Artificial Intelligence 223 (2015) 27–64

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization

Hossein Ahmadzadeh, Ellips Masehian *

Tarbiat Modares University, Jalalé-Alé-Ahmad Highway, Tehran, 14115-143, Iran

ARTICLE INFO

Article history:
Received 6 June 2014
Received in revised form 5 January 2015
Accepted 6 February 2015
Available online 12 February 2015

Keywords:
Modular robotic systems
Solution methods and algorithms
Self-reconfiguration
Gait
Flow
Self-assembly

ABSTRACT

While expected applications of Modular Robotic Systems (MRS) span various workspaces, scales, and structures, practical implementations of such systems lag behind their potentials in performing real-world tasks. Challenges of enhancing MRS capabilities not only are limited to designing reliable, responsive, and robust hardware, but also include developing software and algorithms that can effectively fulfill tasks through performing fundamental functions like shape-formation, locomotion, manipulation, etc. Thus, MRS solution methods must be able to resolve problems arising from the tightly-coupled kinematics of interconnected modules and their inherent limitations in resources, communication, connection strength, etc. in performing such functions through domain-specific operations including *Self-reconfiguration*, *Flow*, *Gait*, *Self-assembly*, *Self-disassembly*, *Self-adaptation*, *Grasping*, *Collective actuation*, and *Enveloping*. Despite the large number of developed solution methods, there is no inclusive and updated study in the literature dedicated to classifying, analyzing, and comparing their specifications and capabilities in a systematic manner. This paper aims to fill in this gap through reviewing 64 solution methods and algorithms according to their application in each operation and by investigating their capabilities in (1) modeling and simplifying MRS problems through *Abstraction methods*, (2) solving MRS problems through *Solution* and *Control methods*, and (3) coordinating actions of modules through *Synchronization methods*. Challenging issues of each solution approach along with their advantages and weaknesses are also analyzed and open problems and improvement outlooks are mentioned. Overall, this paper aims to investigate the research areas in MRS algorithms that have been evolved so far and to explore promising research directions for the future.

1. Introduction

Robots were invented with the vision of helping humans do their tasks, especially 4D (Dirty, Dangerous, Difficult, and Dull) tasks, more comfortably. The conventional approach in designing robots has been to design their hardware and software in conformance with the tasks they are supposed to do. Conventional robots can perform certain tasks accurately, however they are not very flexible and adaptive, and thus applications consigned to them heavily depend to their physical structure on the one hand and their controller capabilities on the other hand. The concept of morphologically variable robots firstly was introduced by Toshio Fukuda in 1985 with the name CEBOT (an abbreviation for 'cellular robotic system'),

* Corresponding author.
E-mail address: masehian@modares.ac.ir (E. Masehian).

http://dx.doi.org/10.1016/j.artint.2015.02.004
0004-3702/

a name which was later changed to *Modular Robotic System*, or *Modular Robot* [55,54]. Modular robots were introduced as a remedy to flexibility and adaptability limitations of fixed-body, monolithic robots, comprising a class of robotic systems that can change their shape in conformance to the task and environmental conditions through assuming various morphologies. A modular robot consists of several units with few degrees of freedom (DOFs) called *modules* which are usually equipped with connection mechanisms to cooperatively connect to or detach from each other in order to create complex structures and configurations with many DOFs such as a snake to slither into narrow tunnels, a loop to traverse flat terrains, or a hexapod to navigate rough terrains. Developing practical Modular Robotic Systems (MRS) is tied with numerous hardware and software challenges which according to recent statements by Stoy and Kurokawa [159] and Fitch et al. [52], must (1) be agile enough to reconfigure in a timely manner, (2) be sufficiently robust and fault-tolerant, (3) be scalable to many-module configurations, (4) be reliable in constructing solid and robust structures, (5) exhibit self-reconfiguration for an extended period of time, (6) deal with uncertainty in the environment and sensed data, (7) be able to handle communication unreliability among modules, and (8) deal with limitations of mechatronic devices in terms of resources, sensing accuracy, actuation power, battery life, etc.

While existing review or survey papers on modular robotics have mainly tackled the architecture and hardware aspects of modular robots, in this paper, we particularly focus on solutions to the challenging problems arisen when developing software components for modular robots. In other words, we study the algorithms and solution methods that have been developed in the context of modular robotics for tackling problems that emerge when modular robots perform tasks through some fundamental functions such as *shape-formation* to form a desired configuration from an initial configuration, *locomotion* for moving from a place to another, *manipulation* for physical interaction with the objects, *supporting* and *balancing* for shoring up unstable objects, etc. Instead of categorizing solution methods to these problems merely by their underlying technical and theoretical aspects, we have organized them according to their contribution toward performing nine basic operations performable by modular robots, namely (1) Self-reconfiguration, (2) Flow, (3) Gait, (4) Self-assembly, (5) Self-disassembly, (6) Self-adaptation, (7) Grasping, (8) Collective actuation, and (9) Enveloping. While there is no guideline that prescribes to consider these operations as basic and underlying, we have deduced such a categorization based on reports on various experiments, success stories, and recommendations in the MRS literature. The rationale behind is that these low-level operations can serve as building blocks for generating high-level behaviors such as shape-formation, locomotion, manipulation, supporting and balancing. For example, reaching a desired shape (shape-formation) can be accomplished through either Self-reconfiguration or Self-assembly basic operations depending on hardware and software capabilities of the modules and the task-specific parameters.

The solution methods for achieving the abovementioned nine operations must address some domain-specific issues that make development of planners/controllers for modular robots very challenging. For example, planning for self-reconfiguration of a modular robot is proved to be NP-complete as it has been reduced to known NP-complete problems like PSAT [65] or 3-PARTITION [78]. Thus, employment of *search-based* methods, which are conventional in Artificial Intelligence, is not straightforward in modular robotics as they need to explicitly represent the state-space and then search it for a solution. In fact, search-based methods usually suffer from intractable configuration-space sizes due to exponential growth of the branching factor in the graph representation of the state-space with the increase of the number of modules. Moreover, the tightly coupled kinematics of the connected modules within a configuration not only limits the number of possible actions of each module, but also urges development of such controllers that avoid taking actions that may lead to undesirable conditions in the structure of modular robot. Examples of undesirable states are *fragmentation* of modular robots into multiple parts, *overcrowding* the structure of modular robot by several modules which intend to enter the same lattice position [196], *hollow* configurations in which modules are trapped in a hole or tunnel within the body, and *solid* configuration in which outer modules cannot find a path toward interior of the robot's body [158]. Such problems can be alleviated to some extent through using *control-based* and *agent-based* approaches that plan for reconfiguration in distributed manner based on local information available in modules. However, their underlying methods must mainly concern with keeping the *connectedness* of the modular robot during reconfiguration steps, considering *convergence* to the desired shape or behavior as a result of local interaction between modules, maintaining *adaptability* to the environmental changes, and exhibiting *robustness* to module failures. These challenges get even worse when modular robots operate in conditions of unpredictable events, sensor noise, uncertainty, and actuator imperfection. Under such circumstances, classical engineering approaches fail to function efficiency, while it can be observed that biological systems, despite their relatively simple interactions, can handle such complex situations efficiently in an autonomous and decentralized manner. Therefore, some *bio-inspired* solution methods are devised in the MRS context motivated from self-organization property of multicellular organisms with the aim of emulating self-organizing behaviors of natural systems by modular robots. Overall, the abovementioned challenges have been treated by various solution methods that can be studied under general categories of search-based, control-based, agent-based, bio-inspired, and other intelligent approaches.

Despite the large number of algorithms devised for solving MRS-related problems, there is no explicit and comprehensive categorization of methods in the literature for identifying their specifications, strengths, weaknesses, as well as application contexts and related challenges. Existing reviews, surveys, or books on modular robotics have mainly tackled the architecture, mechanical, and hardware aspects of modular robots, with less emphasis on implemented solution methods. Murata and Kurokawa [117] studied architecture of modular robots and formally classified them into *lattice*, *chain*, and *hybrid* classes. Yim et al. [195] studied modular robots merely from the hardware point of view, and categorized reconfiguration methods according to the source of module motion into *deterministic* and *stochastic* reconfiguration categories. Butler and

Rizzi [23] studied the benefits that the concept of modularity can contribute to locomotion, manipulation, shape formation, and robustness problems in robotics. Gilpin and Rus [60] studied MRS by focusing on self-assembling/disassembling class of modular robots and reported the state of the art of hardware advancements and algorithmic results for creating such systems. Van Hornweder [175] overviewed the hardware of 23 modular robots developed between 1994 and 2009 and discussed their key features and abilities, however without categorizing them. Finally, the book by Murata and Kurokawa [114] reviewed 15 modular robots mainly from the hardware specifications viewpoint and particularly investigated the metamorphosis methods of M-TRAN and self-assembly methods of Fractum modular robots. The book by Stoy et al. [158], however, investigated MRS from both hardware and software aspects by focusing on the intuition behind self-reconfiguration and control, addressing mechanical and electrical design issues, as well as covering search-based, control-based, and task-driven self-reconfiguration planning and locomotion methods.

