Robotica (2014) volume 32, pp. 225–244. © Cambridge University Press 2014
doi:10.1017/S0263574714000095

searchUCSG: a fast coalition structure search algorithm for modular robot reconfiguration under uncertainty

Ayan Dutta†*, Prithviraj Dasgupta†, José Baca† and Carl Nelson‡

†*Computer Science Department, University of Nebraska at Omaha, Omaha, NE 68182, USA*
‡*Mechanical and Materials Engineering Department, University of Nebraska-Lincoln, Lincoln, NE 68508, USA*

(Accepted January 6, 2014. First published online: February 13, 2014)

SUMMARY
We consider the problem of dynamic reconfiguration by modular self-reconfigurable robots (MSRs) in the presence of uncertainty in their motion and the environment. Specifically, we consider the situation where the MSR is unable to continue its motion in its current configuration and needs to identify a new configuration among the existing modules, which would be the most configuration suitable for performing the robot's assigned task under the current circumstances. To address this problem, we propose a new data structure called an uncertain coalition structure graph (UCSG) that accommodates uncertainty in the MSR's motion and the environment, using a framework from cooperative game theory called the coalition structure graph. We then propose a new search algorithm called *searchUCSG* that intelligently prunes nodes from the UCSG using a modified branch-and-bound technique. We have shown analytically that our algorithm is anytime, that is, if it terminates arbitrarily, it returns the best solution found thus far, which is guaranteed to be within a constant bound from the optimal solution. We have verified the performance of our algorithm experimentally in simulation and shown that it is able to find a solution that is within the worst bound of 80% of the optimal solution while exploring only half of the nodes in the UCSG. Our algorithm also takes lesser computation time than the existing algorithms (that do not model uncertainty) for solving similar problems. Finally, to verify the operation of our algorithm, we have implemented it to partition a set of mobile e-puck robots into clusters and shown how different number of robots and different robot motion uncertainty parameters affect the formed clusters.

KEYWORDS: DARS2012; Modular robots; Multi-robot systems; Space robotics; Automation.

1. Introduction

Over the past few years, modular robots have been proposed as an attractive paradigm for building highly dexterous robotic systems that are capable of maneuvering in environments that are difficult to navigate in. The major advantage offered by a modular robot is that it consists of individual modules that can be dynamically configured into a shape or configuration that is suitable to perform tasks under its current conditions. Consequently, one of the principal challenges in designing modular robots is to solve the reconfiguration problem – given a set of modules in a certain configuration, how to reconfigure these modules into a new configuration while reducing the time and cost expended to identify and achieve the new configuration. This problem is computationally non-trivial if the target configuration is not known *a priori* and the set of all possible configurations has to be explored on-the-fly to find the best possible configuration. Since the space of possible configurations is exponential in the number of modules, conventional search algorithms are unsuitable to solve the reconfiguration

* Corresponding author. E-mail: adutta@unomaha.edu, duttaayan.cs@gmail.com

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

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 modular robot.

To address this problem, we propose a technique that systematically analyzes the expected performance and reconfiguration cost of possible new configurations, and selects the configuration that promises to be the most efficient one in terms of performance and cost. Our proposed technique uses two concepts from cooperative game theory called coalitions and coalition structures. A coalition corresponds to a set of connected MSR modules, while a coalition structure corresponds to disjoint and exhaustive sets of coalitions (connected modules). A convenient way to enumerate different coalition structures is provided by a graphical structure called a coalition structure graph (CSG).[9] To handle uncertainty in modules' movement and the environment, we have augmented the conventional CSG to a new data structure called an uncertain coalition structure graph (UCSG). Each node in the UCSG corresponds to a coalition structure and represents a possible configuration among all the MSR modules. We then formulate the modular robot reconfiguration problem as a graph search problem on the UCSG and propose an algorithm called *searchUCSG* that uses an uninformed search strategy along with a heuristic to direct the search toward the goal configuration, and a branch-and-bound technique[10] to remove graph nodes (configurations) that are impossible to lead to the goal.

We have shown analytically that the *searchUCSG* algorithm is *anytime*,[18] i.e., if prematurely terminated, it can return the best solution found thus far, which is also guaranteed to be within a certain worst case bound of the optimal solution. We have experimentally verified the performance of the *searchUCSG* algorithm through simulations and shown that it is able to find a coalition structure (configuration of modules) that is within the worst bound of 80% of the optimal or best possible configuration while exploring only half the nodes in the UCSG; this bound also improves if we allow the algorithm to run longer and explore more nodes. The *searchUCSG* algorithm also performs better than the existing algorithms for finding the optimal coalition structures (that do not include uncertainty). We have also implemented the *searchUCSG* algorithm on physical e-puck robots and shown that it successfully partitions the set of robots into appropriate clusters for different number of robots and different robot motion uncertainty parameters. The rest of the paper is structured as follows. Section 2 provides an overview of the existing literature on modular robot reconfiguration and the CSG search. Section 3 describes the essential features of the *Modular Robot for Exploration and Discovery* (ModRED) MSR, while Section 4 introduces the dynamic reconfiguration problem in MSRs and provides the *searchUCSG* algorithm. In Section 5, we describe the experimental results of our algorithm on a simulated model of ModRED within the Webots simulator, as well as on mobile e-puck robots, and finally we conclude.

2. Related Work

Modular self-reconfigurable robots (MSRs) are a type of robots that comprise several small modules; the connections between the modules can be changed autonomously by the robot to manifest different shapes or configurations.[21] An excellent overview of the state of the art MSRs and related techniques is given in ref. [23]. 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 reconfiguration problem in MSRs can be classified along three main directions as described below:

- *Search-based reconfiguration*: In this technique, the objective is to find a path, or a sequence of moves among the modules within their configuration space that takes the MSR from an initial configuration to a goal configuration. This technique requires *a priori* knowledge of the goal configuration.[1,2]
- *Control-based reconfiguration*: This technique uses local movement rules for each module and eventually allows the whole MSR to evolve toward the goal configuration. This technique also requires *a priori* knowledge of the goal configuration.[15]
- *Task-driven reconfiguration*: This technique is not focused on achieving a specific goal configuration, rather it tries to get the modules into a configuration that will enable the MSR to perform its assigned task most effectively. In this technique, the MSR's target configuration is

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

selected dynamically based on current conditions (e.g., environment, assigned task, battery power, etc.). This technique offers flexibility in selecting intermediate configurations and gives higher robustness as compared with the techniques that require *a priori* knowledge. This technique has recently shown considerable success[7] in MSR self-reconfiguration.

Our work in this paper is targeted toward task-driven reconfiguration techniques; we do not explicitly specify a goal configuration but allow the reconfiguration algorithm to select a new configuration that is expected to have the best performance under the current circumstances, and simultaneously does not involve excessive reconfiguration costs. For implementing this task-driven reconfiguration technique we have adopted a framework from coalition game theory.

