* (2009), 521–567.
9. Ramaekers, Z., Dasgupta, R., Ufimtsev, V., Hossain, S. G. M., and Nelson, C. Self-reconfiguration in modular robots using coalition games with uncertainty. In *Automated Action Planning for Autonomous Mobile Robots* (2011).
10. Rosa, M., Goldstein, S., Lee, P., Campbell, J., and Pillai, P. Scalable shape sculpturing via hole motions. In *IEEE Intl. Conf. Rob. and Auton.* (Orlando, FL, 2006), pp. 1462–1468.
11. Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohme, F. Coalition structure generation with worst case guarantees. *Artificial Intelligence 111*, 1-2 (1999), 209–238.
12. Stoy, K., Brandt, D., and Christensen, D. *Self-Reconfigurable Robots: An Introduction*. Cambridge, Massachusetts: The MIT Press, 2010.
13. Yim, M., and et al. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics and Automation Magazine 14*, 1 (2007), 43–53.