This paper provides a comprehensive review on algorithms and solution methods of MRS problems along with critical analyses of their advantages and weaknesses, as well as some open problems and improvement outlooks. The organization of the paper is based on algorithms and techniques used in MRS according to their application in solving problems associated with the nine basic operations (Self-reconfiguration, Flow, Gait, etc.) and based on their capabilities in:

(1) Modeling and simplifying the problems, under the title of *Abstraction methods*,
(2) Solving problems, under the titles of *Solution methods* and *Control methods*,
(3) Coordinating actions of modules, under the title of *Synchronization methods*.

Such a scheme is helpful not only for shedding light on capabilities of each solution method in solving particular problems, but also for identifying what sort of algorithms are more applicable to particular operations. In this work, we have tried to cover the algorithmic aspect of modular robotic systems in its broadest sense, through identifying and classifying 64 abstraction, solution, control, and synchronization methods and algorithms, which to the best of our knowledge represents the first and most thorough review in the field.

In Sections 2 to 6 of the paper, all major modeling techniques, algorithms and solution approaches for the Self-reconfiguration, Flow, Gait, Self-assembly, Self-disassembly, Self-adaptation, Grasping, Collective Actuation, and Enveloping operations are reviewed. Since among these operations, Self-reconfiguration, Flow, and Gait are the most-studied operations in MRS, we have reviewed their solution methods in more detail and provided some analyses to investigate the main challenges that these methods may confront as well as their advantages and weaknesses based on either literature facts or our own conclusions. Through the discussions in Section 7, 110 most relevant research papers that have contributions to solution, control, or synchronization methods in MRS are categorized according to their tackled operation and adopted approach. The section also contains about sixty open problems and improvement outlooks collected from the literature or proposed by us. Finally, perspective and concluding remarks are presented in Section 8.

On the whole, this paper aims to investigate the state-of-the-art of abstraction, solution, control, and synchronization methods developed so far in the MRS literature, and to explore promising research directions for the future.

2. Self-reconfiguration methods

The most central activity of a modular robot is to change its shape or morphology. Such a change is the subject of the *Self-Reconfiguration Problem* (SRP), which emerged in the mid-1990s in parallel with evolution of modular robots hardware design. Self-Reconfiguration is the process of transforming a modular robot from an initial configuration to a desired configuration through a set of primitive and module-level actions while the total number of modules is preserved, and *Self-Reconfiguration Planning* is the process of planning such sequence of actions. Fig. 1 shows examples of the reconfiguration process. SRP on one hand concerns with algorithmic and optimization aspects, and on the other hand deals with kinematic constraints of modules. Hence, SRP is not only a software and hardware challenge, but also is such a difficult problem that a general solution for it has not been devised yet [158].

The SRP is formally defined as the problem of planning a sequence of reconfiguration steps that optimally transforms an initial configuration I into a goal configuration G. SRP is proved to be NP-complete for chain modular robots [78]. The main challenge in SRP is the exponential growth of possible configurations as the number of connectors and modules increase. Harary in 1967 proposed a rough exponential lower bound of $a_n > 3.6^n/8$ for the number of morphologically different configurations that can be made by a finite number of cells, in which a_n is the number of arrangement of polygonal cells to form different 'animals' (i.e. configurations) and n is count of square-shaped cells [189]. SRP can even be significantly more challenging than similar problems in conventional robot planning due to the high number of modules and the prevailing simultaneity and asynchronicity in movement and control of modules. However, numerous approaches have been devised to reduce the problem's intractability while a certain level of solution quality is maintained.

In this section we have categorized SRP solution approaches into four classes of *Search-based*, *Control-based*, *Agent-based*, and *Bio-inspired*, each of which studied from two aspects of *Abstraction methods*, which undertake simplification techniques to make SRP tractable, and *Solution methods*, which reflect various ways of finding a feasible solution.

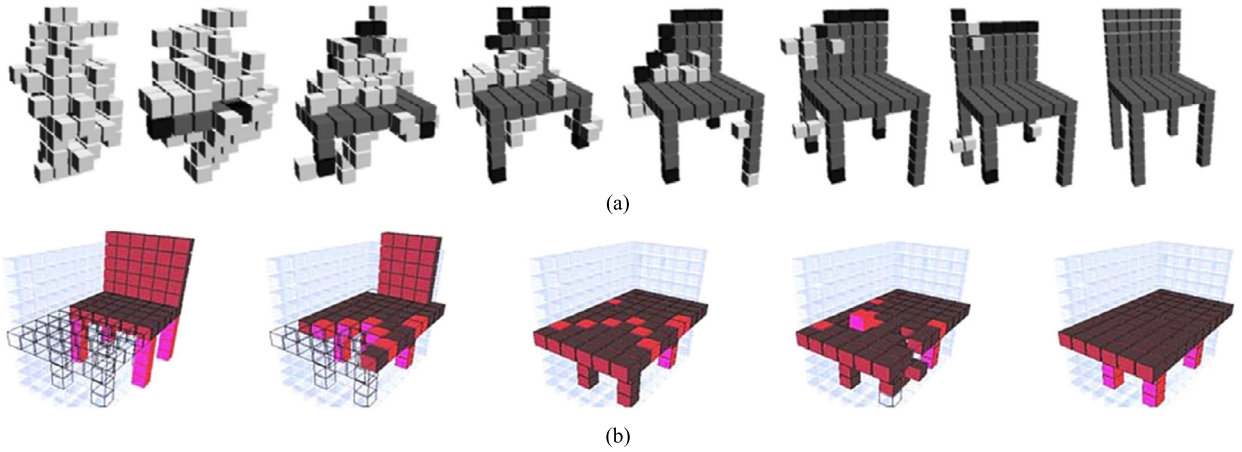


Fig. 1. Examples of reconfiguration planning: (a) self-reconfiguration from an initial random configuration into a chair [157]; (b) self-reconfiguration in presence of surrounding obstacles (walls) [49].

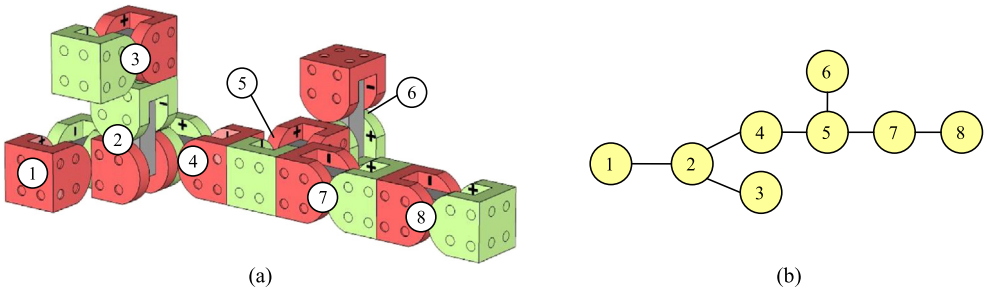


Fig. 2. (a) A configuration of eight M-TRAN modules. The '+' and '−' signs indicated on the modules represent the positive and negative direction of rotation of the actuator in the respective joint. (b) Graph representation for the configuration. The edges represent a connection between modules.

2.1. Search-based approach

As an integral part of classical AI planning methods, search techniques are widely used in solving constraint satisfaction and optimization problems, where all possible solutions to a problem can be represented in the form of a graph $G = (V, E)$ called *Search Space* in which each node $v \in V$ represents a solution to the problem and each edge $e(v, w) \in E$ corresponds to the total cost of transition from node v to node w [128]. Search algorithms provide systematic strategies for searching the graph G until a path that optimizes some quality measure(s) and satisfies particular constraints is found. In the context of SRP, the search space is represented by a *Configuration Graph*, each node of which represents a configuration (solution), and each edge denotes an action (or a set of actions) that transits the robot from a configuration into another. Due to the fact that the branching factor of the configuration graph increases exponentially with the increase of number of modules, the space complexity of search space is intractable. Therefore, it is crucial to take some measures to make the search process tractable, among which, Abstraction methods are used widely and effectively.

2.1.1. Abstraction methods

Abstraction techniques are generally used for reducing the size of the search space and speeding up the search process. They must be compatible with search methods; that is, they have to be defined in such a way that provide a concise representation of the state-space for search algorithms. The most prominent Abstraction methods are as follows:

Graph Representation: Graph theory has been widely used for representing modular robot topologies, through which the morphology of reconfigurable robots is modeled by a one-dimensional combinatorial topology of edges and vertices, creating a planar graph representation, also known as *Connectivity Graph*. In such a graph, nodes represent module IDs and edges denote connections between them [27]. One issue with graph representation is that when modules are connected via different connectors, it is very likely to encounter configurations with identical graph topologies but different functionality. An example of graph representation corresponding to a configuration of eight M-TRAN modules is demonstrated in Fig. 2.

