Autonomous Robots (2019) 43:837–857
https://doi.org/10.1007/s10514-018-9759-9

Distributed configuration formation with modular robots using (sub)graph isomorphism-based approach

Ayan Dutta[1] · Prithviraj Dasgupta[2] · Carl Nelson[3]

Received: 31 January 2017 / Accepted: 18 April 2018 / Published online: 28 April 2018

Abstract
We consider the problem of configuration formation in modular robot systems where a set of modules that are initially in different configurations and located at different locations are required to assume appropriate positions so that they can get into a new, user-specified, target configuration. We propose a novel algorithm based on (sub)graph isomorphism, where the modules select locations or spots in the target configuration using a utility-based framework, while retaining their original configuration to the greatest extent possible, to reduce the time and energy required by the modules to disconnect and connect multiple times to form the target configuration. We have shown analytically that our proposed algorithm is complete and guarantees a Pareto-optimal allocation. Experimental simulations of our algorithm with different numbers of modules in different initial configurations and located initially at different locations, show that the planning time of our algorithm is nominal (order of msec for 100 modules). We have also compared our algorithm against a market-based allocation algorithm and shown that our proposed algorithm performs better in terms of time and number of messages exchanged.

Keywords Modular robots · Configuration formation · Graph isomorphism

1 Introduction

Modular self-reconfigurable robots (MSRs) (Yim et al. 2007) are composed of individual robotic modules which can change their connections with each other to form different shapes or configurations. This configuration adaptability affords a high degree of dexterity and maneuverability to MSRs and makes them suitable for robotic applications such as inspection of engineering structures like pipelines (Enner et al. 2013), extra-terrestrial surface exploration (Knight et al. 2001), information collection (Dutta and Dasgupta 2016) etc. As noted in Neubert and Lipson (2016), the main three advantages of using MSRs over traditional wheeled robots for exploration (e.g., Mars Rover) are: versatility, low cost and robustness. While the self-reconfiguration problem in MSRs has been studied extensively in the literature over the last decade, another fundamental problem, the *configuration formation problem*, has not been studied with that same vigor (Ahmadzadeh and Masehian 2015). The configuration formation problem can be described as follows: we are given a set of modules forming different arbitrary initial configurations that are distributed at different locations within the environment along with a target configuration that needs to be formed at a specified location; the target configuration involves some or all of the modules from the initial configurations. The problem is to select an appropriate subset of modules to occupy appropriate spots or positions in the target configuration, so that, after reaching the selected positions, they can readily connect with adjacent modules and form the shape of the desired target configuration.

As a motivating example, we consider a scenario where a set of modules (either as singletons or as part of different configurations) are collecting information from an environment. To access a specific region of the environment, e.g., an elevated region, they need to form a certain shape (configu-

✉ Ayan Dutta
a.dutta@unf.edu

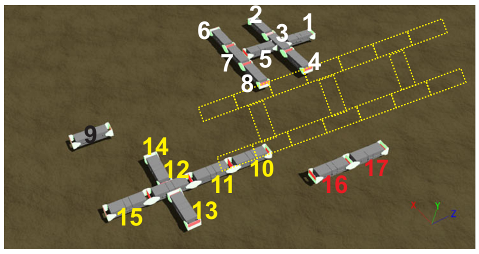
Prithviraj Dasgupta
pdasgupta@unomaha.edu

Carl Nelson
cnelson5@unl.edu

1 School of Computing, University of North Florida, Jacksonville, FL 32224, USA

2 Computer Science Department, University of Nebraska at Omaha, Omaha, NE 68182, USA

3 Mechanical and Materials Engineering Department, University of Nebraska-Lincoln, Lincoln, NE 68588, USA



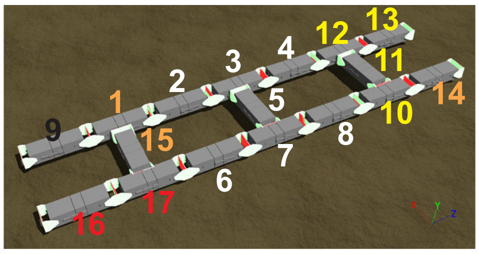


Fig. 1 **a** Four initial configurations consisting of 1, 2, 6 and 8 modules respectively, and desired target configuration (marked with yellow dotted lines), **b** target configuration involving all 17 modules connected in ladder configuration; module numbers marked in white, yellow and red are retained between initial and target configurations (Color figure online)

ration) such as a legged configuration, which allows them to navigate the elevation. An instance of this problem is shown in Fig. 1 where 17 modules initially distributed as singletons or part of smaller configurations go through an online decision making procedure to finally form the ladder-like structure consisting of all 17 modules (Fig. 1b).

In our recent work (Dutta and Dasgupta 2016), we have proposed an algorithm using which singleton modules can form a user-defined configuration while maximizing the amount of information collected on their paths. But in this paper, we have generalized the configuration formation problem from only initial singleton modules to any arbitrary configuration. The generalized configuration formation problem is non-trivial as the modules might already be connected in initial configurations that do not correspond to parts of the target configuration. Also, existing connections between modules in the initial configuration should be preserved in the target configuration, whenever possible, to reduce the energy and time expenditure in disconnecting and re-connecting modules. Moreover, multiple modules from different initial configurations might end up selecting the same most-preferred position (i.e., position involving least time and battery expenditure to navigate to) in the target configuration, leading to failed attempts to achieve the target configuration. To address these challenges, we propose an algorithm that allows modules from initial configurations to select suitable positions in a target configuration using a technique based on graph isomorphism that attempts to improve the utility of the modules by reducing the number of disconnects between modules to achieve the target configuration. [To minimize the number of disconnections among the already existing configurations, our approach uses a graph-based technique to find a (maximum) isomorphic structure of an already existing smaller configuration in the final target configuration]. We have shown analytically that our proposed algorithm is complete and achieves Pareto-optimal allocation. We have also verified the performance of our algorithm in terms of planning time and number of messages exchanged for different numbers of modules and different initial and target configurations. Our experimental results show that our algorithm performs better, in terms of planning time and number of messages passed by the modules, as compared to a market-based allocation algorithm.

1.1 Our contribution

Our contribution in this paper is two-fold. Firstly, to the best of our knowledge, our work in this paper is the first work to address the configuration formation problem where initially modules can be in any arbitrary configuration. Most of the works in the literature assume that initially the modules are singletons which makes the problem a bit less challenging as the problem of simultaneous retaining of the initial configurations for saving time and energy and allocating them to the target configuration does not arise.

Secondly, to solve this unique problem, we propose a decentralized solution which uses concepts from graph theory (e.g., graph isomorphism and maximum common subgraph) to solve it. Although both graph isomorphism and maximum common subgraph have been used before in modular robotic systems to solve the self-reconfiguration problem, i.e., how to change the shape of a configuration without breaking any existing connection among the modules (Hou and Shen 2014; Chirikjian et al. 1996), our approach in this paper uses these graph concepts to solve a different problem—how to merge multiple singletons/configurations into a final target configuration.

2 Related work

Modular self-reconfigurable robots (MSRs) are a type of robots that are composed of several small modules; the connections between the modules can be changed autonomously by the robot to manifest different shapes or configurations (Stoy et al. 2010). An excellent overview of the state of the art in MSRs and related techniques is given in Yim et al.

(2007). Based on architectural properties, modular robots can be divided into three main categories (Yim et al. 2007):

(1) *Chain/tree* In this type of MSR architecture, modules are connected together in a string or tree topology. This type of configuration can fold up to become space filling, but the underlying architecture is serial (Yim et al. 2007).
(2) *Lattice* In this type of architecture, modules are connected in regular, three-dimensional graph structures (e.g., cubic or hexagonal grid). Modules are usually controlled in a parallel manner. Therefore, these configurations follow a liquid-flow like locomotion pattern unlike the worm-like locomotion in chain configurations. In this type of locomotion, each module acts like a molecule of the flowing liquid (Fitch and Butler 2008), (Ahmadzadeh and Masehian 2015).
(3) *Hybrid* This type of architecture is a combination of both chain and lattice type. Modules tend to form large connected networks in hybrid configurations (Yim et al. 2007).

Our proposed approach in this paper is mainly aimed towards chain-type MSRs, but it can be extended to other types of MSRs as well. Over the last decade, the self-reconfiguration problem has been studied extensively by MSR researchers (Ahmadzadeh and Masehian 2015) and it has been proved to be an NP-complete problem (Hou and Shen 2014). Several approaches based on graph theory (Hou and Shen 2008; Asadpour et al. 2008) and control theory (Rosa et al. 2006; Kurokawa et al. 2008) have been proposed. However, as noted in a recent survey on MSRs (Ahmadzadeh and Masehian 2015), configuration formation in modular robot systems has been studied less extensively and the solution approaches proposed so far are not always easy to generalize to all MSR platforms. Configuration formation involves autonomously aggregating modules to form a desired target pattern. A few studies on configuration formation by means of programmable self-assembly can be found for self-actuated modular robots (Klavins 2007), and for modules that lack innate actuation ability, like stochastically-driven modules in a liquid environment (Tolley and Lipson 2010). But these approaches cannot be generalized to all MSR platforms directly.

In swarm robotic systems, configuration formation is known as pattern formation or self-assembly. As the swarm robots are usually not equipped with connectors to connect with other robots, therefore they aggregate nearby to form different patterns. Many studies can be found on autonomous self-assembly of robot swarms. Alonso-Mora et al. (2011) have solved the problem of forming artistic patterns by miniature swarm robots where they are initially distributed arbitrarily (spatially) in an environment and their final objective is to aggregate in such a way that they form the given pattern. Goal positions for robots are specified as Voronoi regions and the Hungarian Algorithm (Kuhn 1955a) is used to allocate robots to goal positions. In Werfel and Nagpal (2008), the authors have provided decentralized movement strategies for robots using random walk, systematic search, or gradient-following to enable them to carry blocks to build user-specified configurations. Recently, we have proposed a semi-distributed solution for configuration formation from singleton modules (Dutta and Dasgupta 2016). A potential limitation of these approaches when applied to MSR configuration formation is that it would require modules already connected in a certain initial configuration to be first disconnected into singletons and then allocated to individual spots in the target configuration, resulting in unnecessary expenditure of energy to undock modules in the initial configuration and possibly re-dock the same modules in the target configuration; inter-module collision avoidance during locomotion of multiple individual modules would also consume more time and energy than when the same modules move together as a connected configuration.

In contrast, our proposed approach attempts to preserve initial configurations in parts of the target configuration wherever possible using graph isomorphism (Cordella et al. 2004) to avoid these issues. Graph isomorphism for MSRs has been investigated by several researchers including Nelson and Cipra (2004), Park et al. (2008) and Hou and Shen (2014), albeit for self-reconfiguring modules (changing positions of modules) that remain part of the same configuration after reconfiguration. For example, in Hou and Shen (2014), the authors have used maximum common subgraph and subgraph isomorphism based techniques for configuration matching which consequently help to find the modules in the configurations which do not need to change positions for the reconfiguration process. As might be expected, past works on the self-reconfiguration problem can contribute techniques which may be partially adaptable to the configuration formation problem. For example, the modeling of modular robot systems as graphs has been used for quite some time, and distance metrics on these graphs are useful for reconfiguration and motion planning (Chirikjian et al. 1996; Pamecha et al. 1997). This representation is also useful for enumerating unique (non-isomorphic) structures (Chen and Burdick 1998; Davis et al. 2016), although it has been pointed out that determining whether a pair of structures are isomorphic to each other is much more difficult than just enumerating the set of unique structures (Chen and Burdick 1998). The Hungarian algorithm based distance metric developed in Chirikjian et al. (1996), Pamecha et al. (1997), is more useful in determining the distance between two configurations where the number of member modules in these two configurations are the same. But in our case, the goal is to find an isomorphic sub-structure of a configuration in a possibly larger target configuration. In this paper, we use a graph representation for the robot system, take advantage of existing algorithms for measuring distance in these graphs, and use the concept of maximum common subgraph (closely related to isomorphism) in addressing the configuration formation problem. Our proposed approach generalizes the self-reconfiguration problem by finding the best positions for multiple configurations and singleton modules within a different, possibly larger or smaller, target configuration.

3 Configuration formation as utility maximization problem

3.1 Notations

Let $\mathbb{A} = \{a_1, a_2, \ldots\}$ denote a set of modules. Each $a_i \in \mathbb{A}$ has an initial pose denoted by $a_i^{pos} = (x_i, y_i, \theta_i)$, where (x_i, y_i) denotes the location of a_i and θ_i denotes its orientation within a 2-D plane corresponding to the environment. Each module has a unique identifier. A configuration is a set of modules that are physically connected. A configuration is denoted as $A_i = \{a_1, a_2, \ldots, a_j\} \subseteq \mathbb{A}$. The topology of configuration A_i is denoted as a graph, $G_{A_i} = (V_A, E_A)$, where $V_A = A_i$ and $E_A = \{e_{kj} = (a_k, a_j) : a_j$ and a_k are physically connected in $A_i\}$. Each configuration has a module that is identified as a leader (Baca et al. 2016) and the leader's pose is used to represent the configuration's pose.