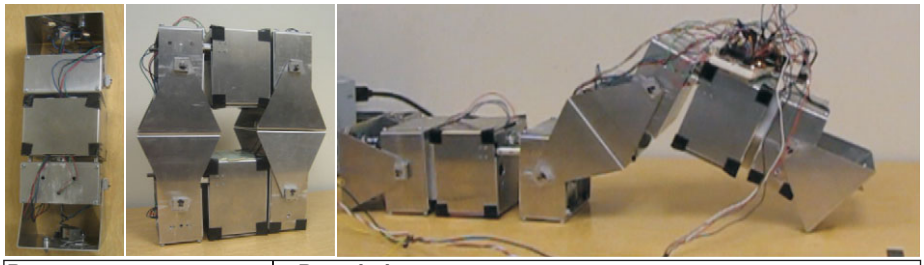
Cooperative or coalition game theory gives a set of techniques that can be used by a group of agents to form teams or coalitions with each other.[9,18] A coalition can be loosely defined as a set of agents that remain together with the intention of cooperating with each other, possibly to perform a task. In terms of MSRs, a coalition represents a set of MSR modules that are connected together. A coalition structure represents a disjoint and exhaustive set of coalitions. In other words, a coalition structure is a partition among the set of agents. In terms of MSRs, a coalition structure represents sets of connected modules. A coalition structure graph (CSG) is usually used to represent the set of all possible coalition structures among a set of agents in a systematic and hierarchical manner. Within coalition games, the CSG search problem that deals with finding the optimal coalition structure or optimal partition among a set of agents, based on a certain objective function, has received significant attention. This problem is known to be NP-complete[18] and many algorithms have been proposed in the last decade to find the optimal coalition structure. These algorithms can be classified into two major classes:

- *Heuristic algorithms*: Heuristic-based search algorithms such as A*[17] or its variants can find relatively good solutions in less time. Also, they scale up well with the number of agents involved. For finding good solutions to search problems on computationally limited processors, such algorithms can be adopted. The main disadvantage of these algorithms is that they do not guarantee any bound on the solution, i.e., if the algorithm faces sudden termination before its completion time, the solution returned can be arbitrarily worse than the optimal solution.
- *Anytime algorithms*: An anytime algorithm[24] first generates an initial solution which is within a bound from the optimal solution, and then as the computation progresses, it successively improves this bound. These algorithms are good for finding an optimal solution. Also, if the algorithm stops before its pre-defined termination state, these algorithms guarantee to return a solution which is within a certain bound from the optimal. This quality makes these algorithms extremely useful. But the disadvantage of these algorithms is that as the bound from the optimal solution gets recalculated at every step, they can end up searching through the entire search space again, resulting in a computation time which can grow in the worst case up to $O(n^n)$ (where n is the number of agents).

Solution concepts used in the above-mentioned two algorithm classes use a variety of techniques ranging from dynamic programming, integer programming, simple heuristics, genetic algorithms to integer partitioning approaches. Sandholm[18] developed an anytime algorithm for the CSG search problem and proved that all the coalitions can be found only in the first two levels in a CSG. So if the algorithm searches the nodes at only these two levels, then it can establish a bound of $\frac{n}{2}$ from the optimal solution. If time permits, the algorithm can continue the search in the remaining graph to improve the bound. The bound has been shown to improve monotonically as more levels are searched. Rahwan[11] proposed a dynamic programming-based approach called IDP which reduced the memory required by a previous dynamic programming-based approach[16] significantly. These algorithms are anytime in nature.

Complementary to these approaches, there are some heuristic algorithms also that do not make any guarantee on the worst case bound of the solution, but they can still yield a good solution in very little time as compared with anytime algorithms. Genetic algorithms have been used to implement these types of heuristic solutions.[19] Shehory and Kraus[20] have provided a decentralized greedy algorithm which takes into account only those coalitions that have size less than a permitted value. Their algorithm also guarantees a worst case bound. Rahwan *et al.*[12] have also developed an integer partitioning-based coalition structure search algorithm that uses a modified branch-and-bound

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press



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: 120 m, transmission power: 1-mW).
Power source	3.7-V Lithium-polymer (Li-Po) battery packs.

Fig. 1. (Colour online) Photographs of ModRED and description of its essential features.

technique to rapidly find the best coalition structure. This algorithm is optimal and anytime. Recently, Dasgupta *et al.*[4] proposed a graph-partitioning-based approach to find the best coalition structure, where agents are represented as nodes, and edges between two nodes determined whether those two agents should belong to the same coalition or not.[4] All these approaches consider ideal environments in the sense that they do not include any uncertainty in the agents' decisions or actions, or in the agents' environment. But in a physical environment, robots are characterized by uncertainty arising out of sensing, motion, and dynamics of the robot. The dynamic and unknown nature of the environment is another source of uncertainty. To address these issues, we describe an augmented CSG called a UCSG (uncertain CSG). Then we propose a new algorithm that includes a branch-and-bound-based pruning technique to rapidly find the best coalition structure within the UCSG.

3. ModRED MSR

The ModRED is a homogeneous modular robot system and has been developed for efficient maneuvering over unstructured surfaces such as extra-terrestrial surfaces such as the moon or mars.[3] Photographs of the ModRED robot are shown in Fig. 1. The robot system is characterized by dexterity of modules and a distributed control architecture. The novel kinematic arrangement of the robot modules is supported by a rotary plate genderless single-sided docking mechanism called RoGenSiD.[6] It enables a module to detach itself from a faulty module which is essential for sustaining the robot system's functionality by means of self-healing.

Each of the ModRED modules has four degrees of freedom (DOF) – three rotational and one prismatic. Each ModRED module has five distinct segments – two end brackets containing the docking interfaces and three box-shaped segments housing the actuators, transmission, circuit components, and power source, as shown in Fig. 2. The module is capable of producing pitch, yaw, and roll and one extension DOF. The four independent DOF are characterized by specific ranges to meet the requirements for generating gaits[22] and reconfiguration. The docking brackets have a rotation range of 90°. Relative to the central box segment, one end segment has an infinite angle of twist whereas the other end box segment has a 0–1 inch (0–25 mm) linear range of displacement. To allow maximum dexterity, all four DOF are made independent, i.e., they have individual actuators. The motivation behind such a design was to keep the design simple with minimal transmission mechanisms. This would result in enhanced robustness by minimizing the parts count and thus minimizing the probability of failure for an overall module. Stepper gear motors are used for the three rotational DOF whereas a stepper linear actuator (lead-screw mechanism) is used for the prismatic DOF.

As an autonomous system, each of the ModRED modules is equipped with necessary electronics to give them such autonomy, i.e., two ATmega328P microcontrollers, rechargeable lithium-polymer

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

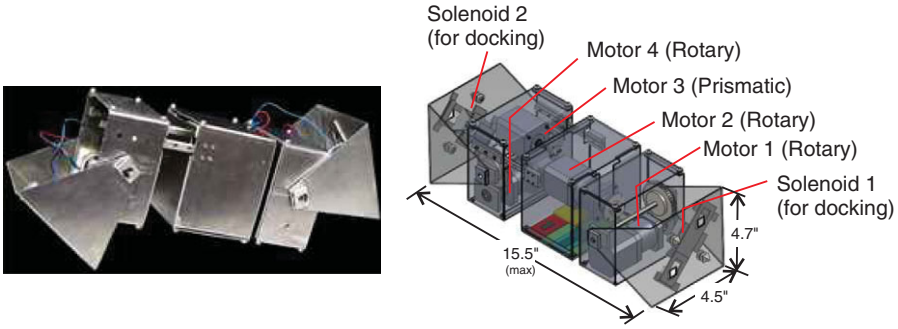


Fig. 2. (Colour online) Different components within a single module of ModRED.