Connector Graph: In order to resolve the problem of identical Graph Representations for configurations with different functionalities, Hou and Shen [78] introduced the Connector-graph (C-graph) method to embed connection ports and orientations in the graph representation of a configuration. In the C-graph representation, a connection between

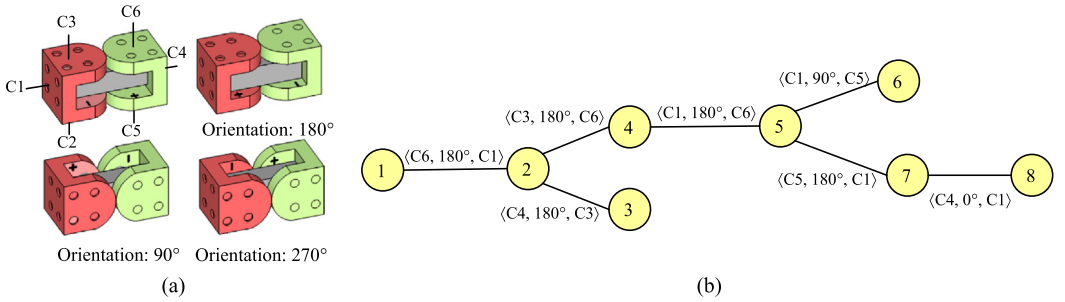
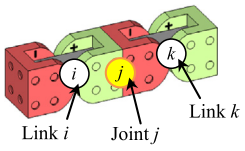


Fig. 3. (a) Illustration of connector labels and orientations for M-TRAN modules. (b) Connector-graph representation for the modular robot configuration of Fig. 2(a).



(a)

$$\text{IM} = \begin{array}{c} \begin{bmatrix} J_1 & J_2 & J_3 & J_4 & J_5 & J_6 & J_7 & \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & L_1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & L_2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & L_4 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & L_5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & L_6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & L_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & L_8 \end{bmatrix} \\ \text{(b)} \end{array}, \quad \text{AIM} = \begin{array}{c} \begin{bmatrix} J_1 & J_2 & J_3 & J_4 & J_5 & J_6 & J_7 & \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & L_1 \\ 1 & 3 & 4 & 0 & 0 & 0 & 0 & L_2 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & L_3 \\ 0 & 6 & 0 & 1 & 0 & 0 & 0 & L_4 \\ 0 & 0 & 0 & 6 & 1 & 5 & 0 & L_5 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & L_6 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & L_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & L_8 \end{bmatrix} \\ \text{(c)} \end{array}$$

Fig. 4. (a) An illustration of a joint and two links of M-TRAN. The Incidence Matrix (b) and Assembly Incidence Matrix (c) representations corresponding to the configuration of Fig. 2(a).

two modules i and j is illustrated by the 3-tuple $\langle C_i, ori, C_j \rangle$, in which C_i and C_j refer to the ID of the connectors of modules i and j, respectively, and ori is selected from the *Orientations* set denoting the orientation of module j relative to module i. In Fig. 3(b) the C-graph corresponding to the configuration in Fig. 2(a) is presented.

Incidence Matrix (IM): IM has been a common approach for representing connectivity graphs in the form of matrices. An IM is an $n \times m$ matrix in which n refers to the number of modules (vertices) within the body and m refers to the number of joints (common faces) among modules [29]. It is based on the two notions of *link* and *joint*, with the former referring to a physical object (e.g. a module) from or to which connections are made, while the latter describes properties of the connection. Each entry $a_{ij} \in \{0, 1\}$ in the IM represents the absence or existence of a connection between link i and joint j, respectively. For example, Fig. 4(a) shows a typical connection between M-TRAN links i and k through joint j. In Fig. 4 (b) the IM representation of the configuration depicted in Fig. 2(a) is provided, in which the links and joints are denoted by L_i (for module i) and J_j (for joint j). In each column J_j, the rows of nonzero entries correspond to the links that are connected to the joint j. Nevertheless, the IM representation does not completely express the modular robot configuration since the information of connection ports and their docking orientation are missing.

Assembly Incidence Matrix (AIM): AIM was introduced by Chen and Yang [30] as an extension to the conventional configuration representation by means of IM. In AIM, '0' or '1' values of an entry a_{ij} in IM is replaced by the port number by which the module (link) i is connected to the joint j. In order to identify isomorphic configurations by AIM, Chen and Burdick [29] utilized the symmetry of modules geometry and graph isomorphism to introduce 'equivalence relations' on AIMs, so that AIMs that belong to the same 'equivalence class' represent isomorphic configurations of modular robots. Fig. 4(c) shows the AIM representation of the configuration depicted in Fig. 2(a), based on the port numbers specified in Fig. 3(a).

Lattice Connectivity Graph: This abstraction method is defined to facilitate measuring distances between modules in a lattice-based configuration. It is based on creating a graph with vertices lying at the center of lattice points occupied by modules, and edges connecting any vertex to all its immediate neighboring vertices. Once such a graph is formed, the length of the shortest path between two given vertices represents the distance between the two modules occupying the lattice points corresponding to those vertices [126].

Configuration Coupling Model (CCM): This model is introduced in [41] as a tool for describing configurations in truss-based modular robots comprised of joint and link parts. CCM is basically an $n \times 3$ matrix where the i-th row represents the joint and link properties of the i-th module in a configuration, such that CCM$(i, 1)$ indicates to what connection port of a joint-module a link-module is connected, CCM$(i, 2)$ indicates the length of the link-module which connects joint modules i and $i + 1$, and CCM$(i, 3)$ indicates the relative connection rotation angle between joints i and $i + 1$.

Logical Modeling: In this representation the SRP is reduced to known NP-complete problems like PSAT (Propositional Satisfiability problem [65] or 3-PARTITION [78]). In PSAT reduction, configurations and transitions are modeled as logical propositions such that solving the SRP is equivalent to solving the PSAT problem. In the 3-PARTITION reduction, the SRP is reduced to the problem of deciding whether a set of integers can be partitioned into subsets with equal sums.

Sliding Cube: This general method abstracts modular robots as *cubes* with connectors on all faces, which are capable to express two motion primitives of sliding on the surface of other modules and convex transition [50]. The concept of abstracting modules by cubes had been previously developed by Butler et al. [21], although it was extended later by Fitch et al. [50] in order to support a class of module hardware instead of one specific modular robot. The Sliding Cube model simplifies kinematic constraints of modular robots and decouples the configuration transition complexities from self-reconfiguration algorithms. It allows modules to translate on the surface or even through the volume of modular robot's body.

Proteo Model: Like the Sliding Cubes model, the Proteo model relaxes kinematic constraints of modules. This model is proposed by Yim et al. [196] for the Proteo modular robot, and encapsulates both module residence spaces and module movement constraints. This model is claimed to be general enough for accommodating a large class of systems with different geometry and motion constraints. It is applicable to regular-shaped modules that inhabit lattice grid positions and abstracts translational steps of modules into discrete motion steps by the constraint that a module can translate into vacant grid cells only when one of its neighboring modules provides support for its motion.

Meta-Module: In this model, a set of modules (virtually) unite with each other to perform high-level actions and offer more motion capabilities than single modules [92,202]. As a result, not only is the size of the search space reduced, but also higher manipulability can be achieved through meta-module actions. Due to usual nonholonomic kinematic constraints of modules, meta-modules can be used in composing a group of modules that can act as holonomic units [40]. Meta-modules transform the SRP from finding sequences of module-level actions into finding a sequence of meta-module actions.

Overall, a critical consideration in implementing Abstraction methods is their ability in distinguishing *isomorphic representations*. Formally, two (dissimilar) representations are said to be isomorphic if they specify a single configuration. Indeed, an effective abstraction should be invariant to isomorphic representations. Since the Graph Representation and Incidence Matrix (IM) abstraction methods are not able to distinguish between isomorphic representations, some isomorphism checking techniques are needed. For example, inspired from the quadratic-time isomorphism checking algorithm for ordered graphs introduced in [79], Asadpour et al. [6] developed an isomorphism-invariant abstraction technique called *Graph Signature* for coding different configurations into a unique string. Also, Park et al. [127] applied spectral graph analysis on the adjacency matrix of graph representation in order to detect isomorphism in representations through analysis of eigenvector of the corresponding adjacency matrix.