In the configuration formation problem studied in this paper, modules, starting from a set of different initial configurations, are required to get into a specified target configuration. The target configuration is also represented as a graph, denoted by $G_T = (V_T, E_T)$, where $V_T = \{s_1, s_2, \ldots\}$ is the set of vertices and $E_T = \{e_{ij} = (s_i, s_j)\}$ is the set of edges. Each vertex in V_T is referred to as a *spot* that a module needs to occupy and two neighboring spots share an edge between them depending on the topology of G_T. Each spot $s_i \in V_T$ is specified by its pose and its neighboring spots in the target configuration, $s_i = (s_i^{pos}, neigh(s_i))$, where $neigh(s_i) \subset V_T$. Visual representation of these two graph structures has been shown in Fig. 2. Even though we have modeled the initial and target configurations as graphs, for testing purposes, we have used only tree configurations. In the rest of the paper, for the sake of legibility, we have slightly abused the notation by using T instead of G_T to denote the target configuration and S instead of V_T to denote the spots in the target configuration. Let $cost^{loc}()$ denote the locomotion cost from a_i^{pos} to s_j^{pos}, $cost^{dock}$ denote the cost of docking a_i with modules in neighboring spots of s_j and $cost^{undock}$ denote the un-docking costs of a_i from neighboring modules in A_i.

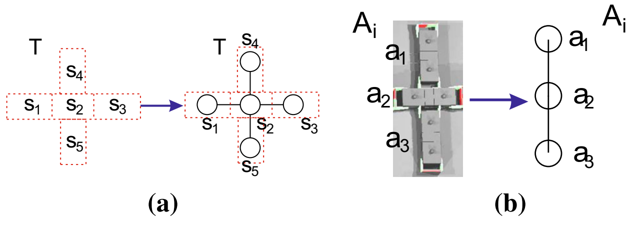


Fig. 2 **a** Graph abstraction of T, **b** Graph abstraction of A_i

3.2 Problem setup

To formulate the configuration formation problem as a utility maximization problem, we first represent the utility of a single module to occupy a single spot in the target configuration, and then extend that representation to a set of modules connected as a configuration to occupy a set of adjacent spots in the target configuration. A single module's utility for a spot is given by the value of the spot to the module minus the costs or energy expended by the module to occupy the spot. As reported in Kamimura et al. (2004), the locomotion of an MSR is significantly affected by the locomotion of the module(s) in the MSR that has more neighbors in the MSR's configuration. For example, for the configuration shown in Fig. 1a, module 12's position at the center of the 6-module configuration is more critical than the other modules for locomotion as it has more neighbors. If module 12 becomes un-operational at any point of time, then four of its connected neighbor modules need to un-dock to get rid of module 12 and then reconnect again to continue working. On the other hand, if any of the terminal modules (e.g., 14) becomes un-operational, then that particular module can be detached from the MSR body with just one un-dock operation.

To capture this position dependency, we have used a concept from graph theory called the betweenness centrality (Brandes 2001) to denote the value of spot s_i, given by:

$$Val(s_i) = \sum_{s_i \neq s_j \neq s_k} \frac{\sigma_{s_j s_k}(s_i)}{\sigma_{s_j s_k}} \tag{1}$$

where $\sigma_{s_j s_k}$ is the total number of shortest paths between any pair of nodes s_j and s_k in G_T and $\sigma_{s_j s_k}(s_i)$ is the number of shortest paths between s_j and s_k which go through s_i.

The cost to a module a_i located at a_i^{pos} to occupy spot s_j at s_j^{pos}, is calculated as a sum of a_i's locomotion costs to reach and occupy spot s_j, and any costs to undock and re-dock with neighboring modules before and after it occupies the spot (Dasgupta et al. 2012). This is denoted as the following:

$$cost_{a_i}(s_j) = cost^{loc}(a_i^{pos}, s_j^{pos}) + \sum_{a_k \in neigh(s_j)} cost^{dock}(a_i, a_k) + \sum_{a_{i'} \in neigh(a_i)} cost^{undock}(a_i, a_{i'}) \quad (2)$$

Note that energy requirements for locomotion of a module are generally higher than those for docking the module with another module as locomotion requires continuous power to all motors and much higher torques than docking; also, docking two modules requires aligning their docking ports first, which takes more energy than un-docking two modules.

When a set of modules is connected in configuration A_i, the cost of occupying a set of spots $S_j \subseteq V_T$ in the target configuration is given by:

$$cost_{A_i}(S_j) = \sum_{s_l \in S_j, a_k \in A_i} cost_{a_k}(s_l) - f_{rwd}(|A_i|) \quad (3)$$

where $f_{rwd}(|A_i|) = \frac{|A_i|-2}{|\mathbb{A}|}$ is a reward function for retaining connections between modules in the existing configuration A_i while being allocated to the target configuration. Because $f_{rwd}(|A_i|)$ increases (and $cost_{A_i}()$ decreases) with the size of A_i, it is cost-wise better to break smaller configurations than to break larger configurations to fit into the target configuration. So, the reward function ensures that keeping the initial configuration intact in the target configuration, whenever possible, results in lower cost. Using the above formulation, it can easily be seen that when A_i can fit entirely into V_T (i.e., $S_j = V_T$), $cost_{A_i}(S_j) < \sum_{s_j \in S_j, a_i \in A_i} cost_{a_i}(s_j)$.

The utility of a spot to a module determines how profitable or beneficial that spot is for the module if it finally ends up occupying that spot. The utility of module a_i for spot s_j is given by

$$U_{a_i}(s_j) = Val(s_j) - cost_{a_i}(s_j) \quad (4)$$

Similar to the cost function described above, the utility for initial configuration A_i to occupy a set of spots $S_j \subseteq V_T$ is given by the sum of the utilities of the individual modules comprising A_i to occupy spots in S_j,

$$U_{A_i}(S_j) = \sum_{s_l \in S_j} Val(s_l) - cost_{A_i}(S_j) \quad (5)$$

Using the above formulation, the spot allocation problem has to assign modules to spots so that each module is allocated to the most eligible (highest utility earning) spot and no two modules are assigned to the same spot.

Formally, we can define the objective function as follows: Given a set of modules $\mathbb{A}$ in a set of initial configurations, and a set of spots S representing the target configuration, find a suitable allocation $P^* : \mathbb{A} \rightarrow S$ such that

$$P^* = \arg\max_{\forall P} \sum_{a_i \in \mathbb{A}, s_j \in S} U_{a_i}(s_j) + \sum_{A_i \subseteq \mathbb{A}, S_j \subseteq S} U_{A_i}(S_j);$$
$$\forall a_k \neq a_i, \quad P^*(a_i) \neq P^*(a_k). \quad (6)$$

Note that, if two modules a_i and a_k both have the same highest utility for spot s_j, then only one of them can be allocated to and occupy s_j. In the next section, we describe our spot selection algorithm that provides a suitable allocation of modules to spots for the above utility maximization problem.

4 Algorithms for configuration formation

We divide the problem into two phases—a *planning phase*, where modules select spots in the target configuration, and an *acting phase*, where modules move to their selected spots and connect with other modules.

4.1 Planning phase

In the beginning of the planning phase, all the modules broadcast their positions and orientations. After having this information, each module calculates the location corresponding to the center target configuration T in the environment, as the mean of all spots' positions. However, a specific desired location can also be given as an input to the modules by the user.[1] Singleton modules then rank themselves according to their distances from the center of T; the rank of a configuration is calculated using the distance of the configuration's leader from the center of T. Singletons and configurations select spots in T based on their rank. Because $cost^{loc}$ has the most significant contribution to the cost function, the distance-based rank ensures that modules and configurations with lower costs (higher utilities) get to select spots in T first. We describe the spot selection techniques in the planning phase in two parts—spot selection by singleton modules and spot selection by configurations.

4.1.1 Spot selection by singleton modules

A singleton module a_{curr} selects a spot to occupy using Algorithm 1. a_{curr} first sorts the spots in order of its expected utility $U_{a_{curr}}(s_j), \forall s_j \in S$. If a spot s_j has not already been selected by another module, or, if it has been selected by another singleton module (module that is not part of a configuration) that can be evicted using the *evict* method, then a_{curr} selects s_j and broadcasts the updated spot-selector pairs to all other modules. If a_{curr} cannot evict the module currently occupying its highest utility spot, then it successively

[1] A common coordinate system can be maintained by modules for localizing themselves following the model described in Suzuki and Yamashita (1997).

Algorithm 1: Spot Allocation Algorithm for Singleton Modules.

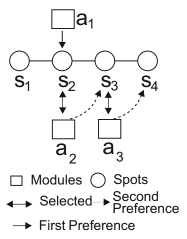
```
1  procedure: spotAllocation()
   Input: $S$: set of spots, $\bar{S}$: set of (spot, selector) pairs; $a_{curr}$:
          module currently selecting spot.
2  $S_{sort} \leftarrow$ Sort $S$ in descending order of utility of spots
3  for each $s_j \in S_{sort}$ do
4      $\mathcal{D} \leftarrow 0$;
5      if ($s_j$ is not selected by another module) $\vee$ (($s_j$ is selected by
       module $a_{block} \notin A_i \subseteq \mathbb{A}$)$\wedge$ (evict($a_{curr}$, $a_{block}$, $\mathcal{D}$) = TRUE))
       then
6          Select spot $s_j$ for $a_{curr}$;
7          Broadcast updated set of spot-selector pairs $\bar{S}$;
8          return;
9  Broadcast NO_SPOT_FOUND message;
```

Algorithm 2: Eviction algorithm used by modules to select alternate spots.

```
1  procedure: evict($a_{curr}$, $a_{block}$, $\mathcal{D}$)
   Input: $a_{curr}$: module currently selecting spot $s_{curr}$; $a_{block}$: the
          module which has already selected $a_{curr}$'s best spot $s_{curr}$;
          $\mathcal{D}$: current recursion depth.
2  if $\mathcal{D} < \mathcal{D}_{max}$ then
3      $s_{block} \leftarrow \arg \max_{s_i \in S \setminus s_{curr}} U_{a_{block}}(s_i)$;
4      $s_{curr'} \leftarrow \arg \max_{s_i \in S \setminus s_{curr}} U_{a_{curr}}(s_i)$;
5      if ($U_{a_{curr}}(s_{curr}) + U_{a_{block}}(s_{block})$ >
       $U_{a_{curr}}(s_{curr'}) + U_{a_{block}}(s_{curr})$) then
6          if $s_{block}$ is not selected by any module then
7              return TRUE;
8          else
9              //$a'_{block} \notin A_i \subseteq \mathbb{A}$ is the module occupying $s_{block}$
10             return evict($a_{block}$, $a'_{block}$, $\mathcal{D}+1$);
11 return FALSE;
```

reattempts spot selection using the spots for which it has the next highest utilities. If none of the spots in S can be selected by a_{curr}, it broadcasts a NO_SPOT_FOUND message to all other modules.

Eviction strategy The *evict method* is used by module a_{curr} to cancel the selection of spot s_{curr} done previously by another singleton module a_{block}. Note that eviction can be done only for a singleton module, and not for modules that are part of configurations, as breaking existing configurations will incur additional time as well as costs for docking and un-docking modules. The method first checks the expected combined utility between a_{curr} and a_{block} for selecting their most (conflicting) and second-most preferred spots. If this combined utility is greater when a_{curr} selects s_{curr} and a_{block} selects its next highest utility spot that it can occupy, then a_{curr} evicts the selection of s_{curr} by a_{block}, as shown in the *evict()* method in Algorithm 2. To limit excessively long cycles of eviction, we have allowed at most $\mathcal{D}_{max}$ successive evictions. An illustration of the eviction process



1. a_1 finds s_2 to be its highest utility spot,
2. a_1 checks if it can evict a_2 occupying s_2,
3. a_2 then checks its next highest utility spot, s_3,
4. a_2 checks if it can evict a_3, occupying s_3,
5. a_3 checks its next highest utility spot s_4,
6.
   - if s_4 is free, a_3 selects s_4, a_2 selects s_3, a_1 selects s_2; return TRUE.
   - if s_4 is not free, $\mathcal{D}_{max} = 3$ is reached; return FALSE.