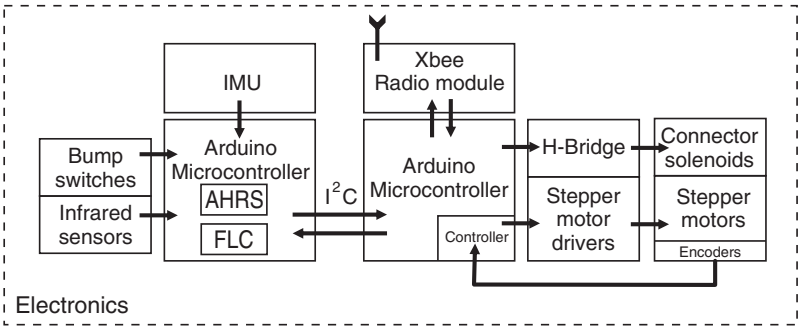


Fig. 3. Electronic architecture of ModRED modules.

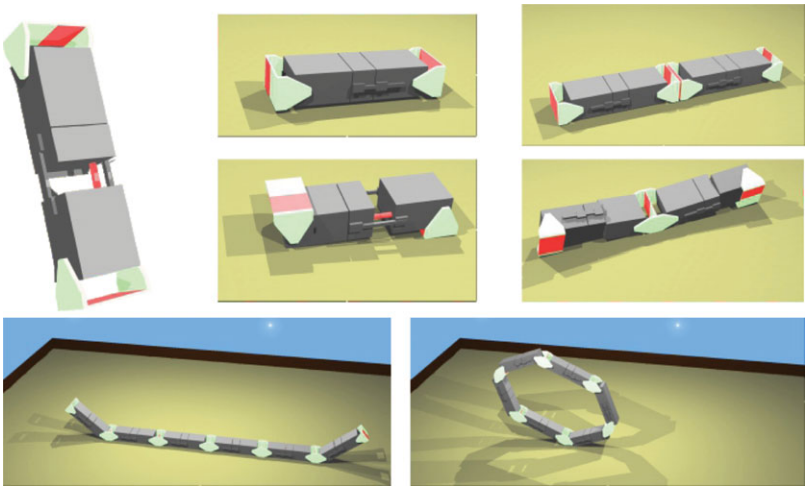
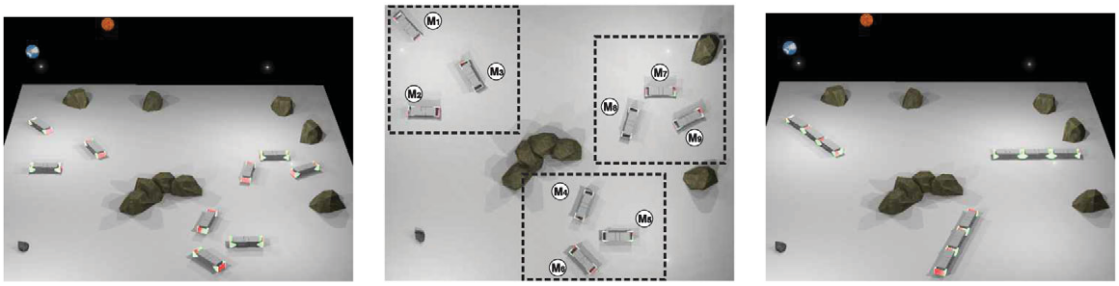


Fig. 4. (Colour online) Simulated ModRED modules within Webots robot simulator. Top-left and top-center: single module; top-right: two-module configurations; bottom: six-module chain and ring configurations.

battery packs, XBee modules to enable wireless communication among modules, one inertial measurement unit (9-DOF Razor), an array of infrared sensors for proximity sensing, and bump sensors for tactile sensing, as shown in Fig. 3.

An accurate 3D model of ModRED has been created using the Webots platform to simulate the execution of different gaits before implementation in the robot, as shown in Fig. 4. Videos of ModRED executing different gait tables with single module and two-module configurations are available at `http://cmantic.unomaha.edu/projects/modred/index.htm`. Figure 4 (top-left) shows a singleton module, where the module is connected with any other module; its (top-right) and (bottom) show configurations of two and six modules respectively.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press



(a) (b) (c)

Fig. 5. (Colour online) An example illustrating the reconfiguration process in ModRED. (a) The initial configuration of modules, (b) The partitions calculated by the *searchUCSG* algorithm, (c) The final configuration of modules.

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 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 $CS(A) = \{A_1, A_2, \ldots, A_k\} \in \Pi(A)$ denote a specific partition of A. We call $CS(A)$ a configuration, and A_i as the ith MSR in that configuration. $A_i = \{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_{|A_i|}}\}$, where a_{i_1} and $a_{i_{|A_i|}}$ are the leading and trailing modules of A_i respectively, and $\{a_{i_j}, a_{i_{j+1}}\}$, $j = 1 \ldots |A_i| - 1$ is the set of physically coupled modules in A_i. When $\mid A_i \mid = 1$, 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 $Val(CS(A))$. Following is the problem that we study in this paper:

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

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

Note that k and k' in the above definition may be different. Such reconfiguration can happen, for example, when a set of modules are 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. Figure 5 shows an illustration of a reconfiguration process done by our proposed *searchUCSG* algorithm to find the optimal partition among a set of nine modules.

In our previous work,[4,13] we have shown that a systematic way to go about analyzing the configurations in $\Pi(A)$ is provided by a hierarchical graph structure called CSG.[9] 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 four agents is shown in Fig. 6. CSG nodes are organized into levels. *Level l* indicates that every node in level l in the 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

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

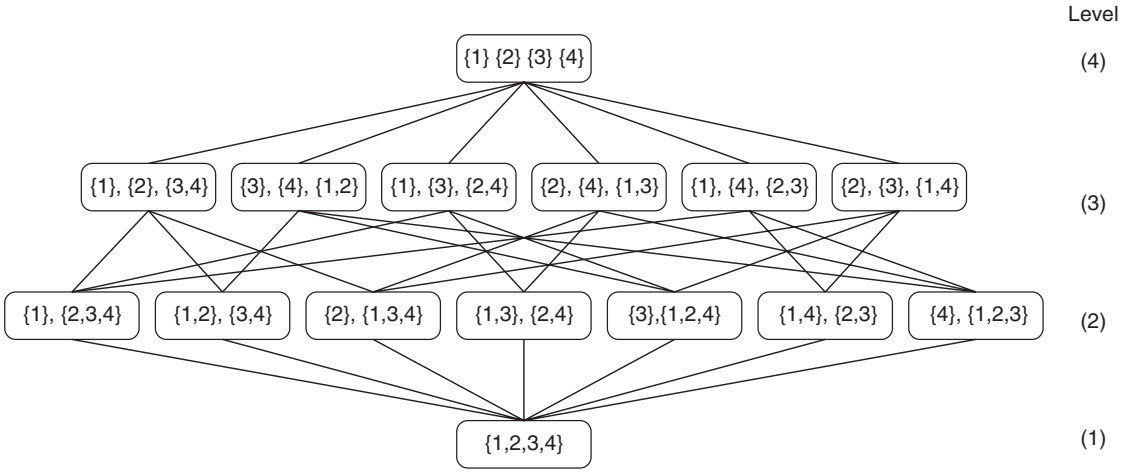


Fig. 6. Coalition structure graph with four agents.