2.1.2. Solution methods

Uninformed Search Methods: Covering algorithms like Depth-First, Best-First and Dijkstra search, uniformed search methods are among the most elementary solution methods to SRP. However, the high branching factor of configuration graphs on one hand, and the explicit enumeration of generated and expanded nodes (e.g. the OPEN or CLOSED lists) on the other hand, make the space complexity of uninformed search methods intractable. Therefore these methods are avoided for solving self-reconfiguration problems in general, though they can be used for self-reconfiguration planning applications under the following two scenarios: (1) they can be utilized as core search components when the size of configuration graph is manageable, as studied in [102,182]; (2) they can be employed as auxiliary components in finding a path between two positions in a lattice structure, as studied in [92,138,49,98].

Informed Search Methods: This class of search methods is expected to reduce search efforts due to their ability in guiding the search frontier towards promising directions to the goal. A* search, as the most prominent informed search method, utilizes a combination of a *deterministic* function (for measuring the path cost from the start configuration to a particular configuration), and a *heuristic* function (for estimating the 'distance' from that configuration to the goal configuration). So, implementation of A* for solving SRP requires a metric for measuring the distance of any configuration to the goal configuration. The A* search is guaranteed to be both optimal and complete if its heuristic function is *admissible*, i.e. it satisfies the following three primary properties: (1) *Positive definiteness*: $d(\mathrm{A},\mathrm{B}) \geq 0$, $d(\mathrm{A},\mathrm{B}) = 0 \Leftrightarrow \mathrm{A} = \mathrm{B}$; (2) *Symmetry*: $d(\mathrm{A},\mathrm{B}) = d(\mathrm{B},\mathrm{A})$; and (3) *Triangle inequality*: $d(\mathrm{A},\mathrm{B}) + d(\mathrm{B},\mathrm{C}) \geq d(\mathrm{A},\mathrm{C})$; in which $d(\mathrm{A},\mathrm{B})$ denotes distance between configurations A and B. Various such metrics have been developed for both lattice- and chain-based modular robots, among which the most popular ones are: (1) 'Overlap', which counts the number of modules in $\mathrm{A} \cap \mathrm{B}$ in $O(n)$ time [126], (2) 'Minimal number of moves', which identifies the minimum number of reconfiguration steps between configurations of A and B, although not practical as it requires that at each iteration reconfiguration between A and B be planned optimally [126], (3) 'Optimal assignment', which optimally assigns each element a_i in configuration A to an element b_j in B in $O(n^3)$ time using the Hungarian method such that the total reconfiguration displacements is minimized [125], and (4) 'Graph edit distance' for

counting the least number of edge additions, deletions, and substitutions that transform graph representation of A into graph representation of B [6,5].

Hierarchical Task Network (HTN): As a method of Automated Planning (which is an AI approach that plans for sequences of actions in order to transform the system from an initial state into a goal state), Hierarchical Task Network plans a sequence of primitive actions so that a certain goal task satisfying a condition is realized. HTN is based on recursive decomposition of *compound tasks* into smaller subtasks until reaching *primitive tasks* performable by *planning operators* [56]. In modular robotics context, SHOP2 (Simple Hierarchical Ordered Planner 2) [121] as a domain-independent planning system based on HTN has been successfully employed by Bihlmaier et al. [10] for reconfiguration planning of SYMBRION and REPLICATOR mobile modular robots. In that work, a domain description of the SRP is provided to the SHOP2 planning system and a heuristic function that counts the number of docking and undocking actions is used for guiding the SHOP2 searches.

Divide and Conquer: This method simplifies a problem by dividing its solution space into subspaces (or subproblems) and then solving each problem in its own subspace locally. In order to solve SRP, this method plans reconfigurations via intermediate (also called canonical) configurations. The key challenge here is to characterize intermediate configurations such that not only they should represent most of 'canonical' configurations in the C-space, but also reconfiguration planning between them should be less complex than planning directly between initial and goal configurations. Casal and Yim [27] adopted this method in solving SRP for a class of closed-chain modular robots in which at first both initial and goal configurations are decomposed into *loop* and *chain* substructures using Hierarchical Substructure Decomposition (HSD) rules, and then reconfiguration between intermediate configurations are pre-computed, pre-optimized, and stored in a lookup table. Intermediate configurations in [174] were characterized in such a way that a transition between them requires at most a specific number of reconfiguration actions. Motion planning for many thousands of modules using divide-and-conquer was studied in [9] in which pre-computed reconfiguration plans were reused for recursively reconfiguring groups of modules into meta-modules. Aloupis et al. [3] employed the divide-and-conquer method for reaching goal configurations through merging canonical forms generated by recursively decomposing lattice surfaces into a hierarchy of square cells.

Dynamic Programming (DP): Dynamic Programming is a major classic optimization method for sequential decision problems that can compute an optimal solution through breaking down a complex problem into a collection of simpler subproblems in the form of a sequence of decisions over time. Like the divide-and-conquer method, DP solves problems by combining the solutions to subproblems; however, the difference is that the subproblems in the former are independent while in the latter they are not independent, meaning that subproblems themselves share subproblems [38]. DP utilizes a sequence of *value functions* corresponding to various states of the system over time and employs a backward scheme to evaluate the value functions utilizing a recursive relationship called the Bellman equation. It examines all possible ways to solve the problem and then picks the best solution. As a solution to SRP, DP has been used as a backward search policy in [122] where the minimum cost of reconfiguring an open-chain robotic system was found in $O(n^2)$ time and $O(n)$ space complexities. The DP has also been used for implicitly searching the state-action space to learn a reconfiguration policy that maximizes the expected sum of discounted rewards by solving an underlying Markov Decision Process (MDP) through Bellman equations. MDP-based methods will be discussed in Section 2.3.2.

Rapidly-exploring Random Tree (RRT): In order to deal with overwhelming growth of the search space as the search frontier deepens, the *Sampling-based* approach was introduced in the mid-1990s which implicitly generates the search space and abstracts it by selecting a limited number of points in the free C-space and connecting them to form a roadmap. The main methods in this approach are Probabilistic Roadmap (PRM) developed by Kavraki et al. [85] and Rapidly-exploring Random Tree (RRT) proposed by LaValle [99]. The RRT method generates a random tree rooted in the initial configuration and incrementally connects new nodes until the tree reaches the goal region, after which a simple backtracking from goal to start yields a path. A version of the RRT called *RRT-Connect* was implemented for reconfiguration planning of ATRON modules in [15], the results of which showed that when the SRP is sufficiently difficult, RRT-Connect finds a solution faster that the A* search although the solution quality of the former is slightly poorer than the latter. Methods for improving the search time of RRT against greedy strategies in reconfiguration planning of Superbot and M-TRAN modules was studied in [61].

Distributed and Local Search: Search-based methods generally assume existence of a central unit that observes the robot's proprioception (i.e. the robot's perception about relative positions of its parts within the body) and decides upon reconfiguration steps. However, this is not the case under actual circumstances due to communication overhead, information uncertainty, and difficulties in precise coordination of modules. Therefore, researchers preferred search methods that plan reconfiguration both in distributed manner and locally, so that each module is responsible for planning its actions based on information gathered via local perception and communication. Local search methods are generally used in solving short-horizon reconfiguration, as in [34] a graph representing possible actions of ATRON meta-modules is utilized for local searching and finding the shortest path of meta-actions. Moreover, local search was used in [35] for finding a sequence with the least number of meta-actions in a graph that represents local (i.e., 6 hops long) reachable-space of meta-modules.

Simulated Annealing (SA): The SA method searches the neighborhood of a solution for an improvement of the objective function through hill-climbing, but accepts worse solutions with a probability of $e^{-\Delta E/T}$, in which ΔE is the

change in 'energy' (quality) of solutions, and T is the 'temperature' parameter controlling the search process. For employing the SA in SRP, it is necessary to identify neighborhood configurations, and compute either the energy level of each configuration or the difference in system energy level after each reconfiguration. The SA has been employed in SRP in [126,31]. Pamecha and Chirikjian [125] computed the energy difference $E(C, G)$ between an arbitrary configuration C and the goal configuration G using the two distance metrics of *Overlap* and *Optimal assignment* denoted by δ^c and δ^o, respectively. A configuration C_N is called a neighbor of C iff $\delta^c = 1 \wedge \delta^o = 1$. Then, the algorithm randomly chooses the next configuration N from neighboring configurations which satisfy $\Delta E = E(C_N, G) - E(C, G) < 0$. In case that all neighboring configurations lead to higher energy ($\Delta E \geq 0, \forall N$), the next configuration will be selected according to the normalized probability of $p_i = e^{-\Delta E_i/T} / \sum_{j=1}^{|N|} e^{-\Delta E_j/T}$, where $|N|$ is the number of neighborhood configurations and T is the current system temperature. Experiments showed that the energy function corresponding to the optimal assignment metric yielded better results than the overlap metric in all cases.