Fig. 3 Illustration of eviction algorithm for 3 modules with $\mathcal{D}_{max} = 3$

with $\mathcal{D}_{max} = 3$ is shown in Fig. 3. One should note that, eviction and reassigning of modules in a circular fashion cannot happen, i.e., the system will not oscillate because of two main reasons. First, as modules get allocated in a sequential fashion, therefore one module which is already allocated to some spot will only get evicted iff a new spot is found for it. Secondly, we have an upper bound on how many recursive evictions can happen ($\mathcal{D}_{max}$) which makes sure that the eviction does not go on forever.

4.1.2 Block allocation by modules connected in a configuration

Preliminaries Following are some definitions which will be needed in explaining our proposed approach.

Definition 1 Graph Isomorphism (Raymond and Willett 2002): two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a one-to-one mapping between the nodes and edges in G_1 and G_2. Formally, this bijection relationship exists—$f : V_1 \rightarrow V_2$.

Loosely speaking, if two graphs are isomorphic, then they will have same number of nodes and if any two nodes in one graph are adjacent, then those nodes will be adjacent in the other graph as well. Graph isomorphism is one of those problems which are neither solvable in polynomial time nor can they be proved to be NP-complete; rather they belong to an 'intermediate' class (Kobler et al. 2012). Unfortunately, from an algorithmic point of view, even if a problem cannot be proved to be NP-complete problem, being outside of the P-class makes it difficult to solve anyway (in the worst-case scenario). Even though graph isomorphism is a well-known notorious problem to solve (Kobler et al. 2012), there are efficient linear time algorithms available for tree isomorphism (Aho and Hopcroft 1974).

Definition 2 Subgraph Isomorphism (Ullmann 1976): two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are subgraph isomorphic if any subgraph G'_1 of G_1 ($G'_1 \subseteq G_1$) is isomorphic to G_2.

Usually, in the case of subgraph isomorphism, one graph is larger in size than the other and the problem becomes

Algorithm 3: Block Allocation Algorithm that a set of modules connected in configuration A_{curr} uses to select a set of maximally adjacent spots in the target configuration.

```
1  blockAllocation($A_{curr}$, $\bar{S}$)
   Input: $\bar{S}$: Set of (spot, selector) pairs; $A_{curr}$: Set of modules
          connected together as a configuration and currently
          selecting spots.
2  $T_{sub} \leftarrow$ Set of all subgraphs of $T$, which are isomorphic to $A_{curr}$.
3  if $T_{sub} == \{\emptyset\}$ then
4      $T_{sub} \leftarrow$ Set of all maximum common isomorphic subgraphs
       of $T$ and $A_{curr}$.
5  for each $t_k \in T_{sub}$ in descending order of utility $U_{A_i}(t_k)$ do
6      if No spot in $t_k$ has been selected yet then
7          Select $t_k$;
8          Broadcast updated set of spot-selector pairs $\bar{S}$;
9      else
10         $S_{block} \leftarrow$ set of spots $\in t_k$ already selected by
           $\{a_{block}\} \subseteq \mathbb{A} \setminus A_{curr}$
11         $s^i \leftarrow$ spot matched to $a_i \in A_{curr}$ but already selected by
           $a_{block} \in \mathbb{A} \setminus A_{curr}$
12         if evict($a_i$, $a_{block}$) = TRUE for every $s^i \in S_{block}$ then
13             Select $t_k$;
14             Broadcast updated set of spot-selector pairs $\bar{S}$;
15         else
16             if all $t_k \in T_{sub}$ have been checked then
17                 for each $a_i \in A_{curr}$ where evict($a_i$, $a_{block}$) =
                   FALSE and $s^i \in t_k$ do
18                     Disconnect $a_i$ from $A_{curr}$
19                     $A_{curr} \leftarrow A_{curr} \setminus a_i$;
20                     spotAllocation($a_i$, $\bar{S}$);
21                     Broadcast updated set of spot-selector pairs $\bar{S}$;
22             if selected $t_k$ is MCS of $A_{curr}$ then
23                 for every $a_i \in A_{curr}$, where $s^i \notin t_k$ do
24                     Disconnect $a_i$ from $A_{curr}$
25                     $A_{curr} \leftarrow A_{curr} \setminus a_i$;
26                     spotAllocation($a_i$, $\bar{S}$)
27                 Broadcast updated set of spot-selector pairs $\bar{S}$;
28
```

to find a subgraph of the larger graph which is isomorphic to the smaller graph. As can be understood, there can be multiple isomorphic subgraphs available in the smaller graph. This problem is a well-known NP-complete problem (Cordella et al. 2004). However, there are approximation algorithms proposed in the literature which solve the problem in polynomial-time for certain graph structures like trees (Shamir and Tsur 1997). We are also interested in the isomorphic subgraphs which are also maximum in size, which leads us to our next definition.

Definition 3 Maximum common subgraph (MCS): given $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a MCS is a subgraph consisting of the largest number of edges isomorphic to both G_1 and G_2.

The problem of finding a MCS between two graphs is a combinatorially intractable NP-complete problem (Raymond and Willett 2002) for which no algorithm of polynomial-time complexity exists for the general case. For finding all possible MCSs having k nodes in a pair of graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the total number of comparisons we have to do is:

$$\frac{V_1!V_2!}{(V_1 - k)!(V_2 - k)!k!} \tag{7}$$

As can be seen in this equation, even with small values of V_1, V_2 and k, computational comparisons can reach an astronomical value. Although it is a computationally difficult problem to solve in graphs, we should mention that polynomial-time approximation algorithms for finding maximum common subtrees can be found in the literature (Akutsu 1993; Reyner 1977).

As discussed earlier, our main objective is to place the initial configurations (A_i) into the target configuration (T) with the least number of disconnections between the modules present in A_i. We have modeled A_i and T as graphs. Therefore if G_{A_i} is an isomorphic subgraph of T, then A_i can readily be allocated to T (provided the spots are free). On the other hand, if G_{A_i} is not an isomorphic subgraph of T, then we look for a MCS so that we can preserve most of the connections in A_i while allocating it to T while the rest of the modules in A_i which are not part of that MCS are detached from it.

Algorithm description The technique used by configuration A_{curr} to select a set of connected spots in the target configuration T is given by the *blockAllocation* procedure shown in Algorithm 3. The algorithm is executed on l_{curr}, the leader of configuration A_{curr}, selected using techniques in Baca et al. (2016).

To place A_{curr} into T without breaking the connections between its modules, we have to find if T, or a subgraph of T, is isomorphic to A_{curr}. An example of this problem is shown in Fig. 4a that shows all possible subgraphs of T which are isomorphic to the configuration A_i using different colors. This problem requires finding the isomorphic subgraphs (IS) (Cordella et al. 2004) of T. However, if A_{curr} is not isomorphic to T or a subgraph of T, then A_{curr} cannot be placed into T without breaking its connections and, thus, changing its shape. In such a scenario, our objective is to reduce the number of connections that are removed between A_{curr}'s modules. For this, we have to find the maximum number of modules in A_{curr}, which can be placed directly into T, without first disconnecting them. An example is shown in Fig. 4b, where the red dotted boxes indicate the maximum common subgraphs of T and A_i, which are isomorphic.

This problem is an instance of the *maximum common subgraph (MCS) isomorphism* problem as discussed earlier

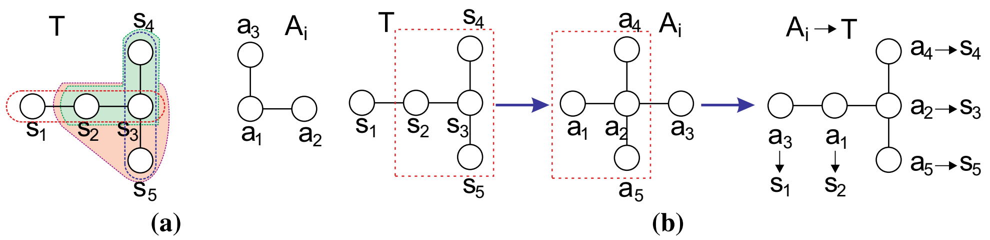


Fig. 4 **a** A scenario where the colored subgraphs of T are isomorphic to A_i, **b** A scenario where a subgraph of t is isomorphic to a subgraph of A_i. The red dotted box shows the maximal common subgraph between T and A_i; the unmatched module a_3 is detatched from A_i and allocated to spot s_1 by our block selection algorithm (Color figure online)

(Raymond and Willett 2002), where, given two graphs T and A_{curr}, the goal is to find the largest subgraph which is isomorphic both to a subgraph of T and A_{curr}. If $|V_{A_{curr}}| > |V_T|$, then we find the maximum size subgraph of T which is isomorphic to $A'_{curr} \subseteq A_{curr}$ and allocate the spots to matched modules, using a similar technique as in the *blockAllocation* algorithm. On the other hand, if $|V_T| = |V_{A_{curr}}|$ and G_T, $G_{A_{curr}}$ are isomorphic, then A_{curr} can be allocated to T; otherwise, we find the MCS between A_{curr} and T which can be readily allocated to T while the rest of A_{curr} can be allocated following the proposed *blockAllocation* algorithm.

Our algorithm first finds subgraphs of G_T that are isomorphic to G_A. If there are no isomorphic subgraphs, it checks for maximal common isomorphic subgraphs. These subgraphs are stored in set T_{sub} (lines 2–4). As modules want to maximize the utility earned from the allocation, the subgraphs t_k within T_{sub} are ordered by utility to A_{curr}. The algorithm then inspects each subgraph t_k. If all the spots in t_k are free, then t_k is selected by A_{curr} and l_{curr} broadcasts a message to notify every module in $\mathbb{A}$ about this selection (lines 6–9). On the other hand, if any spot $s^i \in t_k$ is already selected by a singleton a_{block}, A_{curr} checks to see if it can eviction a_{block} using the *evict()* method. If eviction is successful, t_k is selected for A_{curr} and the updated set of spot-selector pairs are broadcast to all modules in $\mathbb{A}$ (lines 11–15). If eviction is not successful, it means that some modules in A_{curr} could not occupy some spots in the target configuration (or its subgraph) as some other modules that did not belong to configuration A_{curr} had already selected those spots. In this case, the modules of A_{curr} that could not find a spot in t_k will be disconnected from A_{curr}. Single spot selection algorithm is then used to select other spots in t_k for these modules (lines 17–21).

Finally, because selection of t_k by a configuration A_{curr} is done by means of matching modules of A_{curr} to unique spots in t_k, if t_k is an MCS of A_{curr} (i.e., $|V_{t_k}| < |V_{A_{curr}}|$), then some of the modules in A_{curr} will not be matched to any spot in t_k. Those unmatched modules will disconnect from A_{curr}, become singletons and will execute singleton module *spotAllocation()* algorithm, in the order of their distances from the center of T, to get allocated to a spot (lines 22–24). Note that all other modules in A_{curr} whose matched spots in t_k were free to occupy, will occupy the matched spots while retaining their configuration. The updated set of spot-selector pairs are broadcast to all modules.

Algorithm 4: Movement strategy for the modules to assume appropriate spots in T.

```
1  procedure: MoveToSpots()
   Input:
   $S_{sort}$ ← Sorted $S$ in descending order of betweenness centrality values.
   $a_c$ ← The module which is allocated to the central spot (highest betweenness centrality).
   $s_c$ ← The central spot.
   $A'$ ← Set of modules which have already assumed their allocated spots.
   $\bar{A}$ ← Set of modules which will take neighboring spots of the modules in $A'$.
2  if $a_c$ is a singleton then
3      Move to $s_c$.
4      $A' \leftarrow A' \cup a_c$.
5  else
6      //$a_c \in A_i$
7      $A_i$ moves to $T$ and therefore $a_c$ is allocated to $s_c$.
8      $A' \leftarrow A' \cup A_i$.
9  while configuration is not completely formed do
10     for each $a' \in A'$ do
11         Notify other modules of the spot that $a'$ has assumed.
12     Update $\bar{A}$.
13     Clear $A'$ (← ∅).
14     for each $\bar{a} \in \bar{A}$ do
15         Move to $T$ and assume allocated spots.
16     Update $A'$.
```

4.2 Acting phase

After the planning phase is finished and all the spots in the target configuration have been selected by modules, the modules have to move to their respective selected spots. Note that no module moves until all the spots are selected. If there is no proper order of modules for assuming spots, then a dead-

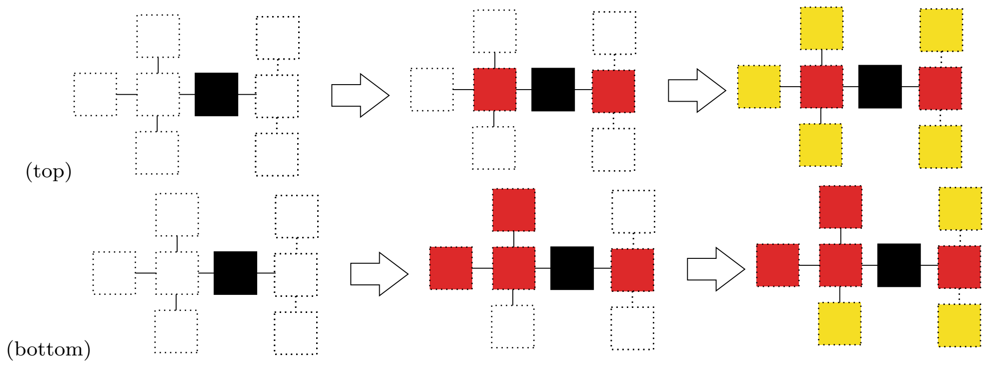


Fig. 5 Illustration of acting phase: dotted boxes represent the spots in T. First, the spot with the maximum betweenness centrality gets allocated (black spot). Next its neighbors get allocated (red spots) and finally neighbors of red spots get allocated (yellow boxes). (top) all modules are singletons; (bottom) red modules on the left side were part of an initial configuration. Therefore they occupied the spots at the same time even though two of the extreme red modules were not immediate neighbors of the central black module (Color figure online)

lock situation might arise. For example, in Fig. 1b, if all the modules occupy their spots before module 5 does, assuming module 5 is a singleton, then it will be difficult for module 5 to occupy its spot properly, unless other modules give it space for moving. But then they will have to align themselves again, which is a difficult task. To avoid this, the module which has selected the spot with highest betweenness centrality value (or, central spot), will move first and assume its position. Once it is in its proper position, it will broadcast a message to notify this to all other modules. Next the spots neighboring the center spot will be filled and so on. The procedure is shown in Algorithm 4.