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 for 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 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 that 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 $d = \mid A \mid -l + 1$. The depth of the root node of the CSG is 1 and that of the bottom-most 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.*, motion uncertainty from robot physics and from the environment, and 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 the following three major sources of uncertainty under this category that could affect the reconfiguration process:

1. *Distance uncertainty*: It is the uncertainty arising out of the distance required to be traversed by a pair of MSRs before docking with each other. We posit that when two modules are separated by a large distance, they have a lower chance of docking successfully than when modules are close to each other. This is because the modules far away from each other might encounter terrain features or obstacles while moving close to each other and get displaced from a pose that allows successful alignment and docking of the modules' end connectors. We have modeled the distance uncertainty as a half-Gaussian distribution $\mathcal{N}(\mu_{du}, \sigma_{du})$, as shown in Fig. 7 (left). Figure 8 shows the effect of inter-module distance on the docking between modules. Here the distance between modules M_1 and M_3 is less than the distance between modules M_1 and M_2. So M_1 will have a higher chance of docking successfully and spend less energy if it connects or forms a coalition with M_3 rather than with M_2.

2. *Alignment uncertainty*: It is the uncertainty arising out of the angle each MSR in a pair of docking MSRs needs to rotate before they can align with each other prior to docking. As in the case

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

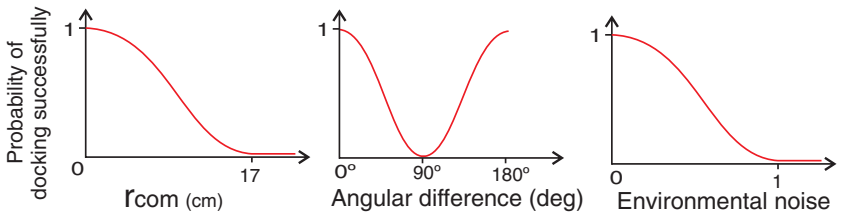


Fig. 7. (Colour online) Probability of a pair of MSRs to dock successfully with each other for (left) distance between the MSRs, (middle) angular difference between the MSRs, and (right) environment noise values.

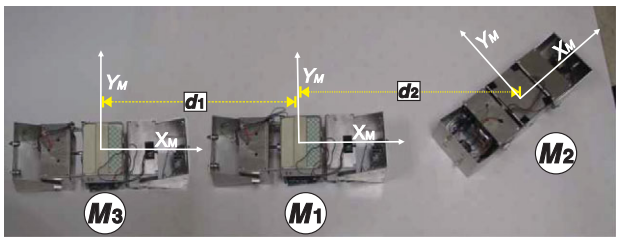


Fig. 8. (Colour online) Three ModRED modules where modules M_3 and M_1 are closer to each other than modules M_2 and M_1. Here M_1 will have higher probability of docking successfully with M_3 than with M_2.

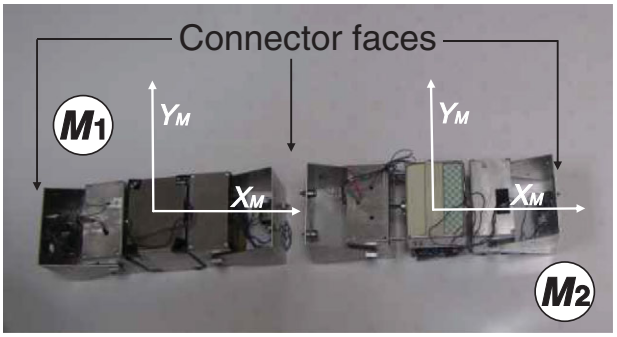


Fig. 9. (Colour online) Angle between modules is 0° (or 180°); easier for modules to connect and they will have a high probability of docking successfully.

of distance uncertainty, larger angular separation between the two docking modules results in a lower chance of successful docking. Note that because the end connectors on either end of a module are identical, therefore two modules are most aligned when the rotational difference between them is 0° or 180°, while they are most unaligned when the rotational difference between them is 90°. We have modeled the alignment uncertainty between two modules as a Gaussian distribution, $\mathcal{N}(\mu_{au}, \sigma_{au})$, as shown in Fig. 7 (middle). The effect of angular difference between two modules on coalition formation is shown in Figs. 9 and 10. If the rotational difference between modules is 90° (Fig. 10), then it will be more difficult for the modules to rotate, align, and dock successfully, and in this process they will have a considerable amount of energy. But on the other hand, if the modules are already aligned, i.e., rotational difference between modules is 0° or 180° (Fig. 9), then it will be easier for them to connect and they will spend less energy in the docking process.

3. *Environment uncertainty*: It 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 toward 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 $\mathcal{N}(\mu_{eu}, \sigma_{eu})$. It is modeled as half-Gaussian simply due to the fact that it is specifically a folded standard normal distribution where the fold occurs at a cumulative probability of 0.5, i.e., at the mean, which is zero, as shown in Fig. 7 (right).

To combine the Gaussians representing the motion uncertainty, we consider their weighted mean with variance.[8] As proved in ref. [8], if the populations of types of data have a common mean

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

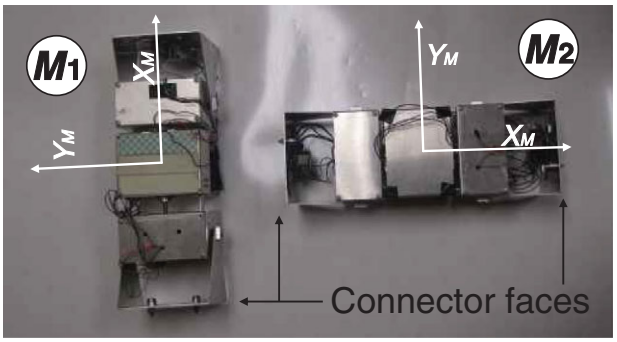


Fig. 10. (Colour online) Angle between modules is 90°; modules will have lower probability of docking successfully.

but different variances, then the best possible way to combine the data is to calculate the weighed mean of the given data where weights are inverse of the variance estimates. We associate with each Gaussian a weight that denotes the Gaussian's 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 Gaussians 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:

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

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

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 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 the 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, $CS^* = \arg\max_{CS \in V} EV_{lb}(CS)$.