Genetic Algorithm (GA): The GA method works by initializing a population of solutions in the form of arrays and assigning a fitness value to each individual of the population (entry of array). Then, at each iteration, a number of individuals are selected according to a selection operator to form 'parents', which are used to reproduce 'offsprings' by means of crossover and mutation operators. Finally, a replacement strategy determines which individuals will be passed to the next generation. This process iterates until a stopping criteria is satisfied. In SRP, the GA-based search have been used for various purposes. For example, it has been: (1) employed particularly for evolving emergent behaviors by modular robots [97], (2) used for evolving algorithm parameters in complex systems which lack obvious ways of finding their optimal values, (3) used for finding network weights in artificial neural networks [34], and (4) used for tuning parameters of Central Pattern Generators of gait controllers [83]. Besides, GA can be utilized in evolving bodies that fit on-hand tasks, as Chen [28] employed it to find task-optimal configurations via a task-oriented objective function which was formulated as a combinatorial optimization problem. The procedure was to search among non-isomorphic configurations identified by the help of defining some chromosomes called *assembly strings*. A hybrid GA–SA method was introduced in [41] for multi-objective optimization of both generation of configuration and selection of joint variables of the modular robots to achieve a specified orientation and position.

2.1.3. Analysis of search-based methods

Due to the overwhelming space complexity of SRP, these methods are not competent in finding optimal solutions, but only suboptimal ones. Indeed, escaping from local minima, handling the intractability of the search space, and distributing the search process through local proprioception (local information about the structure of modular robot) have been among the most challenging problems addressed by different search-based methods. In Table 1 we have compiled a summary of advantages and weaknesses of different Search-based methods in addressing such issues.

In addition, challenges of SRP are beyond finding an optimal sequence of attachment and detachment actions between modules. In fact, module-level actions must be planned accurately so that undesirable conditions that are likely to emerge in modular robotic systems are avoided. The most likely undesired situation that a planner may face is *Fragmentation* in which modular robot's body gets separated into multipartite MRS. Therefore, a planner must assure connectivity of modular robot by avoiding reconfiguration actions that separate robot's body. Subramanian et al. [164] proposed a distributed search algorithm for checking whether displacement of a module endangers MRS connectivity or not. In that work, a special message called 'connectivity token' is circulated among modules with the aim of collecting information about existence of a path that can connect neighbors of a particular module excluding that module. Such a path is searched for in proximity of the modules by means of iterative deepening search.

The planner may face a *Solid* configuration which is a circumstance that an outer module cannot reach its destination inside the body of the modular robot via a path that completely lays on the surface of the modular robot. Conversely, a *Hollow* configuration occurs when a module is confined in a hole or tunnel and cannot find a path to reach the outer surface of the configuration [158]. In order to avoid hollow configurations during self-reconfiguration, Pickem et al. [131] proposed a hole detection algorithm based on the fact that a hole increases the number of connected components in the graph representation of configurations. Their algorithm computes eigenvalues of the graph Laplacian of the configuration adjacency matrix and counts the number of zero eigenvalues to determine the number of holes.

The *Overcrowding* situation mostly occurs in local- and distributed-search methods, when several modules traverse concurrently on the surface and may block each other [196]. This issue, however, can be resolved through coordination between local planners. Also, falling into *local minima* is a very common problem when local search methods are employed. This issue can be addressed by adding randomness or turbulence to the search methods. *Scaffold* structures, introduced in [161], are supplementary structures for assisting the reconfiguration process by restricting modules to certain structures. These structures are defined in such a way that guarantee avoiding the above issues as long as modules are attached to them. Pamecha and Chirikjian [125] mentioned that the 'overlap' and 'optimal assignment' distance metrics (described in Section 2.1.2) used in the Simulated Annealing algorithm may cause *Branching*, which is the undesirable growth of numerous branches of modules from the initial configuration towards the goal configuration, leading to deep local minima.

Table 1
Analysis of Search-based solution methods for Self-reconfiguration planning.

Method	Advantages	Weaknesses
Uninformed Searches	• Optimal (in shortest path algorithms) • Complete • Useful in finding an upper-bound cost of reconfiguration • Proper for short-horizon planning	• Explicit representation of the search space • Memory-intensive in storing *Open*/*Close* nodes • Not applicable to large problems • Centralized: single point of failure • Sequential: not benefiting parallelism • Modules must be synchronized • Communication-intensive • Each module has to know the location of other modules
Informed Searches	• Faster than uninformed search • Can find better solutions compared to Uninformed searches within equal time	• Speed and performance depends on heuristic quality • Admissible heuristics are hard to find • Performance worsens considerably for distant initial and goal configurations • The heuristic's performance is implicitly dependent to the module geometry
Hierarchical Task Network (HTN)	• Is a domain-independent and deterministic planning approach • Is applicable to find plans in domains with very large search space such as SRP • Can produce *anytime*† plans	• The scale of tractable state-space depends on the quality of the heuristic function [56] • SRP domain must be described carefully; otherwise, the search space will become intractable
Dynamic Programming	• Optimal (in Backward search) • Complete • Models SRP as an MDP • Can be executed in parallel	• Single point of failure (in centralized versions) • MDP conditions (i.e. static environment) must be met
Divide and Conquer	• Reduces the complexity of search space • Applicable to large-scale SRPs • Reuses pre-computed reconfiguration plans	• Specifying proper canonical configurations is hard • Huge C-spaces yield numerous canonical configurations
Rapidly-exploring Random Tree	• Better performance than A* search in complex problems • Effective in solving far initial and goal configurations • Faster than A* search in difficult problems	• Solution quality is poor in simple problems compared to A* search • The algorithm is not complete
Distributed and Local Search	• Planning based on local information about the location of other modules • Less communication overhead	• Effective in short-horizon SRP • Modules must coordinate their actions with other modules • The modular robot is prone to fragmentation • Deadlock and Overcrowding are likely to occur
Simulated Annealing	• Can search intractable C-spaces • Can escape from local minima	• Its performance is dependent to the quality of the used distance metric • The distance metric must be admissible • Deep local minima due to the 'Branching' phenomenon [125]
Genetic Algorithm	• Evolves behaviors [97] • Evolves proper bodies [28] • Evolves parameters [83,34] • Also used for multi-objective SRP [41]	• Prone to redundant solutions if chromosomes are not defined properly [67] • Cannot plan in real-time

† An 'anytime' algorithm returns a valid solution even if it is interrupted at any time before it ends.

2.2. Control-based approach

The Control-based approach tackles the SRP from the Control Theory perspective, and deems self-reconfiguration as the result of local and distributed acts of controllers [156]. A direct correspondence can be established between the elements of SRP and those of conventional control systems by mapping *reconfiguration action sets* to the 'input', *consecutive configurations* to the 'output', and *goal configuration* to the 'reference' elements of a control system. Therefore, these elements should necessarily be characterized in any control-based approach to self-reconfiguration. In the Control-based approach, the role of Abstraction methods is to describe the goal configuration for the controller, while the role of Solution methods is to develop mechanisms for selecting suitable actions to transform a modular robot into its ultimate goal configuration.

2.2.1. Abstraction methods

Abstraction methods in the Control-based approach constitute techniques that facilitate presentation of goal configuration either implicitly or explicitly for each module, such that local controllers can easily detect whether or not they have reached the goal configuration. The most straightforward technique is to provide each module with a common-knowledge about geometrical characteristics of the goal configuration. Different techniques have been developed for this purpose:

Coordinates Map: In this technique the workspace is discretized into a 2D or 3D grid and the coordinates of each module in the goal configuration are explicitly specified and stored. This, however, not only consumes large amount of memory in each module, but also urges the modules to have localization capability [32].

Connection Information: In order to reduce the volume of stored data, the Connection Information technique is proposed to just keep the types of neighborhood connections on each connection port at the goal configuration [118]. Although this technique reduces the required memory and does not impose localization capabilities on modules, it cannot characterize the goal configuration completely so that it is likely to end up configurations that despite matching with the connectors' information, do not expose the same geometrical properties of the original goal configuration [158].

Volume Approximation: This method describes the goal configuration as a geometrical expression and requires localization capability of modules. Volume Approximation describes the volume of the goal configuration by boxes of same size as studied in [49,2,58], or by boxes of different sizes called *overlapping bricks* that can overlap each other to represent complex geometries, as used in [161,156]. The accuracy of approximation is affected by the two parameters of size and number of modules. In fact, the more modules (boxes) with smaller size are available, the finer approximation of the goal geometry can be achieved [158].

Surface Approximation: The goal configuration can also be geometrically described by its exterior surface, where its boundary faces partition the space into inner and outer subspaces [51]. This method is similar to what is done in computer graphics for defining different shapes using small triangular faces. However, a module must be equipped by the capability of detecting whether it is located inside, outside or on the boundary of the approximated volume, which is inherently computation-intensive considering the embedded (limited) processors of modular robots [158].