If some spot s_i is allocated to a module which is part of an initial configuration A_j, then the whole configuration moves together to assume the allocated spots. As the initial configuration is a connected graph, therefore s_i's neighbors and their neighbors will get filled up by this. Next, the spots adjacent to A_j's allocated set of spots and the empty spots which are closer to s_i which did not get assumed because of A_j's allocation will be assumed (Fig. 5). Similar inside-out growth approaches have been proved to be very effective in mitigating the challenges like hole covering, deadlock avoidance etc. in swarm robotic self-assembly (Rubenstein et al. 2014; Werfel and Nagpal 2008; Dutta et al. 2012) and also in our earlier work of configuration formation in modular robots from singleton modules (Dutta and Dasgupta 2016). Techniques described in Dutta et al. (2018) can then be used for locomotion of the modules.

4.3 Theoretical analysis of algorithms

In this section, we provide the theoretical analysis of our proposed algorithms for singleton and initial configuration allocation.

Theorem 1 spotAllocation *and* blockAllocation *algorithms are complete when sufficient numbers of modules are available to form desired target configurations.*

Proof We prove the completeness of the algorithms by showing that there is no empty spot or hole in the target configuration when the number of modules is at least equal to the number of spots in the target configuration T, i.e., when $|\mathbb{A}| \geq |S|$. A hole exists in T if there is a spot s_h that is not occupied by any module. This can happen because of two conditions: 1) No module has selected s_h, or, 2) module a_h, which selected s_h, could not reach its spot because another module blocked the path to its selected spot by occupying a spot that was further from the center of T than the selected spot. We show that these two conditions cannot arise. If $|\mathbb{A}| \geq |S|$, then because of the recursive approach in the *evict* method of Algorithm 1, each module will try to select a spot in T, as long as there are available spots. This guarantees that condition 1 never arises as at least one module a_h will select s_h. Condition 2 will never arise because, as described in Sect. 4.2, modules' priority to move is based on the betweenness centrality of their selected spots, and spots nearer to the center of T are occupied first, followed by outer ones. In other words, no module will occupy an outer spot before its neighboring spot, that is nearer to the center of the target configuration gets occupied. Consequently, T cannot have a hole. Hence proved. □

Lemma 1 *Any module a_i allocated to any spot s_j before the* evict() *method will still be allocated to some spot s_k after the execution of the* evict() *method even if $s_k \neq s_j$.*

Proof We prove this by contradiction. Let us assume that as a result of the *evict()* method, a_i will be allocated to a null spot, s_k, i.e., $s_k = NULL$. But according to our proposed eviction strategy, if a_i does not get a spot to be allocated to, then the module which is trying to evict it will not be able to do that and as a result, a_i will still be allocated to its spot $s_j \neq NULL$. Therefore, *evict()* method will not reduce the number of modules that are already allocated to some spots. □

Corollary 1 *The number of modules allocated to unique spots in the target configuration increases monotonically over time.*

Lemma 2 *Eviction of module is eligible iff the total utility earned by the modules increases.*

Proof We prove this by contradiction. Let's assume that module a_i evicts another module a_j from spot s_k and then a_j selects its next highest utility spot s_l. If U^* and U' denotes the total utility earned by all the modules with and without this eviction and $s_{k'}$ denotes a_i's next highest utility spot, then we assume $U^* < U'$. For the sake of simplicity, let's assume that s_l was not selected by any other module before and no other modules are executing the *evict()* function. So, $U^* = U_{a_i}(s_k) + U_{a_j}(s_l) + U^{rest}$ and $U' = U_{a_i}(s_{k'}) + U_{a_j}(s_k) + U^{rest}$. From Algorithm 2, we can guarantee that eviction is possible *iff* $U_{a_i}(s_k)+U_{a_j}(s_l) > U_{a_i}(s_{k'})+U_{a_j}(s_k)$ and therefore $U^* > U'$. Hence our initial assumption was incorrect and it's proved that the *evict()* function maximizes the total utility. □

Theorem 2 (spotAllocation) *algorithm returns a Pareto-optimal allocation between modules and spots, i.e., any module's earned utility cannot be improved without making another module's utility worse.*

Proof Let $s_{i,k}$ denote the *k*th highest utility spot for module a_i. Because each module orders the spots based on utilities, it follows that $U_{a_i}(s_{i,k}) > U_{a_i}(s_{i,k+1})$. Consider two modules a_i and a_j that have the highest utility for the same spot s (i.e., $s_{i,1} = s_{j,1} = s'$, but $U_{a_i}(s') > U_{a_j}(s')$. Also, assume that a_j has selected spot s' first. Now, if the *spotAllocation* allocates a_i to its next best spot, $s_{i,2}$ and a_j remains at s', then the total utility is $U^1 = U_{a_i}(s_{i,2}) + U_{a_j}(s')$. On the other hand, if *spotAllocation* method evicts a_j from s' and allocates it to its next best spot $s_{j,2}$ (assuming it is free), then the total utility becomes $U^2 = U_{a_i}(s') + U_{a_j}(s_{j,2})$. From Algorithm 1, if eviction is possible, then $U^2 > U^1$. On the other hand, if eviction does not happen, then it implies $U^1 > U^2$. For any other allocation strategy that does not do eviction even if $U^2 > U^1$, then the total utility earned by the alternate allocation strategy is always less than the utility earned by the *spotAllocation* algorithm. From the above equations, we can conclude that, if any two modules a_i and a_j have same ranking for a particular spot, s', then one of the modules will be allocated to that spot and the other will be pushed to its next highest utility spot, i.e., its earned utility reduces, and no other allocation would increase their utilities as well as the overall utility. Hence the allocation strategy is Pareto-optimal. □

Lemma 3 *Both* spotAllocation *and* blockAllocation *algorithms are deterministic in nature, i.e., no two modules will be allocated to the same spot as a result of our proposed strategy.*

Proof We divide the proof into two following scenarios.

Case I

Let us assume that a singleton a_i selects a spot s_j which is already allocated to another singleton module a_k and also a_k's allocation does not change due to this, i.e., both a_i and a_k are now allocated to s_j. But according to Algorithm 1, a_i will first try to evict a_k from s_j and then it can be allocated. If a_k cannot be evicted, then a_i will not select s_j. Also, following Lemma 1, we can guarantee that if a_k is evicted, then it will be allocated to some spot $s_l \neq S_j$. On the other hand, if a_k is a member of an initial configuration, then a_i cannot evict it anyway; rather it will look for the next best available spot. Therefore it is not possible that both a_i and a_k will be allocated to the same spot s_j.

Case II

If $a_i \in A_m$, and a_k is a singleton module, then a_i has permission to evict a_k if all other required conditions are satisfied. Following the similar logic as before, we can guarantee that if a_k is evicted by $a_i \in A_m$, then a_k will be allocated to some spot $s_l \neq S_j$. If a_k cannot be evicted, then a_i will look for a different spot (different isomorphic subgraph or as a singleton module if detached). A similar thing will happen if $a_k \in A_l$. Therefore, we can guarantee that if a_i is part of an initial configuration A_m, it will not be allocated to the same spot s_j with a_k.

Hence proved. □

Theorem 3 *As $\mathcal{D}_{max}$ approaches $|S|$, the total utility earned by the modules (U) approaches the optimal utility U^*.*

Proof If there is no conflict among the modules about their best spots, i.e., each module's highest utility spot is unique, then the *spotAllocation* algorithm allocates highest utility spots to all the modules and thus achieves the optimal utility. But if there is a conflict among modules for the same spots, then the eviction method is invoked. From Algorithm 1, we can conclude that the total utility earned by the modules increases by successively calling the evict method. For $\lim_{\mathcal{D}_{max} \to |S|}$, any subsequent evictions will consequently increase the total utility. If eviction fails, then that means the total utility cannot be improved any further. Thus, every time the eviction method is invoked it will increase the total utility, going towards the optimal utility. □

Theorem 4 *The proposed configuration formation process converges with time.*

Proof Following Theorem 1, Lemma 1, and Corollary 1, we can guarantee that the configuration formation process will converge over time. □

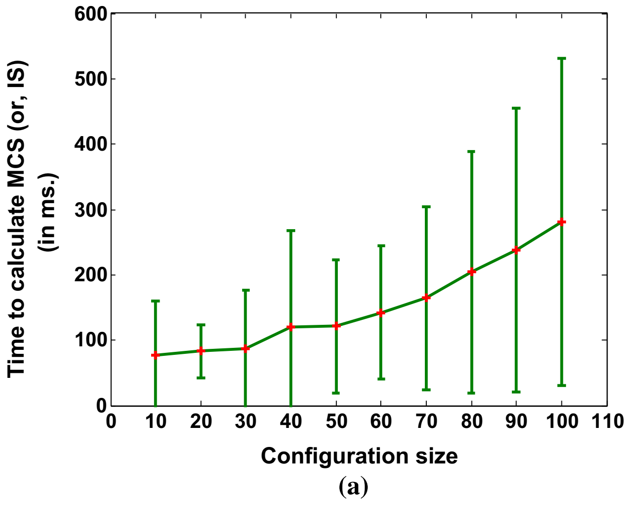
Note on complexity: The *spotAllocation* algorithm (Algorithm 1) has a time complexity given by $O(|S|^{\mathcal{D}_{max}})$ where $|S|$ is the number of spots in the target configuration and $\mathcal{D}_{max}$ is the depth up to which the eviction of modules is allowed. In the *blockAllocation* algorithm (Algorithm 3), target configurations are considered to be trees and finding all possible isomorphic subtrees in the target configuration has a polynomial worst case time complexity of $O((|S||A_i|)^{d+1})$ (Cordella et al. 2004), where $|A_i|$ and $|S|$ are the number of modules and spots in intial and target configurations respectively, and d is the maximum branch factor of either configuration.

5 Experimental evaluation

5.1 Settings

We have implemented the spot allocation algorithm on a desktop PC (Intel Core i5-960 3.20GHz, 6GB DDR3 SDRAM). We tested instances where random numbers of singletons and initial configurations with sizes between 2 and 10 modules need to be allocated to target configurations with between 10 and 100 spots. In all cases, unless otherwise mentioned, the total number of modules in the environment is equal to the total number of spots in the target configuration. Each module is modeled as a cube of size 1 unit × 1 unit × 1 unit. The modules are placed at random locations within a 16 unit × 16 unit environment, their initial orientations are drawn from a uniform distribution in $\mathbb{U}[0, \pi]$, and the initial positions of singletons and leaders of the initial configurations are drawn uniformly from $\mathbb{U}[(0, 15), (0, 15)]$. For all the tests, $\mathcal{D}_{max}$ has been set to 3. Changing the value of $\mathcal{D}_{max}$ from 3 to 10 affected the algorithm's performance (both time and quality wise) negligibly; therefore this is not included in the results.