Since 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

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

the nodes of the UCSG that include those pairings. Starting from singleton modules at the root of the UCSG, the earliest such inefficient pairings that can be determined is at depth 2 when two singletons come together to form a two-module MSR. We mark such inefficient two-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 coalition 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 in 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 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 the 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 $p_{k1}, p_{k2}, p_{k3} \ldots p_{kd'}$ 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 $p_{l1}, p_{l2}, p_{l3} \ldots p_{ld'}$ 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 $p_{ln} (m, n = 1 \ldots d')$ are probabilities determined by the operational conditions. Consider a case where for every m, n $p_{km} \gg p_{ln}$. 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 $v^{child}_{j,l}$ the successor node at each level l with the 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 $v^{child}_{j,l}$ is calculated as $grad_j$. In the projection phase, we first identify the best node (with the 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 $grad^{max}_k$.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

Algorithm 1: Algorithm to search for the 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 $EV_{ub}(root)$;
$EV^*_{lb} \leftarrow EV_{lb}(root)$;
$\bar{v}^{children}_{root} \leftarrow$ Generate children of root;
$BC \leftarrow IdentifyBadCoalition(v^{children}_{root})$;
$OPEN \leftarrow \bar{v}^{children}_{root} \setminus BC$;
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;

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.

Theorem 1. searchUCSG algorithm is anytime.

Proof: Let $EV^*_{lb,d}$ denote the best value of the expected lower bound found in a node at depth d and v_{opt} denote the node that gives the optimal solution. Let $\alpha_d = \frac{EV^*_{lb,d}}{EV_{lb}(v_{opt})}$. Since the *searchUCSG*

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

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 $winner$;

algorithm starts at the root of the UCSG where $d = 1$, we get $\alpha_1 = \frac{EV_{lb,1}^*}{EV_{lb}(v_{opt})}$. Note that $\alpha_1 \leq 1$ and it represents the initial or worst case bound from the optimal value; α_d then represents the bound from the optimal value at depth d. The update rule of EV_{lb}^* given in Algorithm 1, $EV_{lb}^* \leftarrow \max(EV_{lb}^*, \max(EV_{lb}(OPEN))$, implies that EV_{lb}^* either remains unchanged or it is updated to a higher value. Therefore, it follows that $EV_{lb,d}^* \geq EV_{lb,d-1}^*$, and, consequently, $\alpha_d \geq \alpha_{d-1}$. In other words, as the algorithm explores further down the UCSG and d increases, α_d approaches 1. Therefore, the initial worst case bound is continuously improved as the *searchUCSG* algorithm proceeds, and if the algorithm terminates abruptly at depth d, it returns a node that has EV_{lb}'s value within a bound of α_d from the optimal solution. This proves the anytime property of the *searchUCSG* algorithm.

5. Experimental Evaluation

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 3.20 GHz, 12 GB DDR3 SDRAM). We also implemented our algorithm on physical e-puck robots.

Experimental setting. For simulation, we consider a setting where a set of $n = 4 \ldots 25$ 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

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

Table I. Different parameters used for our simulations.

Parameter	Symbol	Values
Number of agents	$\|A\|$	$\{4 \ldots .25\}$
Maximum desired coalition size	n_{max}	$2(n = 4), 4(n = 6), 6(n > 6)$
Maximum 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 standard deviation for distance uncertainty	p_{du}, σ_{du}	0, $\frac{17}{3}$(comm. range of modules = 17 cm)
Mean and standard deviation for angle uncertainty	p_{au}, σ_{au}	$\frac{\pi}{2}, \frac{\pi}{6}$
Mean and standard deviation for env. uncertainty	$p_{eu=hi}, \sigma_{eu=hi}$	0, 0.6
	$p_{eu=med}, \sigma_{eu=med}$	0, 0.3
	$p_{eu=lo}, \sigma_{eu=lo}$	0, 0.1
Prob. for estimating EV_{lb} from V_{CS}	p_l	0.5 for $n \leq 12$
		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

value of an MSR $A_i \in CS$ is defined as:

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

This value function causes value of an MSR to increase monotonically from a singleton up to a certain size n_{max}, beyond which it decreases exponentially. In other words, it gives preference to forming larger coalitions or MSRs up to certain maximum size n_{max}, which is given by the physical limitations of MSR modules to connect with each other and maneuver while remaining connected. Larger value for bigger coalitions has been studied and used earlier in ref. [14]; our value function follows this formulation. However, forming excessively large coalitions can create complications for maneuvering MSRs – when too many modules are connected together, each module is no more capable of lifting the weight of other modules; synchronization and coordination issues between modules comprising the MSR prohibit the desired movement of the MSR and maintenance costs for maneuvering the MSR also become very large.[20] To address these issues, our value function gives preference to larger coalitions only up to a certain size n_{max}, beyond which larger coalitions are penalized by yielding lower values. [1] The preferred size of a coalition is relevant to the task assigned to the MSR. For example, consider the desired motion of the MSR to cross an obstacle is to tumble along in a ring shape. As shown in a snapshot of ModRED from the Webots simulator in Fig. 4 (bottom-right), the minimum number of connected modules that gives the MSR sufficient traction for its tumbling motion is 6. Then the value of n_{max} is set to 6. 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 I. The environment uncertainty is set to *medium* for most experiments, unless otherwise stated.

For physical robot implementation, we used four or five e-puck robots, which were randomly positioned in the environment. Each e-puck is attached with a colored triangle-shaped marker on its top to denote its orientation. An overhead camera is used to record the positions and orientations of each e-puck. The data are transferred to a computer where the Roborealm image processing software is used to determine the coordinates and heading of the robot on a 2D coordinate system. We take successive samples with the initial positions and orientations of the robots. Then we run the *searchUCSG* algorithm in a computer within the Webots robot simulator using the average of the sampled data as input. The algorithm returns the best coalition structure, i.e., best teams or coalitions to form, as output. When two teams are formed, we designate a distinct goal location for each team.

[1] Both the value function and appropriate value of n_{max} can be learned from the history of tasks done by the MSR, and the efficiency in performing those tasks. Since the focus of this work is on the search algorithm for finding the best configuration of modules, we do not consider the learning problem further in this paper.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

Table II. Ratio of values of coalition structure found using the *searchUCSG* algorithm with $targetdepth = n - 1$ to the optimal valued coalition structure found using exhaustive search, and corresponding running time taken by the *searchUCSG algorithm*.

No. of agents	Ratio to opt. value	Runtime (min.)
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

The goal locations do not affect the operation of the *searchUCSG* algorithm in any way; they are chosen in such a way that the partition between the robots into two clusters is clearly visible. A team or coalition of e-puck robots correspond to a configuration in ModRED or a set of connected modules. Different noise types were incorporated in some of the test cases – $\{lo, mid, hi\}$ noise were incorporated in terms of number of samples taken from the camera. For example, for *hi* noise we took 10 samples, for *mid*, we took 100, and for *lo* noise we took 1000 samples. Fewer samples give higher localization error. Therefore, when noise is *hi*, the environment uncertainty p_{eu} was set to a value higher than 0.66; when noise is *mid*, the environment uncertainty was set to a value between 0.33 and 0.66; and when noise is *lo*, the environment uncertainty was set to a value lower than 0.33. We used *mid* noise for test cases if not mentioned otherwise. For legibility, in the rest of this section, we have used n to denote the number of agents, $|A|$.