2.2.2. Solution methods

Random Action: The simplest though least efficient control method is to choose reconfiguration actions at random, where in each step the controller hopes to reach the goal position by chance [118]. The performance of this policy is not predictable and obviously has no guarantee to converge to the goal configuration in a finite number of steps.

Predefined Rules: This alternative control-based method encodes human experience and knowledge about appropriate actions in different situations in the form of If-Then rules. The precondition (IF clause) of rules are usually based on local configurations of modules (as in [16]) and need few communications between modules, so rule-based controllers can perform reconfiguration in a distributed manner, though they have to coordinate simultaneous actions of modules in order to avoid collisions, race conditions, and fragmentation of the robot. An example of rule-based solution to SRP is [186], in which a system of hexagonal lattice modules reconfigures from an arbitrary initial configuration into a straight chain goal configuration. The rules are designed to be executed in parallel by modules such that modules located on the 'north' and 'south' sides of the west-most column of modules begin moving over their adjacent neighbors to the east. Modules execute the same algorithm while in order to avoid collisions and fragmentations, the number of modules that can move simultaneously is controlled by some conditions on the direction of the goal chain, direction of the last movements of the modules, and location of goal cells. Although rule sets work much better than the pure Random Action selection policy, their performance is bounded by the designer's capability to predict possible states that the system may face, and so are not scalable to MRS with many modules due to exponential growth of the state-space as the number of modules increases. Predefined rules have also been used as transition rules in Cellular Automata-based solution methods, where rule sets play a central role in producing desired behaviors such as shape-formation and flow-like locomotion. In Sections 2.3.2 and 3.1, more implementations of predefined rules in MRS problems are investigated.

Self-evolving Controllers: This method aims to evolve suitable controllers for on-hand tasks either automatically or semi-automatically, while reducing the need for predesigned controllers and *a priori* information about the environment [205,41]. As evolution of a controller takes place across several populations, this method is suitable for circumstances in which problem specifications do not change significantly from instance to instance. For example, when defining a goal configuration *subjectively* (i.e. not through explicit definition of a specific configuration, but defined to meet a functionality like locomotion) problem specifications do not change much. Thus, self-evolving controllers are mostly adopted to solve problems in which various *behaviors* (without emphasizing on assuming specific morphologies) must be exhibited by modular robots, such as Supporting and Balancing functions, in which a modular robot acts as a stanchion to support an unstable object [34], or Locomotion through Gait [72,133].

Gradient-based: A control policy could be as simple as following a gradient that guides a module to a position in the goal configuration. Thereby the complexity of self-reconfiguration is mostly incorporated into a process which generates the gradients rather than controllers. In the Gradient-based method, some modules or cells of a lattice grid become a 'source' module and play the role of a 'seed'. Seeds then generate 'attractors', which similar to artificial chemicals, propagate and concentrate in all neighboring cells. Non-source modules calculate the concentration of artificial chemicals in their neighborhood and follow them according to the steepest descent scheme [154]. Gradient-based methods have been employed in growing objects from an initial seed module in Stoy [156] and [160]. In [156], for instance, gradient-based motions are produced for cubic lattice modules by utilizing the attraction between vacant spaces and modules, which reside inside and outside of the goal configuration, respectively. Both the modules and holes contribute to generating gradients and direction and distance information of the movement are encoded in

gradients. Here the holes play the role of seeds and produce integer *concentration values* which are propagated to their neighbors. Each neighbor then recursively adds a unit number to the received value and propagates the new value to its own neighbors. In this way, the concentration value is propagated all over the structure. When the gradient propagation front reaches a module, it starts to traverse across the structure by following a decrementing path of concentration values until reaching the source module.

Distributed Planning: In this method, similar to the Gradient-based method, a path is searched from a seed (a target position in the goal configuration) to a 'mobile' module (a module that can be moved without disconnecting the structure) using a distributed depth-first search method. Once a mobile module is assigned to a seed, the mobile module backtracks the path (i.e. the tree generated by the search algorithm) until reaching the seed [25,24]. In order to plan motion of modules in a distributed manner, Butler and Rus [24] developed the PacMan algorithm inspired from a video game with the same name for a system of Crystalline modules, in which 'target' modules attract 'mobile' ones via a motion path drawn for each module by means of special marks called *pellets*. Their algorithm places pellets in the grid environment such that undesirable situations such as deadlocks, fragmentation, and conflicts are avoided. Since in the Distributed Planning method a module 'recruits' a mobile module to fill its unfilled goal position, it is referred in [158] as *Recruitment* method.

2.2.3. Analysis of control-based methods

Control-based methods of self-reconfiguration must be implemented with some considerations: a main issue in generating controllers is their *Convergence* to the goal configuration as they are usually developed based on local interactions and communications between modules. Stoy [156] devised two parallel-running processes of Scaffold and Fill to guarantee convergence and connectivity of an MRS toward constructing a particular shape. Another consideration is *Robustness* which deals with the ability of a controller in preserving system functionality when a sudden failure in modules occurs or external parameters of the system change abruptly. Taking p as the failure probability of a module, the failure probability of at least one module in a system composed of n modules will be $1-(1-p)^n$, which tends to 1 as n increases [191]. This fact implies that a large modular robot is very likely to face situations with defective modules and need recovery to normal situation. Although hardware of modular robots are inherently robust thanks to the presence of redundant modules that can replace each other, it is the responsibility of the controller (or the software) to make a robust control happen. In the context of modular robotics, robust controllers are typically studied through fault-tolerant and self-repairing self-reconfigurable robots, as done in [34,35,145,119].

Adaptability, as another consideration, maintains a controller's capability in recovering the functionality of a system once it is lost, and usually is defined in terms of a time period Δt within which the system functionality is restored. In fact, robust systems can be considered as adaptable systems with the difference that Δt in robust systems is equal to zero [143]. Adaptation has been introduced into robotics through 'classic control architectures', 'behavior-based robotics', and 'evolutionary robotics' [33]. The last methodology introduces simultaneous evolution of robot morphology and controller [151,100] which seemingly fits well to the MRS due to flexible bodies of modular robots. Moreover, a controller must be *Scalable*; that is, it should keep on its performance as the number of modules in the system increases. Parallelism plays a central role in any self-reconfigurable robotic system that cares about scalability. Kotay and Rus [92] proposed a scalable controller based on a parallelized algorithm for the 'Molecule' module that could merge several serial trajectory plans from start to goal positions into a single parallel plan.

Control-based methods are mostly applicable to goal configurations that are defined subjectively based on their functionalities because guaranteeing convergence to a particular geometry is relatively more difficult than guaranteeing convergence to a behavior. Control-based methods generally decide upon next actions based on local communications between modules. Thus, using limited communication bandwidths effectively and preserving the connectedness of MRS via techniques that can check connectivity of the system quickly are of great importance in the Control-based approach to self-reconfiguration. A possible improvement in Control-based methods may be through defining predesigned control rules in higher levels of abstraction, which could be beneficial as this lets them to be extended and reused in a wider range of MRS problems. This can be done for example by designing control rules for meta-modules. However, an issue that is likely to emerge is that meta-actions may not conform to the kinematic constraints of modules. A remedy could be to automatically develop controllers through self-evolution or simultaneous coevolution of body and controller for the task at hand. Table 2 summarizes the properties of Control-based self-reconfiguration methods.

2.3. Agent-based approach

In the software realm, 'Agent-based computing' has emerged as a promising strategy for developing distributed complex systems, in which agents can migrate between different hosts while their codes and data are kept intact [91]. In robotics, multi-agent systems emerged as Distributed Multirobot Systems, in which agents are considered as autonomous rational entities which observe their surroundings through sensors and act in it using actuators. In a broader view, MRS are inherently multirobot systems but mainly differ with them in the kinematic constraints the modules impose upon each other. This analogy paves the way for incorporating some common methods in multi-agent and swarm robots coordination into self-reconfigurable robots. Here we study major agent-based abstraction and solution methods utilized in SRP.

Table 2
Analysis of Control-based solution methods for Self-reconfiguration.

Method	Advantages	Weaknesses
Random Action	• The simplest control method • Efficient when the density of wandering modules† is high [158]	• No convergence guarantee • Unpredictable performance • Needing special care for keeping MRS connected • Inefficient when few wandering modules† are available [158]
Predesigned Rules	• More efficient than random controllers	• Performance is bounded by the designer's skills • Not scalable to many-module MRS
Self-evolving Controllers	• Can evolve controllers for complex behaviors • Suitable for subjectively-defined goal configurations	• Evolves behaviors rather than reconfiguration planners • Hardly applicable to geometrically-defined goal configurations • Offline behavior adaptation
Gradient-based	• Encapsulates motion constrains in gradients • Can be executed in parallel [196] • Avoids local minima when 'scaffolds' are used [158]	• Gradient propagation is communication intensive • A module is not guaranteed to reach the 'seed' module • Trade-off between gradient path optimality and communication [158] • Gradient direction may conflict with kinematic constraints of modules
Distributed Planning	• Guarantees that a wandering module† will be attracted to a vacant position in the Goal configuration • Guarantees that only one module is attracted to a vacant position in the Goal configuration	• The underlying depth-first search does not find the closest wandering modules to the seed

† A 'wandering' module in the system is the one that has not reached to a position according to the goal configuration.