Extracting 'better' isomorphic subgraphs Initial and target configurations were restricted to be trees based on the connections the modules in our MSR platform are capable of, although our algorithms can be applied for any other kinds of graphs as well. As there can be numerous subtrees present in the target configuration, which are isomorphic to the initial configuration, and finding all possible isomorphic subtrees can take considerable time, we set an upper bound, MAX, on the number of isomorphic subtrees that the *blockAllocation* algorithm (Algorithm 3) will check. MAX is set to 20; different values of $MAX = 10, 30$, or 40 did not change the performance of the algorithm. To get higher utility isomorphic subtrees, first the nodes in the target configuration are sorted in descending order of betweenness centrality values, because if the costs to occupy two different spots are the same, then higher betweenness centrality (spot value) indicates higher utility of the spot. For every node in the sorted list of spots, every node in current configuration A_i is made the root of A_i once and checked for subtree isomorphism with target configuration T while making each node in T the root once, for every possible tree in A_i. The checking of isomorphic subtrees between A_i and T is stopped as soon as the first MAX isomorphic subtrees are found. All results are averaged over 50 runs.



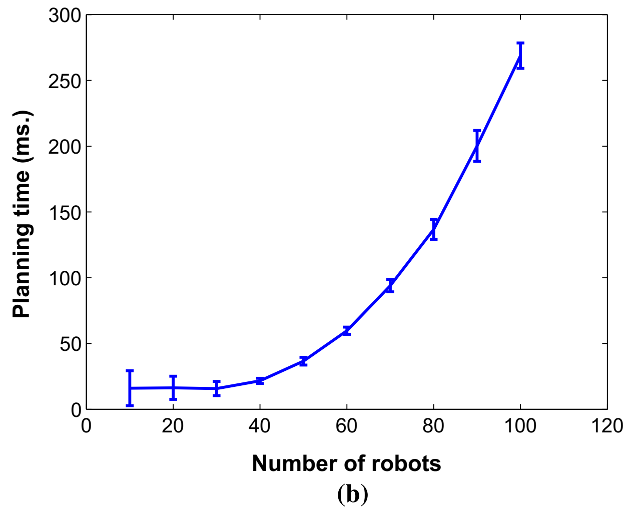
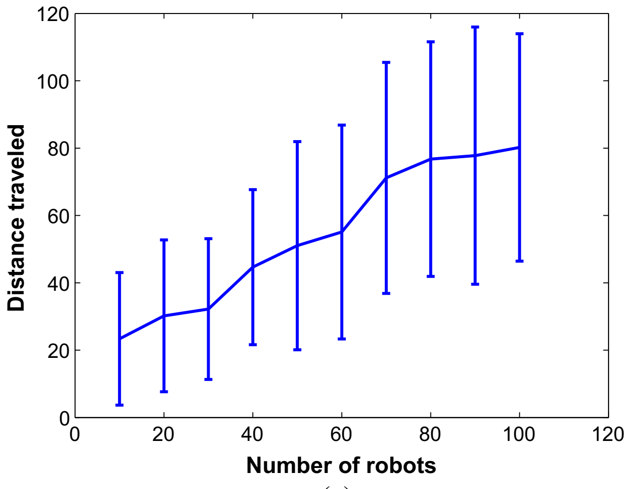


Fig. 6 **a** Time to calculate MCS or IS versus. different initial configuration sizes, **b** Total planning time for different number of modules in environment

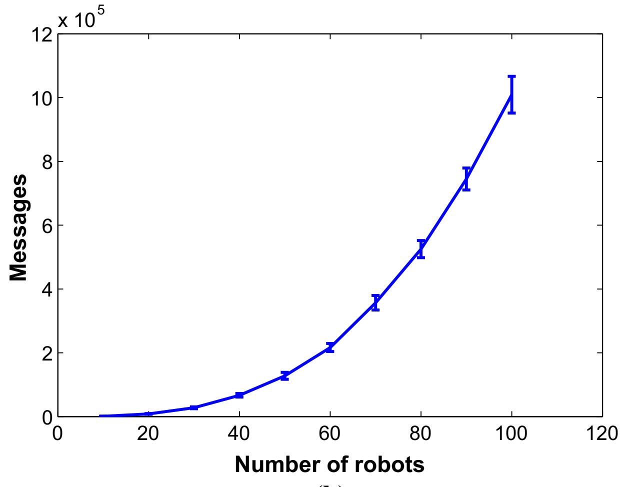
5.2 Results

Performance analysis of our approach First we have shown how much time it takes to find MAX number of MCS (or, IS). The result is shown in Fig. 6a. The x-axis denotes the size of a single configuration and the y-axis denotes the time in milliseconds to find MAX number of MCS (or, IS) of that configuration in the target configuration. For this test, total spots in the target configuration have been set to 100. Though the run time increases with the size of the initial configuration, which can be expected because of the complexity results shown in Shamir and Tsur (1997) for finding isomorphic subtrees, still it was always well within a reasonable bound. In the next set of experiments, we have focused on the main contribution of this paper—how to construct a modular robotic system from an initial set of singletons and configurations. Figure 6b shows how the planning time changes with different numbers of modules; the y-axis denotes the total planning time in milliseconds and the x-axis denotes the number of modules. It can be noted from this plot that though for a small set of modules, time change is almost constant, as the configuration size as well as the number of modules increases, elapsed time increases in a polynomial fashion. This elapsed time indicates only the planning phase execution time of the modules. Figure 7a shows how with increasing number of modules the total distance traveled by them changes. This metric is calculated by adding the distances traveled by each module from their initial positions to their respective spots in T. The figure shows that the total distance traveled by the modules increases linearly. We have also calculated the total number of messages passed among modules while the configuration formation process is occurring. Figure 7b shows how the number of total messages changes with the number of modules. As can be expected, with a higher number of modules in the environment, the number of messages increases in a polynomial fashion (cubic on the number of robots). Note that, this curve shows the total message passed among the robots, i.e., a total of messages sent and received. If only sent messages are considered, then this curve becomes quadratic in nature.

We are also interested in understanding the completion rate of the planning phase. The percentage of planning phase completion indicates what percentage of the total modules are allocated to their spots in T. Figure 8a shows the planning completion rate for different numbers of modules between 10 and 100. We can see that with increasing numbers of modules, the completion rate increases and is more evenly distributed over time. For instance, with $|\mathbb{A}| = 10$, after 70% time completion, only 30% of planning has been completed, whereas with $|\mathbb{A}| = 100$, 30% of planning gets completed after only 25% of time completion. The relationship between planning phase completion and the number of passed messages for 100 modules has been shown in Fig. 8b. All the graphs from 50 runs have been plotted. We observe that the message count is increasing almost-linearly with completion rate. For the next set of experiments, we have kept the number of spots, $|S|$, fixed at 50 and we have varied the number of modules between [50, 100]. Figure 9a shows planning completion rate for different numbers of modules. We can see that with increasing number of modules, completion rate increases and is more evenly distributed over time. This behavior is similar to what we have seen in Fig. 8a. Although



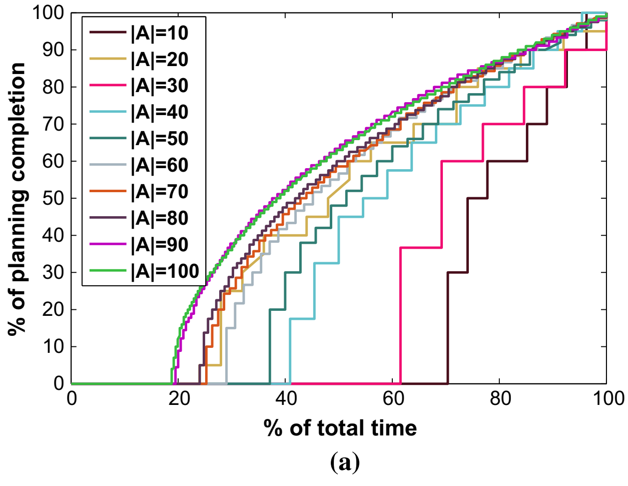
(a)



(b)

Fig. 7 **a** Distance traveled by modules to reach target configuration for different number of modules in the environment, **b** Number of messages exchanged between modules to select positions in the target configuration for different numbers of modules in the environment

in Fig. 8a, for most of the module sets, the planning phase completes almost at the end of their respective time-lines, in the case of Fig. 9a, we can notice that the planning phase finishes at different stages of their time-lines, for different numbers of modules. As an example, for $|\mathbb{A}| = 100$, the planning phase almost converges at 50% the of total elapsed time, whereas for $|\mathbb{A}| = 50$, it takes almost 100% time to converge. Figure 9b shows the comparison of the number of passed messages by the different numbers of modules, between the cases where $|S| = 50$ and $|S| = |\mathbb{A}|$. It can be observed from this figure that with same number of modules, fewer messages are passed if there are fewer spots than modules, i.e., if $|S| < |\mathbb{A}|$. For example, with, $|\mathbb{A}| = 100$ and $|S| = 50$, 8×10^5 messages are passed, whereas with $|S| = 100$ and keeping $|\mathbb{A}|$ fixed to 100, the number of messages increases to 10×10^5. This result shows that the total



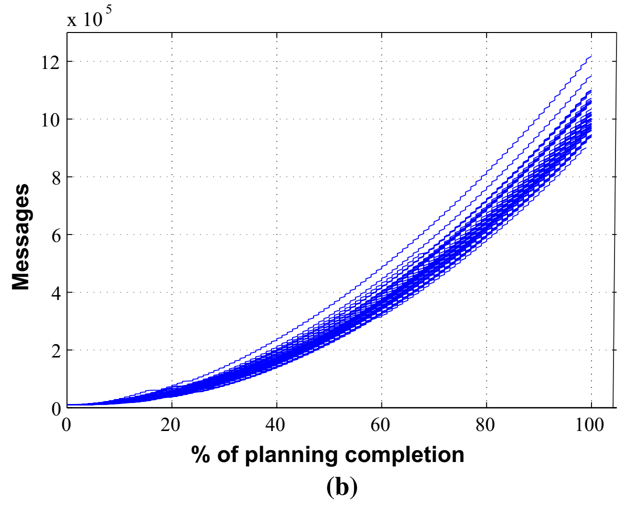
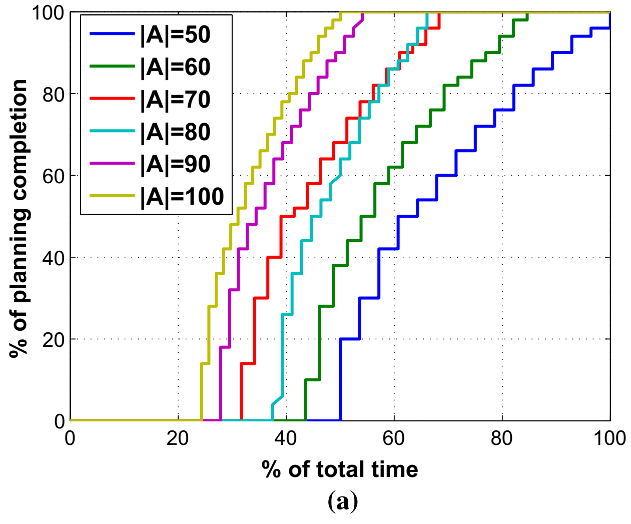


Fig. 8 **a** Change in % of planning completion with % of time completion, for different no. of modules, **b** Change in no. of messages at different time steps, for 100 modules



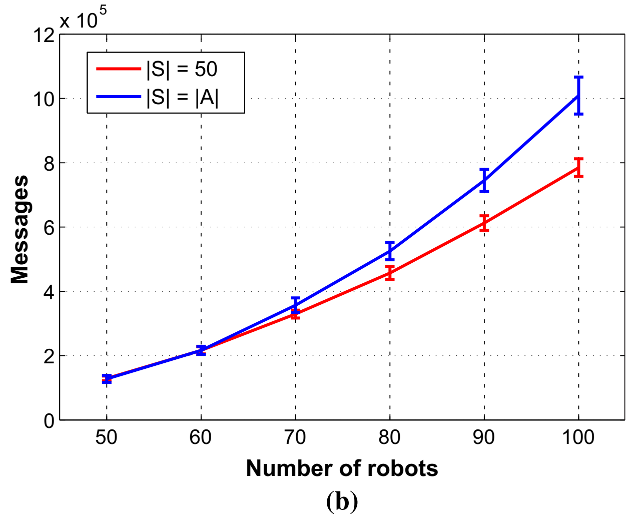


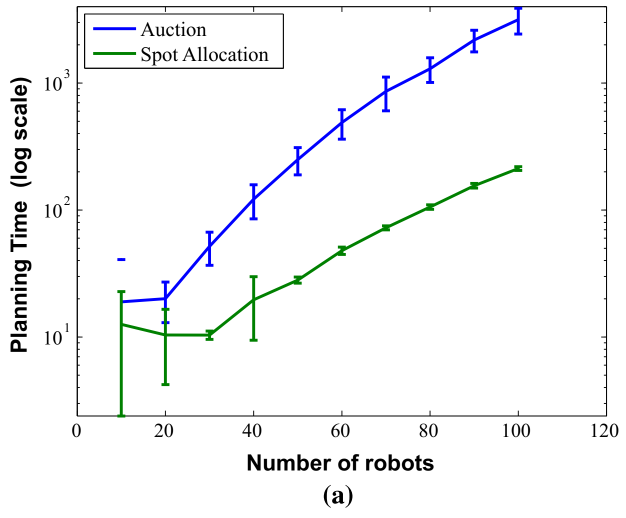
Fig. 9 **a** Change in % of planning completion with % of time completion, for different no. of modules and $|S| = 50$, **b** Change in no. of messages for different no. of modules and different no. of spots

number of messages depends on both the number of modules and spots. Next we have run experiments to check how the subgraph isomorphism technique used in this work helps to reduce the number of disconnections from initial configurations. For this test, we have kept $|S| = |\mathbb{A}| = 100$. Initially all modules were part of some smaller configurations and each initial configuration has the same size. We have varied the sizes of each initial configuration between [10, 20, 25, 50] and thus in these cases the number of initial configurations have been varied between [10, 5, 4, 2]. The planning times and number of modules required to be disconnected for these cases are shown in Table 1. As can be seen, with increasing size of initial configurations, the number of disconnected modules increases. This is because the probability of finding isomorphic subgraphs in T decreases with increasing size of initial configurations. But the low numbers of disconnected modules show that it is always beneficial, in terms of number of connections detachments and re-attachments, to use our proposed approach than to break all initial configurations into singletons and then form the target configurations with them.