5.1. Simulation results

In the first set of experiments, we analyzed the performance of the main concept of our algorithm, i.e., finding the best coalition structure or configuration among the modules. For $4 \leq n \leq 12$, we were able to do an exhaustive search in the space of all coalition structures to find the optimal coalition structure. So we were able to compare the value of the best coalition structure found by our algorithm with the optimal coalition structure. As shown in Table II, our algorithm is able to find the optimal coalition structure for every value of n in reasonably good time. For $n > 12$, exhaustive search becomes prohibitive as its time complexity grows as $O(n^n)$. 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 ratios between the optimal ($n \leq 12$) or best ($n > 12$) value and the value found by our algorithm, for different values of $targetdepth$, are shown in Figs. 11 and 12. Figure 11 shows that for $n \leq 12$ if we vary the exploration depth, $targetdepth$ in the UCSG, then for $targetdepth = \frac{n}{2}$, on an average we can get 80% of the optimal value.[2] If we increase the limit as $targetdepth = \frac{2n}{3}$, then the achieved value is almost 90% or above 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 explore deeper. For $n = 15 \ldots 25$ agents, in Fig. 12, we can see that if we explore up to $targetdepth = \frac{n}{2}$, 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.

[2] As we have fixed n_{max} at 6, for seven 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 the obtained value is only 60% of the optimal value.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

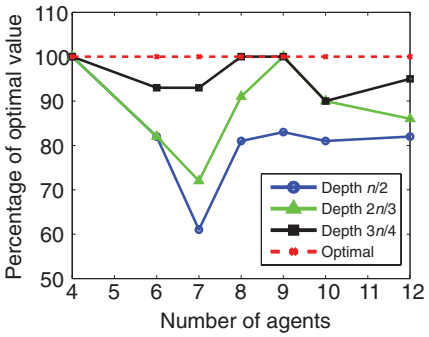


Fig. 11. (Colour online) Percentage ratio of value found by the *searchUCSG* algorithm to the best value found by exploring with pruning up to depth $n-1$ for four to 12 agents when the *searchUCSG* algorithm explores up to different *targetdepths* of $\frac{n}{2}$, $\frac{2n}{3}$, and $\frac{3n}{4}$.

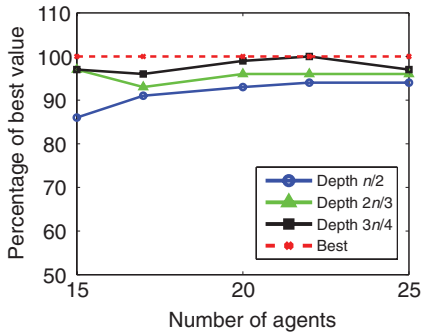


Fig. 12. (Colour online) Percentage ratio of value found by the *searchUCSG* algorithm to the best value found by exploring with pruning up to depth $n-1$ for 15 to 25 agents when the *searchUCSG* algorithm explores up to different *targetdepths* of $\frac{n}{2}$, $\frac{2n}{3}$, and $\frac{3n}{4}$.

The quality of the solution found by our algorithm is also dependent on the ratio between exploration depth $targetdepth$ and n_{max}. Recall that the value function we have defined, gives the highest value for an MSR (coalition) that is of size n_{max}. The first time a coalition of size n_{max} occurs in the UCSG is at depth n_{max}. This implies that if $targetdepth \geq n_{max}$, then the *searchUCSG* algorithm will find better nodes than if $targetdepth < n_{max}$. Thus, the higher the value of $\frac{targetdepth}{n_{max}}$, the closer is the solution to the optimal. For example, in Fig. 11, with $n = 12$ and $targetdepth = \frac{n}{2} = 6$, which is equal to n_{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 n_{max}, the algorithm has already seen several nodes with coalitions of size $n_{max} = 6$ and consequently finds a better solution which is within 94% of the best value.

For our next experiment, we have analyzed how our algorithm's result compares with the optimal value (for exhaustive search up to $n \leq 12$) and the best value (with pruning, for $12 < n \leq 25$). To do this, we compared the ratio (expressed as a percentage) between the value of the coalition structure found by our algorithm and the value of the optimal or best coalition structure. Simultaneously, we also compared the percentage ratio between the times to find the time required to find a solution using our algorithm and the optimal or best solution. The results are shown in Figs. 13(a)–(c) for up to 25 agents. We see that 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 in Figs. 13(a)–(c), as $targetdepth$ increases from $\frac{n}{2}$ to $\frac{3n}{4}$, the relative difference between the percentage value found and the percentage time taken also increases – the time gets successively lower and the best value found gets successively higher. This implies that as our algorithm proceeds, the improvement in the value of the node found is more than the cost (time) incurred to find the node, that is, our algorithm takes successively lesser time to find a better node as it proceeds further.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

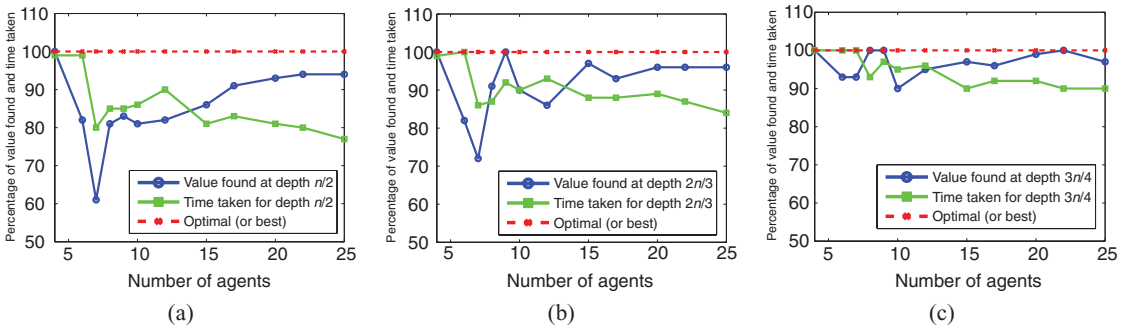


Fig. 13. (Colour online) Comparison of (1) percentage ratio between the value found up to $targetdepth$ and the best value after the completion of the search, and (2) percentage ratio between the time taken to explore up to $targetdepth$ and the total time taken for the completion of the search, for $targetdepths = \frac{n}{2}, \frac{2n}{3}$, and $\frac{3n}{4}$.

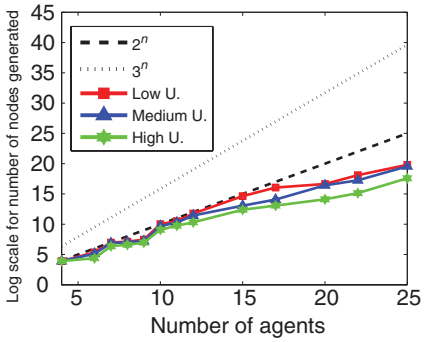


Fig. 14. (Colour online) Comparison of number of nodes explored by the *searchUCSG* algorithm and two other CSG search algorithms with worst case performance of 2^n and 3^n respectively for different values of environment uncertainty. The results are shown on a log scale.