2.3.1. Abstraction methods

Abstract Module Model: Since Agent-based methods address modules individually, special-purpose abstraction models can be developed for abstracting properties of individual modules (such as motion constraints) and numerous intra-module kinematic couplings of modular robots. We call such models 'Abstract module model' which are helpful in encapsulating several module-level actions into a high-level one. For instance, a *four-armed* abstract model is proposed in [150] for implementation of Q-Learning method.

Sliding Cube: In addition to what was mentioned in Section 2.1.1, the general-purpose Sliding Cube model can be successfully utilized for abstraction purposes in various agent-based methods as well, as in [187,177,48,166].

2.3.2. Solution methods

Cellular Automata (CA): Having been introduced by Von Neumann in the 1940s, CA are finite-state machines that decide upon their next state based on both their current state and state of their neighboring cells. CA were first employed in developing distributed reconfiguration controllers by Butler et al. [20], in which a randomly-selected seed automaton grew onto a desired configuration based on local rules. The effectiveness of CA local rules depends on the completeness of the transition rules and is bounded by the designer's capability to cover as many states as possible. Thus, in order to alleviate the incompleteness of the rules in confronting unseen situations, researchers opted to generate transition rules automatically, such as automatic generation of cellular automaton by Stoy [154] in which state transition rules are generated automatically in such a way that the goal configuration ultimately evolves from an initial randomly-chosen seed module.

Markov Decision Process (MDP): An MDP is defined by the 4-tuple $\langle S, A, T, R(s)\rangle$ in which S represents the set of states, A represents the set of valid actions at each state, $T = P(s, a, s')$ refers to the transition probability of reaching a new state s' as a result of performing action a at state s, and $R(s)$ refers to reward value corresponding to state s. By solving an MDP problem, an agent can learn a policy that maximizes the expected sum of discounted rewards $U(s)$ using the Bellman equation $U(s) = R(s) + \gamma \max_a \{\sum_{s'} P(s, a, s')U(s')\}$. MDP is often used in SRP for planning a path between two lattice cells such that a module located off the goal lattice cells is routed to a goal position. For instance, Fitch and Butler [48] utilized the MDP for parallel path planning of modules from their initial positions to positions in a goal region. In that work, the MDP is stored in distributed manner, i.e. state values are updated by adjacent module pairs, and instead of mapping modules to actions, lattice cells are mapped to abstract actions of sliding cubes. However, the planned paths may intersect and lead to collisions between modules, deadlocks, and loss of connectivity of the robot. So, a locking mechanism to handle these problems was introduced in which each module locks its destination cell in order to prevent other modules from occupying it. The lock is released when the module has finished its motion. In a later work, Fitch and McAllister [51] enhanced their algorithm such that 'native' module kinematics are incorporated in the action set of MDP.

MDP formulation is applicable in cases where only one agent (module) aims to learn its state–action policy, otherwise the convergence of Bellman equation is questionable due to violation of stationary world assumption of the Markov process. Therefore, MDP is not suitable for situations in which several modules aim to learn their policies simultaneously and coordinate their actions in such a way that collisions are avoided and connectivity of

the robot is maintained. A resolution to this problem is to represent states and actions of modules collectively so that modules can learn their joint actions as a function of the total MRS state, although it is not practical due to exponential growth of the state-space as the number of modules increases. Another approach is to consider lattice modules as independent agents which plan their paths to the cells in the goal configuration independently and sequentially while considering collision and fragmentation avoidance constrains [139].

Reinforcement Learning (RL): In RL, agents (modules) learn what action to take in various states of the system via trials and reward signals. RL algorithms are generally based on Dynamic Programming and MDP formulation of the world, but with a difference that they are designed to handle large MDPs where complete knowledge of the system is not at hand [165]. In RL, an agent learns a mapping between the state variable s and the action variable a in the form of a policy pair $\pi(s, a)$ in such a way that the long-term discounted reward is maximized for each state–action pair. The most important component of an RL algorithm is the *critic* component, which assigns reward and punishment signals to the adopted actions at each state based on the resulting next state. In order for the critic component to determine to what extent an action contributes to the current state toward the goal configuration, a holistic knowledge about the solution of the problem is required. Therefore, employing RL for solving SRP requires prior knowledge about planning preferences and searching policies in the state-space, which is hard to achieve because the huge size of the C-space dramatically restricts the designer's dominance over the solution of the problem. A remedy to reduce the size of the learning state-space is through limiting ourselves to simpler SRPs and choosing such learning objectives that their critic component can be designed straightforwardly. Once SRP is modeled as RL, Q-Learning, which is a widely-used RL method, can be used for learning action policies. For instance, Q-Learning has been successfully implemented in [150] for learning self-reconfiguration of an MRS consisting of four-armed modules and a learning objective of achieving cooperative motion between adjacent modules in order to transform the morphology of the robot between three presumed shapes of 'Line' (a straight line), 'Mass' (a grid of connected modules), and 'Ring' (a loop made by modules). In that work, Q-learning is implemented for each module independently in such a way that the constituting modules serially select an action based on their Q-value table and then obtain reward and punishment signals accordingly. By assuming modules as individual agents, Distributed (Multi-agent) RL techniques can also be used for learning coordinated actions of modules. RL can be utilized for locomotion through Flow and Gait purposes, too, as we will discuss in their corresponding sections.

Game Theory: Game Theory is the study of independent decision makers and is considered as a common tool in modeling and solving conflicts among interacting agents able to coordinate their actions toward a shared goal. Game theory is widely used in the area of multi-agent and swarm robot systems, and has great potential for solving self-reconfiguration problems. *Coalition Game Theory* as a branch of Game Theory has been effectively used for formation of teams of multiple agents, in which each agent benefits from sufficient incentive to remain in the team and not leave it arbitrarily [135]. Ramaekers et al. [134] utilized this property in solving self-reconfiguration problem because it ensures that modules will remain together during the reconfiguration process and the connectedness of the MRS is preserved while the actions of modules towards the goal configuration are coordinated. Dutta et al. [43] addressed the problem of dynamic identification of new 'best' configuration for a modular robot when the current configuration does not suffice to perform a given task efficiently. In that work, coalition game was utilized to enumerate all possible partitions of modular robots in terms of a 'coalition structure graph'. Then, the problem of finding a suitable configuration reduces to finding the 'best' node in the graph and is solved by a branch-and-bound search algorithm which eliminates coalitions with worse utilities from the search procedure. In that work the uncertainties in operation of modules arising from distance, alignment, motion, and environmental conditions were modeled in the utility functions of modules as Gaussian distributions around the maximum utility received under ideal conditions. In a more recent work, Dutta et al. [42] adopted a similar way for incorporating uncertainties into the utility functions of modules and proposed an approach to speed up the search for coalitions with maximum utilities by limiting the number of modules in each coalition to $n_{\max}$ (determined by the task at hand and module constraints). They devised an algorithm called 'Block Partitioning' which employs a search scheme to selectively generate a coalition structure from an initial one. The algorithm limits the search procedure only to new coalitions that lead to utilities closer to the *ideal* coalition in which the whole structure is partitioned into coalitions of size $n_{\max}$.

Leader-Follower: This method has been widely used both in cooperative multi-agent systems, and for formation control and swarm optimization in such a way that an agent called Leader marches toward goal and other agents follow it. The leader usually is elected according to some qualifications. Followers coordinate their actions toward the system's global objective and in conformance with the leader's actions [103]. Due to the leader's favorable situation regarding the goal, most of the planning task will be performed at the leader's side while the followers just get involved in local coordination or deadlock resolution. Suzuki et al. [166] adopted the Leader-Follower method for reconfiguration of the CHOBIE-II module, where a module assigns an 'undesirability value' to the farthest module in its row and column in the lattice structure. They employed an adjacency matrix to represent a configuration in such a way that an entry in row i (column j) holds value '1' if modules located on both ends of row i (column j) are positioned at the same state in the goal configuration, otherwise, a '0' value is assigned. Then, a *leadership index* is computed from the adjacency matrix such that the module with the highest leadership index takes the

control of the reconfiguration process. The leader also makes decisions for generating necessary transformations to resolve those undesirable states around itself that had the highest contribution in choosing it as the leader. Another example is the technique of *follow-the-leader*, in which 'Kilobot' modules collectively shape and keep a particular formation [137]. Recently, Hou and Shen [77] drew an analogy between the 'distributed configuration matching' problem of the SRP and Distributed Constraint Optimization (DCO) problems. In that work, each module tries to connect itself to other modules in such a way that edges corresponding to its connection with other modules in the graph representation match the edges in the graph representation of goal configuration. Since finding a solution to this problem through distributed coordination of agents suffers from exponential space complexity and requires exponential number of messages, the authors proposed to use a leader module to receive the information about the largest common edge sub-configuration between initial and goal configurations. The leader sorts the received information according to the number of edges and then broadcasts all matched nodes of the goal configuration to the leftover unmatched modules.