Comparison with auction-based allocation We have also compared our approach for MSR configuration formation with an auction algorithm (Bertsekas 1990) that finds an optimal assignment between spots and modules. Using the auction mechanism a group of modules bid for a set of spots. First the modules bid for their most preferred spots; conflict among modules for the same spot is resolved by revising bids in successive iterations. The assignment is done in a way such that the utility is maximized. The auction algorithm does not take connected configurations of modules during allocation. Therefore only for the tests which compare the performances of our algorithm against the auction algorithm, initially all the modules are considered to be singletons.

Table 1 Planning times and the numbers of disconnected modules (average and standard deviation) in the configuration formation process, where all initial configurations have same sizes ($|S| = |\mathbb{A}| = 100$)

Size of all initial configurations	Planning time (ms)	No. of modules disconnected
10	171.48 (avg.)	0.12 (avg.)
	15.13 (std.)	0.32 (std.)
20	166.66 (avg.)	4.32 (avg.)
	12.88 (std.)	3.56 (std.)
25	172.10 (avg.)	8.76 (avg.)
	11.30 (std.)	4.85 (std.)
50	218.28 (avg.)	29.68 (avg.)
	19.57 (std.)	5.33 (std.)



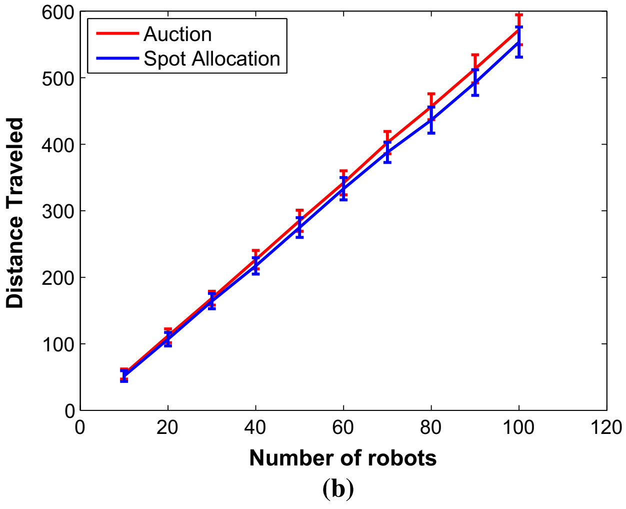
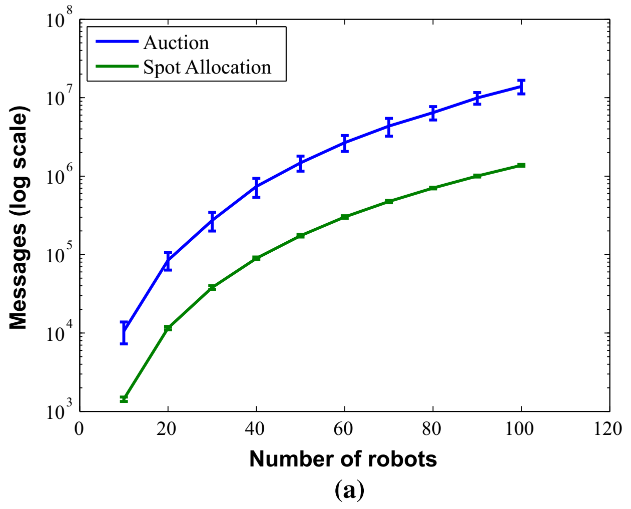


Fig. 10 **a** Log scale comparison of planning phase execution time with auction algorithm, **b** Comparison of total traveled distances with auction algorithm



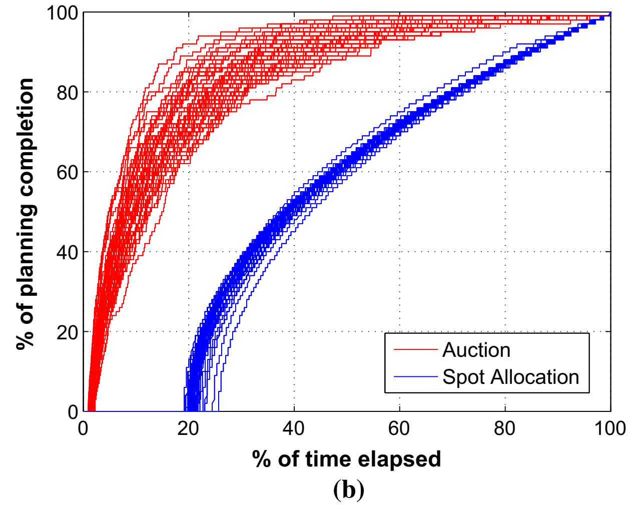


Fig. 11 **a** Log scale comparison of no. of messages with auction algorithm, **b** Change in % of planning completion with % of time completion and comparison with auction algorithm. 50 lines indicate 50 runs

A log scale comparison of planning times between spot allocation and the auction algorithms is shown in Fig. 10a. As can be seen from this graph, with increasing the number of modules, the difference between planning times of these two algorithms increases, i.e., our proposed algorithm's performance gets better with increased number of modules compared to the auction algorithm. Comparison of distances traveled by the modules using our algorithm and the auction algorithm is shown in Fig. 10b. As we can see in this plot, in most of the cases total traveled distance by the modules is the same. But with higher numbers of modules, using the proposed spot allocation algorithm modules travel less distance than by using the auction algorithm. Thus the spot allocation algorithm assigns the spots to the modules in very nominal time, keeping the cost for movement almost the same (or less in some cases), compared to the auction algorithm. A log scale comparison of number of messages generated, by the spot allocation and auction algorithms, is shown in Fig. 11a. This figure indicates that the spot allocation algorithm generates fewer messages than the auction algorithm, which helps to reduce the communication overhead. Figure 11b compares the completion rates of planning phases of the auction and spot allocation algorithms—the x-axis denotes the percentage of total time elapsed. This result indicates that completion rate of the auction algorithm is higher, even though the auction algorithm takes longer than the spot allocation algorithm.

5.3 Case studies

In this section, we have shown 8 specific cases of the configuration formation process that are shown in Fig. 12. Each of the initial and target configurations used for this set of experiments have been shown to be feasible and stable for the ModRED MSR in Hossain et al. (2014). To show the generalization of our approach, we have used both tree and graph structured MSR configurations as opposed to only tree configurations used in the previous sections. This was also made possible due to not-so-large configurations used here. Squares represent the modules and the links between two squares denote the connection between those two modules.

For each case illustrated, the left-most diagram shows the initial configurations and/or singletons, the middle diagram shows the detected MCS (or, IS) and the diagram on the right shows the final formed configuration. The modules are color-coded to show the final allocations. MCS (or, IS) are shown with dotted boxes. Grey-colored modules represent the modules that remain connected to the same neighboring module between initial and target configurations, but only change the connector through which they are connected. Although this operation requires one un-docking and one re-docking operation, it consumes less energy than if the module were to be connected to a non-neighbor module. The planning time and number of disconnections for each case are provided alongside each configuration formation case in Fig. 12. We can see that each of the test cases requires less than 200 milliseconds of planning time. Target configurations are also formed with relatively few link disconnections (maximum being 4) (Fig. 12).

5.4 Hardware experiments with ModRED II MSR

The main objective of hardware experiments is to show how much time it takes for the singleton modules and the leader modules to do the local computations. We have chosen the ModRED II modular self-reconfigurable robot platform (Hossain et al. 2014) for experimental purposes. Each ModRED II module is a 4-DOF robot (similar to its predecessor ModRED I (Baca et al. 2014) with four connectors (unlike its predecessor which has only two connectors in both ends). Due to its four in-built connectors, ModRED II (Fig. 13) is able to form more complex configurations compared to ModRED I. For more details on ModRED II hardware architecture and features, readers are referred to Hossain et al. (2014). Each ModRED II module also houses a BeagleBone Black, a Linux based computer, on-board. It has 512MB DDR3 RAM and 4GB 8-bit eMMC on-board flash storage. It is also equipped with a AM335x 1GHz ARM® Cortex-A8 processor.

As we have mentioned earlier that our main objective is to show how much on-board computation is needed by the singletons and the leader modules, we have used a single ModRED II module for our experiments which alternatively worked as a singleton and a leader module. For these experiments, we have implemented our algorithms on the Beaglebone Black Processor inside the ModRED II module and collected the results. We have also compared our algorithms' performance against the auction algorithm's performance by implementing the auction algorithm on the ModRED II as well. As we ran hardware tests on a single module, the reported run time results only consider the computational time, and do not include the communication time between modules.

First, the ModRED module acted as a leader. In our earlier work (Baca et al. 2016), we have shown how much time it takes to elect a leader and to map the topology of the configuration for varying sizes of the configuration. This result is reproduced here in Fig. 14a to show how much preprocessing will be needed before we can start executing our proposed algorithms in this paper. This shows that with 7 modules present in the configuration, it takes less than a second of time to elect the leader and map the topology of the configuration.

Next, the elected leader module searches for MAX MCS (or IS) for the given target configuration. For this test, we have provided the topology of the initial configuration and also the target configuration to the leader module. Similar to the simulation results, these configuration trees have been generated randomly. Figure 14b shows how with the increasing size of the configuration, run time to search the MCS changes. Note that the size of the target configuration was set to 100 for all the cases here. We can observe that the run time increases almost linearly in fashion even though it took longer time than simulated experiments. We noticed that running the *blockAllocation()* algorithm along with this took a negligible amount of extra time, so that result is not included in the paper. The main reason behind this is that calculating the possible MCSs is the most computationally intensive component in our proposed *blockAllocation()* algorithm.