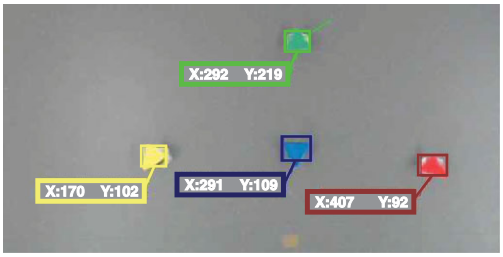
We have also compared the number of nodes generated by the *searchUCSG* algorithm with the number of nodes generated in the worst case by two previously existing algorithms for the CSG search problem,[12,18] where the complexity of the algorithms was $O(2^n)$ and $O(3^n)$ respectively. As can be seen in Fig. 14 (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 by our *searchUCSG* algorithm is lower than other algorithms. For $n < 12$ agents, although the curves for our algorithm 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.

5.2. E-puck implementation

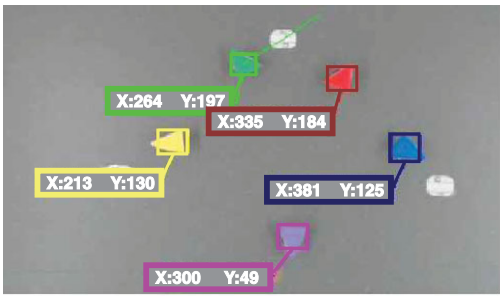
We have implemented our algorithm on e-puck robots to show how it behaves in real environment situations.[3] We have tested with four and five e-pucks inside an arena within our laboratory. The e-pucks were initially placed randomly in the arena. Positions and angles were sampled from the data taken by an overhead camera as shown in Fig. 15. Our main objective was to find the best possible coalition structure, i.e., disjoint clusters or sets of teams or coalitions among the robots using the *searchUCSG* algorithm. As we have assumed that the ModRED modules are randomly distributed as singletons initially, one e-puck robot can be considered to correspond to a single ModRED module. And, although e-puck robots cannot physically connect with or grip each other, we can still verify the operation of the *searchUCSG* algorithm by partitioning the e-puck robots into clusters and looking upon a cluster of e-pucks as a set of connected modules in ModRED. Each cluster of e-puck robots represents one MSR of ModRED modules. We have showed how the uncertainty model affects the outcome of the algorithm. For four e-pucks, we did sets of experiments varying n_{max} value through

[3] Currently only two ModRED modules have been built and the overall construction of ModRED modules is still not ready for reconfiguration experiments involving docking and undocking of modules.

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press



(a)



(b)

Fig. 15. (Colour online) Positions sampled from overhead camera for (a) 4 e-pucks and (b) 5 e-puck robots.



Fig. 16. (Colour online) Five e-pucks used for our experiments (left), initial positions (middle), forming two clusters corresponding to the best coalition structure determined by the *searchUCSG* algorithm (right).

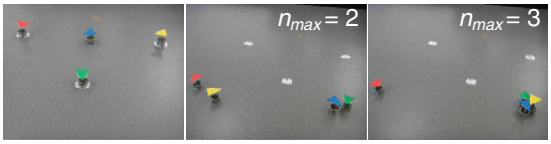


Fig. 17. (Colour online) Five e-puck robots starting with different initial orientations (left) and forming two clusters based on the partition calculated by the *searchUCSG* algorithm with $n_{max} = 2$ (middle) and $n_{max} = 3$ (right).

2 and 3. For every n_{max} value, we used four different sets of experiments. For the first case, we kept the distances between the robots fixed and performed two sets of experiments varying their angular positions. For the second case, we kept the angular orientation of the robots fixed and performed two sets of experiments while varying the distances between them. For five e-pucks we experimented with $n_{max} = 3$ and for one set of distance and orientation only.

Figure 16 (left) shows the five e-pucks that we used for our first experiment; (middle) shows a snapshot of the initial scenario; and (right) shows the final positions of the robots with two clusters, one with two and another with three robots, where the clusters are formed according to the partition determined by the *searchUCSG* algorithm. For our next experiment, we ran two sets of experiments where we kept the initial positions of the robots same but varied their initial orientation. The results of the experiments using four e-puck robots are shown in Figs. 17 and 18. Because of different initial orientations, the best coalition structure, i.e., final clusters calculated by the *searchUCSG* algorithm were different when n_{max} was set to 2 and 3 respectively. In our next experiments, we tested a similar scenario as the last experiment – we kept the initial orientations of the robots fixed between

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press



Fig. 18. (Colour online) The same experiment as shown in Fig. 17 with the same initial robot positions but different initial orientations (left). The robots once again form two clusters based on the partition calculated by the *searchUCSG* algorithm with $n_{max} = 2$ (middle) and 3 (right).

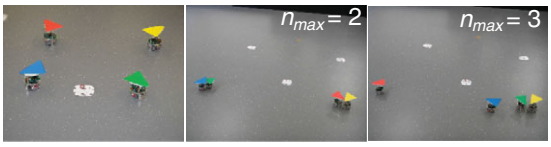


Fig. 19. (Colour online) Five e-puck robots starting with different initial positions (left) and forming two clusters based on the partition calculated by the *searchUCSG* algorithm with $n_{max} = 2$ (middle) and 3 (right).

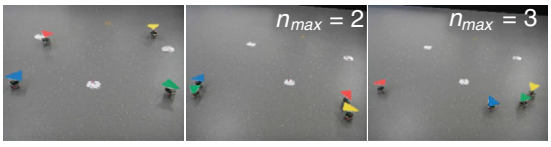


Fig. 20. (Colour online) The same experiment as shown in Fig. 19 with the same initial robot orientations but different initial positions (left). The robots once again form two clusters based on the partition calculated by the *searchUCSG* algorithm with $n_{max} = 2$ (middle) and 3 (right).

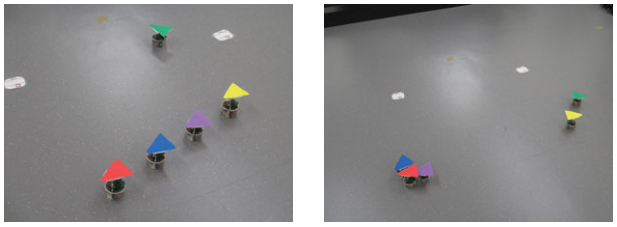


Fig. 21. (Colour online) Five e-pucks with *lo* localization noise: Initial configuration (left), and final configuration (right).

experiments while we changed their initial positions. The results are shown in Figs. 19 and 20. Since the probability of successful docking is affected more by the relative orientation between modules (robots) and to a lesser extent by the distance between them, in Eq. (1) we allocated a higher weight to the angular differences between robots than to the distances between them. Consequently, the final clusters were the same in these two sets of experiments. These experiments demonstrated the main idea of finding the best coalition structure under uncertain, dynamic conditions and how they change depending on current environmental scenario.