Next, we have implemented the *spotAllocation()* algorithm on the singleton ModRED module. The result is shown in Fig. 15a. This result shows that a singleton module will take much less time (0.019 s) even with 100 spots. Finally, we implemented the auction algorithm on the ModRED module

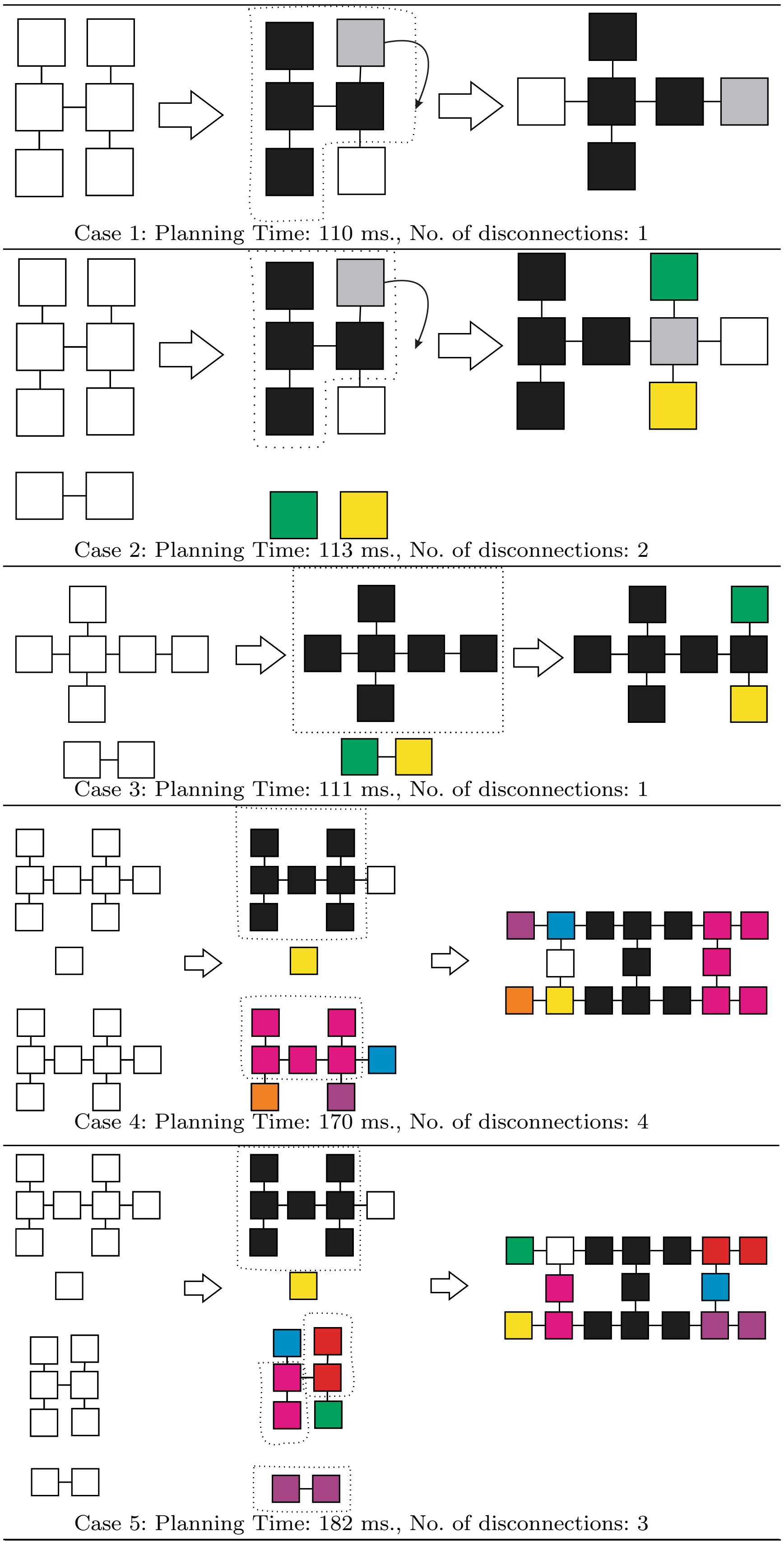
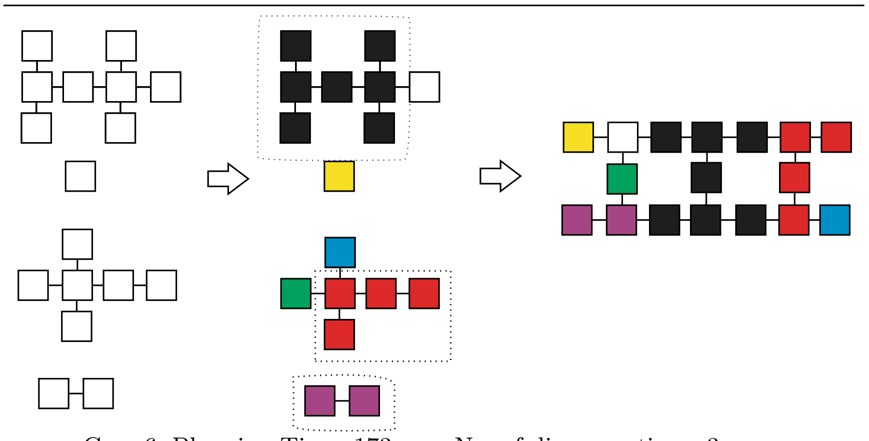
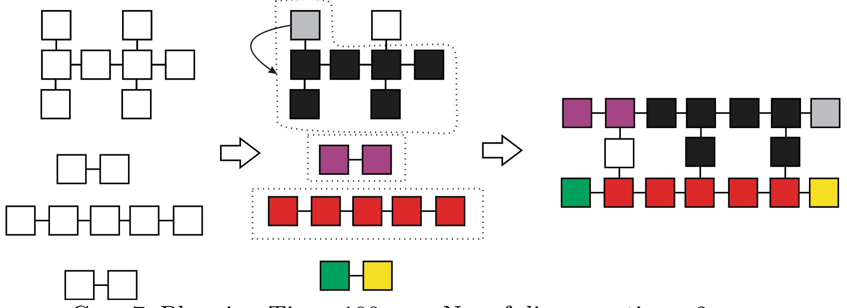

Case 1: Planning Time: 110 ms., No. of disconnections: 1
Case 2: Planning Time: 113 ms., No. of disconnections: 2
Case 3: Planning Time: 111 ms., No. of disconnections: 1
Case 4: Planning Time: 170 ms., No. of disconnections: 4
Case 5: Planning Time: 182 ms., No. of disconnections: 3

Fig. 12 Cases showing configuration formation procedure along with corresponding planning times and number of disconnections required. Leftmost figure in each case shows the initial configurations and singletons, middle figure shows the MCS (or, IS) found (marked by dotted boxes) by executing our algorithms, rightmost figure shows the final formed target configuration with modules selecting spots (shown in a color-coded fashion) (Color figure online)

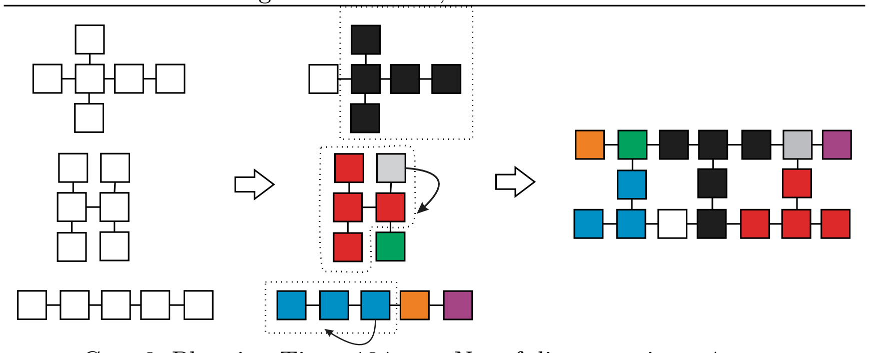
Fig. 12 continued



Case 6: Planning Time: 173 ms., No. of disconnections: 3



Case 7: Planning Time: 190 ms., No. of disconnections: 2



Case 8: Planning Time: 184 ms., No. of disconnections: 4

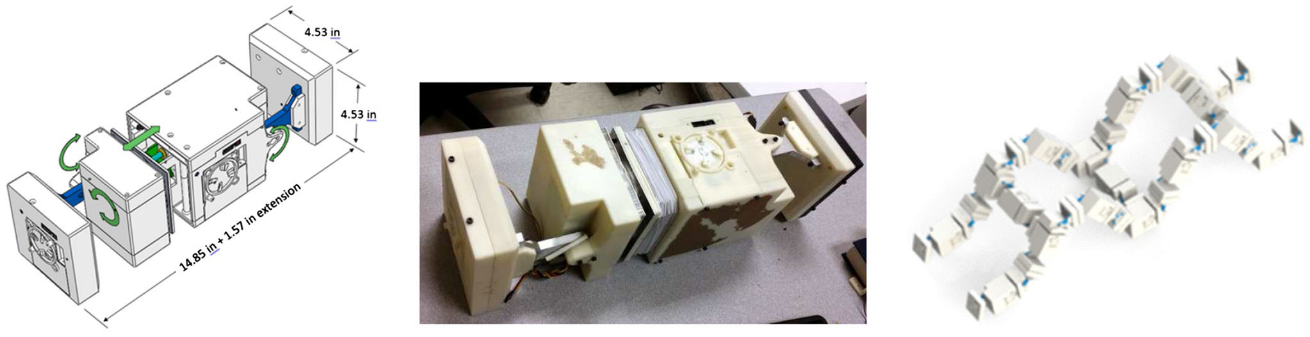
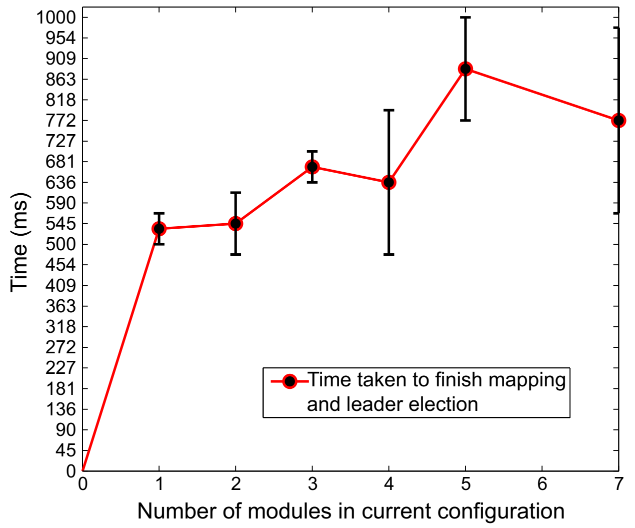
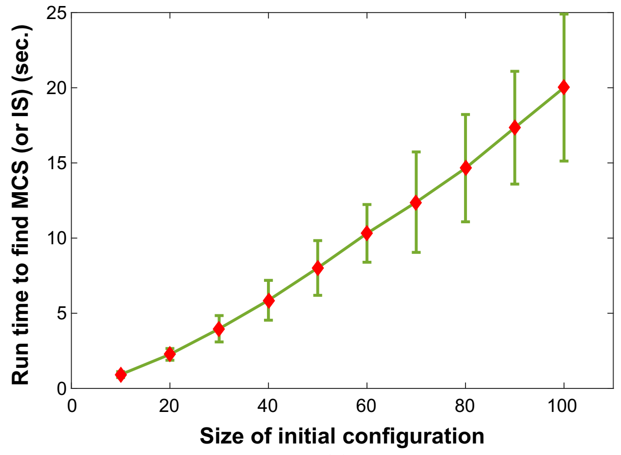


Fig. 13 A single ModRED module used for the configuration formation algorithm **a** CAD drawing, **b** hardware. Each module has 4 connectors which enables it to form branched configurations. **c** A 17-module branched, ladder configuration similar to Fig. 1b that is capable of complex maneuvers and forming truss-like structures (Baca et al. 2014)

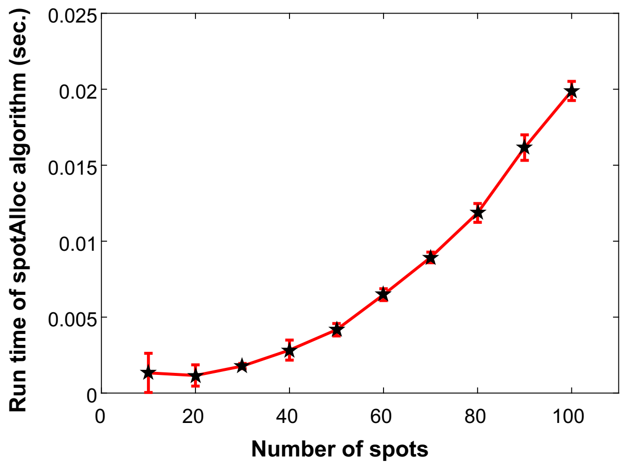


(a)

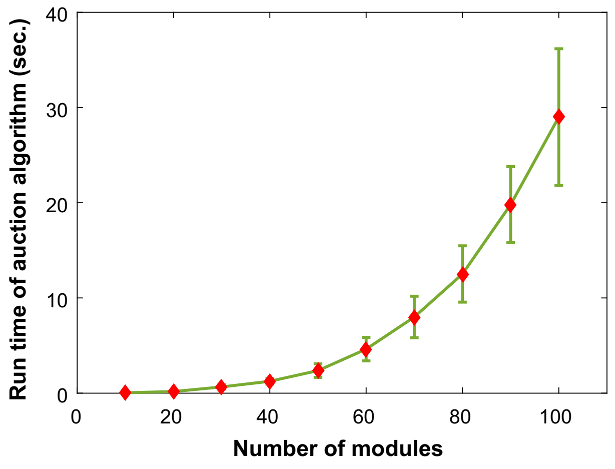


(b)

Fig. 14 **a** Comparison of run times to elect a leader and map the topology of the ModRED configuration against the configuration size, **b** Change in run time to find MCS with different number of modules in the initial configuration



(a)



(b)

Fig. 15 **a** Change in run time of the spotAllocation() algorithm with different number of spots, **b** Change in run time of the auction algorithm with different number of spots

and in this case, the ModRED module acts as a centralized auctioneer agent. The number of spots is set equal to the number of modules in this case. The result of this test is shown in Fig. 15b. We can notice that with increasing numbers of modules, the run time of the auction algorithm increases significantly. For example, with 100 spots, the ModRED module takes 29 s to run the auction algorithm whereas it takes only 0.019 s to run the *spotAllocation()* algorithm, a 1526-times improvement.

6 Discussions

Our main objective in this paper was to find an efficient solution for the configuration formation problem where initially, modules could be either singletons or part of an already connected configuration. We have argued that as docking and un-docking of modules are costly operations, these operations should be minimized by preserving the initially formed configurations as much as possible. In this paper, we have proposed subgraph isomorphism based checking and allocation algorithms that retain maximal portions of connected modules while forming a target configuration. From our results, we can notice that our solution produces good results consistently, both in terms of planning time, distance traveled and number of connections/disconnections among modules, given the combinatorially intractable nature of the used techniques. Although most of the results reported in this paper are produced using tree-like MSR structures, our case studies show that even with graph structures, our methods are able to produce considerably good results especially in terms of number of disconnections among the initially formed configurations.

As allocating modules to target spots is an instance of the classical bipartite graph matching problem, algorithms like Hungarian matching can also be used for the allocation process (at least for singleton modules) (Kuhn 1955b). As our approach is distributed in nature, a relevant issue is the scalability of the number of messages passed between modules for synchronizing intermediate calculations of the algorithms. As the modules need to reach consensus about allocation in a distributed manner, they need to continuously exchange information about the current state of the allocation process with each other. A semi-centralized method, where part of the decision is made by a central supervisor, can be used to mitigate this problem (Dutta and Dasgupta 2016). However, this increases the risk of potential failure of the whole process if the supervisor fails. Finally, we have tested our approach with homogeneous modules only, but it remains an open research problem for future researchers to investigate the configuration formation problem with heterogeneous modules where initially modules can be part of different configurations instead of just singletons. Besides modular robotics, we believe that our proposed approach can be used for parts assembling in the manufacturing and automobile industries where smaller portions (initial configurations) of objects can be brought together and assembled to form a large object (target configuration).

7 Conclusions and future work

In this paper, we have proposed novel spot allocation algorithms for configuration formation in MSRs. To the best of our knowledge, our approach is the first one to handle the problem where modules might not be just singletons in the beginning, but they can be in any arbitrary configuration. From there, modules need to find an allocation to the target configuration such that the initial configurations can be directly allocated to the target configuration as much as possible while reducing the number of modules that will have to undock and re-dock. Our proposed approach is distributed in nature. Modules use messages to get informed about the global state of the allocation procedure. Our results show that our proposed approach takes very nominal time for calculating the final allocation. Also, using our allocation, the total distance traveled by the modules from their start to target locations increases linearly with the number of modules. Moreover, our proposed approach outperforms the auction algorithm in run time while maintaining similar solution quality (in terms of distance traveled). Our proposed approach is also shown to perform well in terms of preserving initial configurations—using our approach, very small numbers of disconnections are usually needed among the initially connected modules.

Our proposed approach in this paper is the first step towards solving this difficult yet highly important problem in modular robotics. Currently, our proposed approach does not model the uncertainty in the environment. In the future, we are planning to add uncertainty in modules' movements as well as in their message passing. Right now, we assume that there is only one target configuration that needs to be formed. But there can be cases where multiple target configurations need to be formed at the same time. In the future, we also plan to consider this scenario, which will make the decision-making problem more complex for the modules. The acting phase in our approach is sequential in nature—it allocates one module (or configuration) at a time. Even though it guarantees no deadlock, it is slower than if modules move simultaneously. However with simultaneous locomotion of modules, it will be more difficult to guarantee properties like deadlock-free. We are currently working towards solving this particular problem. This will make the system faster and more robust in nature, giving new possibilities of different applications.

References

Ahmadzadeh, H., & Masehian, E. (2015). Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization. *Artificial Intelligence*, *223*, 27–64.

Aho, A. V., & Hopcroft, J. E. (1974). *The design and analysis of computer algorithms*. Bengaluru: Pearson Education India.

Akutsu, T. (1993). A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *76*(9), 1488–1493.

Alonso-Mora, J., Breitenmoser, A., Rufli, M., Siegwart, R., & Beardsley, P. (2011). Multi-robot system for artistic pattern formation. In *2011 IEEE international conference on robotics and automation (ICRA)* (pp. 4512–4517). IEEE.

Asadpour, M., Sproewitz, A., Billard, A., Dillenbourg, P., & Ijspeert, A.J. (2008). Graph signature for self-reconfiguration planning. In *IEEE/RSJ international conference on intelligent robots and systems, 2008. IROS 2008* (pp. 863–869). IEEE.

Baca, J., Hossain, S., Dasgupta, P., Nelson, C. A., & Dutta, A. (2014). Modred: Hardware design and reconfiguration planning for a high dexterity modular self-reconfigurable robot for extra-terrestrial exploration. *Robotics and Autonomous Systems*, *62*(7), 1002–1015.

Baca, J., Woosley, B., Dasgupta, P., Dutta, A., & Nelson, C. (2016). Coordination of modular robots by means of topology discovery and leader election: Improvement of the locomotion case. In *Distributed autonomous robotic systems* (pp. 447–458). Berlin: Springer.

Bertsekas, D. P. (1990). The auction algorithm for assignment and other network flow problems: A tutorial. *Interfaces*, *20*(4), 133–149.

Brandes, U. (2001). A faster algorithm for betweenness centrality*. *Journal of Mathematical Sociology*, *25*(2), 163–177.

Chen, I. M., & Burdick, J. W. (1998). Enumerating the non-isomorphic assembly configurations of modular robotic systems. *The International Journal of Robotics Research*, *17*(7), 702–719.

Chirikjian, G., Pamecha, A., & Ebert-Uphoff, I. (1996). Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of Field Robotics*, *13*(5), 317–338.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(10), 1367–1372.

Dasgupta, P., Ufimtsev, V., Nelson, C., & Hossain, S. (2012). Dynamic reconfiguration in modular robots using graph partitioning-based coalitions. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems-volume 1. International foundation for autonomous agents and multiagent systems* (pp. 121–128).

Davis, J. D., Sevimli, Y., Eldridge, B. R., & Chirikjian, G. S. (2016). Module design and functionally non-isomorphic configurations of the Hex-DMR II system. *Journal of Mechanisms and Robotics*, *8*(5), 051,008.

Dutta, A., Chaudhuri, S.G., Datta, S., & Mukhopadhyaya, K. (2012). Circle formation by asynchronous fat robots with limited visibility. In *Distributed Computing and Internet Technology* (pp. 83–93). Berlin: Springer.

Dutta, A., & Dasgupta, P. (2016). Simultaneous configuration formation and information collection by modular robotic systems. In *2016 IEEE international conference on robotics and automation (ICRA)* (pp. 5216–5221).

Dutta, A., Dasgupta, P., & Nelson, C. (2018). Distributed adaptive locomotion learning in modred modular self-reconfigurable robot. In *Distributed Autonomous Robotic Systems* (pp. 345–357). Cham: Springer.

Enner, F., Rollinson, D., & Choset, H. (2013). Motion estimation of snake robots in straight pipes. In *International conference on robotics and automation* (pp. 5148–5153).

Fitch, R., & Butler, Z. (2008). Million module march: Scalable locomotion for large self-reconfiguring robots. *The International Journal of Robotics Research*, *27*(3–4), 331–343.

Hossain, S., Nelson, C. A., Chu, K. D., & Dasgupta, P. (2014). Kinematics and interfacing of modred: A self-healing capable, four-dof modular self-reconfigurable robot. *JMR*, *13*, 1256.

Hou, F., & Shen, W.M. (2008). Distributed, dynamic, and autonomous reconfiguration planning for chain-type self-reconfigurable robots. In *IEEE international conference on robotics and automation, 2008. ICRA 2008* (pp. 3135–3140). IEEE.

Hou, F., & Shen, W. M. (2014). Graph-based optimal reconfiguration planning for self-reconfigurable robots. *Robotics and Autonomous Systems*, *62*(7), 1047–1059.

Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., & Murata, S. (2004). Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In *Proceedings. 2004 IEEE/RSJ international conference on intelligent robots and systems, 2004.(IROS 2004)* (Vols. 3, pp. 2370–2377). IEEE.

Klavins, E. (2007). Programmable self-assembly. *IEEE, Control Systems*, *27*(4), 43–56.

Knight, S., Rabideau, G., Chien, S., Engelhardt, B., & Sherwood, R. (2001). Casper: Space exploration through continuous planning. *IEEE Intelligent Systems*, *16*(5), 70–75.

Kobler, J., Schöning, U., & Torán, J. (2012). *The graph isomorphism problem: Its structural complexity*. Berlin: Springer.

Kuhn, H. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1–2), 83–97.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1–2), 83–97.

Kurokawa, H., Tomita, K., Kamimura, A., Kokaji, S., Hasuo, T., & Murata, S. (2008). Distributed self-reconfiguration of M-TRAN III modular robotic system. *The International Journal of Robotics Research*, *27*(3–4), 373–386.

Nelson, C.A., & Cipra, R.J. (2004). An algorithm for efficient self-reconfiguration of chain-type unit-modular robots. In *ASME DETC,* (pp. 1283–1291). New York City: American Society of Mechanical Engineers.

Neubert, J., & Lipson, H. (2016). Soldercubes: A self-soldering self-reconfiguring modular robot system. *Autonomous Robots*, *40*(1), 139–158.

Pamecha, A., Ebert-Uphoff, I., & Chirikjian, G. S. (1997). Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, *13*(4), 531–545.

Park, M., Chitta, S., Teichman, A., & Yim, M. (2008). Automatic configuration recognition methods in modular robots. *The International Journal of Robotics Research*, *27*(3–4), 403–421.

Raymond, J. W., & Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, *16*(7), 521–533.

Reyner, S. W. (1977). An analysis of a good algorithm for the subtree problem. *SIAM Journal on Computing*, *6*(4), 730–732.

Rosa, M., Goldstein, S., Lee, P., Campbell, J., & Pillai, P. (2006). Scalable shape sculpturing via hole motions. In *IEEE international conference on robotics and automation*, (pp. 1462–1468). Orlando, FL.

Rubenstein, M., Cornejo, A., & Nagpal, R. (2014). Programmable self-assembly in a thousand-robot swarm. *Science*, *345*(6198), 795–799.

Shamir, R., & Tsur, D. (1997). Faster subtree isomorphism. In *Proceedings of the Fifth Israeli symposium on theory of computing and systems, 1997* (pp. 126–131). IEEE.

Stoy, K., Brandt, D., & Christensen, D. J. (2010). *Self-reconfigurable robots: An introduction*. Cambridge: The MIT Press.

Suzuki, I., & Yamashita, M. (1997). Agreement on a common XY coordinate system by a group of mobile robots. In *Intelligent Robots—Sensing, Modeling And Planning* (pp. 305–321).

Tolley, M.T., & Lipson, H. (2010). Fluidic manipulation for scalable stochastic 3D assembly of modular robots. In *2010 IEEE international conference on robotics and automation (ICRA)* (pp. 2473–2478). IEEE.

Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, *23*(1), 31–42.

Werfel, J., & Nagpal, R. (2008). Three-dimensional construction with mobile robots and modular blocks. *The International Journal of Robotics Research*, *27*(3–4), 463–479.

Yim, M., Shen, W. M., Salemi, B., Rus, D., Moll, M., Lipson, H., et al. (2007). Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, *14*(1), 43–52.

Ayan Dutta is an Assistant Professor in the School of Computing at the University of North Florida. He has received Ph.D. from the Computer Science Department in University of Nebraska at Omaha (May 2017). Ayan's current research interests include self-assembly and locomotion planning for self-reconfigurable modular robots and multi-robot path planning.

Prithviraj Dasgupta is a full professor in the Computer Science department at the University of Nebraska, Omaha and the founder-director of the C-MANTIC Robotics lab. He has led multiple, large, federally-funded projects in the area of multi-robot/ multi-agent systems and published more than 140 research papers in leading conferences and journals in the areas of modular robot systems, multirobot and multi-agent systems. He received his Ph.D. in 2001 from the University of California, Santa Barbara.

Carl Nelson received the B.S. degree in mechanical engineering from the University of Oklahoma (Norman, OK, USA) in 2000, followed by the M.S. and Ph.D. degrees in mechanical engineering from Purdue University (West Lafayette, IN, USA) in 2002 and 2005, respectively. He is a Professor in the Department of Mechanical and Materials Engineering at the University of Nebraska-Lincoln (Lincoln, NE, USA). His research interests include robotics and mechanical design, kinematic analysis and synthesis, modularity in engineering design, and medical applications of robotics.