In our final set of experiments we demonstrated the effect of noise in the environment on coalition formation. We modeled the effect of noisy environment as an error in localizing the robot using the position information from the image data obtained by the overhead camera. The noise level was varied between *lo* and *hi*. The experiments were done once again with five e-pucks; n_{max} was set to 3. Two different instances of this test case are shown in Figs. 21 (low noise environment) and 22 (high noise environment). In the low noise environment, the *searchUCSG* algorithm finds the best coalition structure where the {red, blue, purple} e-pucks were in the same cluster and {green, yellow} e-pucks formed another cluster. When the noise was changed to high, {red,blue,yellow} e-pucks form a cluster and {purple,green} e-pucks formed the second cluster. This experiment showed that our

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

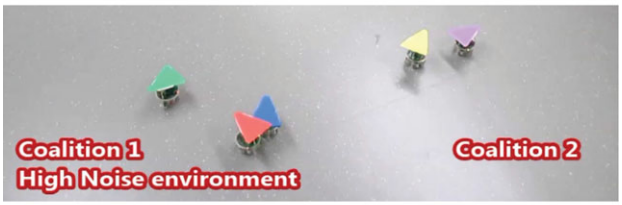


Fig. 22. (Colour online) Five e-pucks in *hi* noise environment showing their final configuration, the initial configuration was the same as in Fig. 21(left).

algorithm adapts to the environmental noise and still finds the best coalition structure. Videos of our different test cases are also provided as attachment.

6. Conclusion and the Future Work

In this paper we have proposed an MSR reconfiguration planning technique that models the problem using a new data structure called a UCSG. We then described an algorithm to intelligently prune the search space within the UCSG to find the best configuration of a set of MSR modules. We showed that our algorithm takes less time in terms of number of nodes explored than the existing algorithms. To the best of our knowledge, our algorithm is one of the first to model uncertainty into coalition formations using a CSG. Also, our algorithm is one of the first that proposes a coalition game theory-based solution to solve the MSR self-reconfiguration problem. Our algorithm also scales well with the number of agents. In terms of value and time trade-off, we have showed that our algorithm does well. 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 the Bayesian game framework. Yet another direction that we are investigating is to automatically determine the optimal $targetdepth$ based on the values of n and n_{max}. Another area of research we are looking at is how to come up with a structured way to define the value function. For this we are looking at learning techniques to learn the value function from the previous tasks done by the MSR modules. Finally, we are working on implementing this algorithm on physical ModRED modules.

Acknowledgments

The research in this paper was done as part of the ModRED project that is funded by NASA (Grant number NE-NNX11AM14A).

References

1. Z. Butler, S. Brynes and D. Rus, "Distributed Motion Planning for Modular Robots with Unit Decompressable Modules," *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2001) pp. 790–796.
2. G. Chirikjian, A. Pamecha and I. Ebert-Uphoff, "Evaluating efficiency of self-reconfiguration in a class of modular robots," *Robot. Syst.* **13**, 317–338 (1996).
3. K. Chu, S. G. M. Hossain and C. Nelson, "Design of a Four-DOF Modular Self-Reconfigurable Robot with Novel Gaits," *ASME International Design Enggineering Technical Conferences (DETC2011-47746)* (2011).
4. P. Dasgupta, V. Ufimtsev, C. Nelson and S. M. G. Mamur, "Dynamic Reconfiguration in Modular Robots Using Graph Partitioning-Based Coalitions," *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* (2012) pp. 121–128.
5. J. Evans, *Optimization Algorithms for Networks and Graphs*, Vol. 1 (CRC Press, Boca Raton, FL 1992).
6. S. M. G. Hossain, C. Nelson, P. Dasgupta, "RoGenSiD A Rotary-Plate Genderless Single-Sided Docking Mechanism for Modular Self-Reconfigurable Robots, *ASME 2013 International Design Engineering Technical Conferences (ASME IDETC)* (Aug. 4–7, 2013).

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press

7. A. Kamimura, E. Yoshida, S. Murata, H. Hurokawa, K. Tomita and S. Kokaji, "Distributed Self-Reconfiguration of M-TRAN III Modular Robotic System," *Intl. J. Robot.* **27**(1), 373–386 (2008).
8. P. Meier, "Variance of a weighted mean," *Biometrics* **9**(1), 59–73 (1953).
9. R. Myerson, *Game Theory: Analysis of Conflict* (Harvard University Press, Cambridge, MA, 1997).
10. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Trans. Comput.* **100**(9), 917–922 (1977).
11. T. Rahwan and N. Jennings, "An Improved Dynamic Programming Algorithm for Coalition Structure Generation," *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Vol. 3* (2008) pp. 1417–1420.
12. T. Rahwan, S. Ramchurn, N. Jennings and A. Giovannucci, "An anytime algorithm for optimal coalition structure generation," *J. Artif. Intell. Res. (JAIR)* **34**, 521–567 (2009).
13. Z. Ramaekers, P. Dasgupta, V. Ufimtsev, S. G. M. Hossain and C. Nelson, "Self-Reconfiguration in Modular Robots Using Coalition Games with Uncertainty," *Proceedings of the Automated Action Planning for Autonomous Mobile Robots Workshop* (2011).
14. S. Ramchurn, M. Polukarov, A. Farinelli, C. Truong and N. Jennings, "Coalition Formation with Spatial and Temporal Constraints," *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems* (2010) pp. 1181–1188.
15. M. Rosa, S. Goldstein, P. Lee, J. Campbell and P. Pillai, "Scalable Shape Sculpturing via Hole Motions," *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, Orlando, FL (2006) pp. 1462–1468.
16. M. Rothkopf, A. Pekeč and R. Harstad, "Computationally manageable combinational auctions," *Manage. Sci.* **44**(8), 1131–1147 (1998).
17. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (MIT Press, Cambridge, MA, 2008).
18. T. Sandholm, K. Larson, M. Andersson, O. Shehory and F. Tohme, "Coalition structure generation with worst case guarantees," *Artif. Intell.* **111**(1–2), 209–238 (1999).
19. S. Sen and P. S. Dutta "Searching for Optimal Coalition Structures," *Proceedings of Fourth International Conference on Multi-Agent Systems* (2000) pp. 287–292.
20. O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.* **101**(1), 165–200 (1998) (Elsevier).
21. K. Stoy, D. Brandt and D. Christensen, *Self-Reconfigurable Robots: An Introduction* (MIT Press, Cambridge, MA, 2010).
22. D. Winter, Biomechanics and *Motor Control of Human Gait: Normal, Elderly and Pathological* (Waterloo Biomechanics, Ontario Canada, 1991). ISBN 0-88898-105-8.
23. M. Yim *et al.*, "Modular self-reconfigurable robot systems: Challenges and opportunities for the future," *IEEE Robot. Autom. Mag.* **14**(1), 43–53 (2007).
24. S. Zilberstein, "Using Anytime Algorithms in Intelligent Systems," *AI Mag.* **17**(3), 73 (1996).

https://doi.org/10.1017/S0263574714000095 Published online by Cambridge University Press