Reconfigurable Software: Utilization of *reconfigurable software* in modular robotics originates from the *Code Mobility* potential of those C libraries that are compatible with the mobile agency framework developed by the Foundation for Intelligent Physical Agents (FIPA) organization. An instance is Mobile-C [91], a C/C++ mobile agent system developed based on the 'Ch' scripting language by Vona and Rus [181]. Mobile-C agents are FIPA-compatible; therefore, they are capable of distributing tasks or resources by taking advantage of the *migration* feature that is the agent's capability to change its execution environment (processor or host robot) intelligently while its code and data state remains intact [53]. This feature is used to spread computing system over clusters of modules through sharing the modules' computational resources such that realization of fault-tolerance, agent-based distributed computing, distributed sensor fusion, distributed gait generation, and distributed genetic algorithm can be achieved. Although this method can aggregate computational powers of all modules, a challenging problem here is to poll neighboring hosts and migrate code execution intelligently from a module with flooded CPU to a rather less busy neighboring processor such that utilization of the total computational power is optimized [91].

2.3.3. Analysis of Agent-based methods

Since Agent-based methods are based on distributed and local communication of modules, a problem which may arise is the assurance of whether such interactions converge to a goal configuration or desired behavior. The *Convergence* problem becomes more serious in Multi-Agent Reinforcement Learning methods, where agents (modules) need to exhibit more exploratory behavior in order to investigate the environment and generate behavioral models of other modules. In such circumstances, more exploration of the state space is required, which in turn destabilizes the learning dynamics of other agents and has a negative effect on convergence of the learning process. As in the Control-based approach, *Scalability* and *Robustness* are again two major issues in this approach; in fact, achieving *Scalability* and *Robustness* remains a challenge even in real-world single-agent Reinforcement Learning problems, let alone in multi-agent problems like SRP [19]. The problem of managing *communication bandwidth* arises when agents need to coordinate their actions while executing distributed algorithms. Therefore, Agent-based methods must be designed so that the best utilization of the bandwidth is realized. The parallel execution of state transition rules in these methods may lead to *overcrowding* [196] or fragmentation of the MRS into multiple clusters. On the other hand, Agent-based methods enjoy *fault-tolerance* and *behavior-adaptation* due to redundancy of modules and independence of the self-reconfiguration process from a central node.

Considering MRS as a multirobot system makes utilization of Reinforcement Learning in multi-agent systems to the field of modular robotics attractive. However, Multi-Agent Reinforcement Learning (MARL) encompasses several fundamental challenges, the most prominent of which according to [19] are: (1) difficulty of defining good learning goals for each learning agent, and (2) need for keeping track of the behavior of other learning (and thus nonstationary) agents in order for an agent to coordinate its behavior with them towards achieving a coherent joint action. Moreover, dynamism of modules violates the Markov assumption of 'stationary world', which is central in securing the convergence property of most single-agent RL algorithms including Q-learning.

In Table 3, major characteristics of Agent-based methods for self-reconfiguration are outlined.

2.4. Bio-inspired approaches

The challenge of developing controllers for self-reconfiguration gets more complicated when the designer faces some undesirable facts such as unpredictable events, sensor noise, and actuator imperfection, under which circumstances classical engineering approaches fail to function efficiency. However, it is interesting to observe that biological systems, despite their relatively simple interactions, are able to handle such complex situations efficiently in autonomous and decentralized manner. For this reason, biological systems have been the source of inspiration for many engineering and mathematical methods, such as bioinformatics, biomimetics, bionics, biocomputation, etc.

In the context of modular robotics, bio-inspired approaches take their motivation from the self-organization property of multicellular organisms in such a way that the principles that govern self-organization of natural systems are translated into algorithms that can exhibit comparable self-organizing behaviors in MRS. There are two categories of bio-inspired methods: (1) methods inspired from morphology development and growth process of multicellular organisms, and (2) methods inspired from organisms' physiology (the means by which organs carry out their chemical or physical functions in an or-

Table 3
Analysis of Agent-based solution methods for Self-reconfiguration.

Method	Advantages	Weaknesses
Cellular Automata	• High degree of parallelism [154] • Utilizes local communication • Constant information and computation requirement	• Rule-sets tend to be complex [48] • Difficult to define transition rules • Prone to disconnection [49] • Prone to overcrowding
Markov Decision Process	• Applicable to MRS with large number of modules • Works best when MRS configuration is dense [48]	• Formulates modules as independent agents and thus is prone to collisions, deadlocks, and disconnection issues • Performance degrades in sparse configurations as it is less likely to find safe-to-move modules that do not threaten MRS connectivity
Reinforcement-Learning	• Can learn distributed controllers • Online behavior adaptation [177] • Suitable for lattice-based encapsulations	• Convergence to a solution is endangered due to partial observability of the world for each module • Convergence of single-agent RL methods is threatened due to nonstationary world caused by simultaneous learning of modules [19] • Learning convergence is uncertain • Modules need huge exploration due to dynamism of other modules
Game Theory	• Can coordinate the actions of modules toward a common goal • Can utilize 'coalition game theory' to keep the modular robot connected	• Hard to determine utility values of modules • Complexity of state of the art coalition formation algorithms are $\Omega(2^n)$ and $O(n^n)$ [76]
Leader-Follower	• Is fault-tolerant as no 'Permanent brain' module exists, meaning that any module can become a leader [149]	• Followers must be synchronized with the leader [115] • High coordination demands as followers are likely to block each other
Reconfigurable Software	• Fully distributed • No single point of failure • Is fault-tolerant as module's state can be recovered by neighbors upon failure	• Requires a reliable intra-module communication channel

ganism). The former is applicable to the process of producing different morphologies in modular robots while the latter is mainly applicable in the process of developing controller for behaviors such as locomotion through gait.

In addition, there are other biologically-inspired computing methods such as Genetic Algorithm (GA), Artificial Neural Networks (ANN), and Cellular Automata (CA) which are inspired from natural evolution, brain, and life, respectively. However, despite their bio-inspired nature, in this work we have studied these methods under sections where they have been best utilized. To be precise, GA is studied in Section 2.1.2 where it is employed for efficient search of the C-space, ANN is studied in Section 4.1 where learning of controllers matters, and CA is studied in Sections 2.3.2, 3.1, and 5.1 where modules (agents) as finite state machines interact locally to fulfill operations.

2.4.1. Abstraction methods

Abstraction techniques in Bio-inspired approach are more or less similar to those in the Agent-based approach because both assume modules as entities that cooperatively exhibit a particular behavior. However, a fact to be considered is that the more abstraction is made, the linkage between the original biological system and the artificially-developed system will be weaker [144].

Genome Data Structure: This abstraction method was proposed in [87] to store configuration information of self-reconfigurable robotic systems to make it applicable to the evolutionary computation operations of selection, mutation, and inheritance.

Morphogen Concentration Gradients: This is a technique for abstractly describing target shape information in the methods inspired from the biological morphogenesis process in organisms [7].

2.4.2. Solution methods

Virtual Embryogenesis (VE): The process of embryogenetic development and evolutionary adaptation of living organisms (*EvoDevo*) was the inspiration source in development of the *Virtual Embryogenesis* (VE) method [167]. The VE method uses artificial evolution to evolve processes which manage assembly of mobile units such that *regenerative abilities* are exhibited. In an experiment, the top two rows of a T-shaped robotic organism, which had not been evolved for regeneration, were removed, but then the organism started to regrow from the central axis and restored its original shape after almost 100 time-steps.

Morphogenetic Robotics: Jin and Meng [81] studied the *morphogenesis* procedure in multicellular organisms in order to tackle the challenges of autonomous adaptation of modular robots morphologies with environmental changes. Morphogenesis is a biological process that causes an organism to develop its shape, during which, *gene expression* (the

process of synthesizing of a gene product like protein or RNA) generates various cellular functions. Expression of genes, however, is regulated by their own protein products, as well as by proteins produced by other genes in the same cell or its neighborhood through intracellular and intercellular diffusion. Such interactions form a complex regulation network called *Gene Regularity Network* (GRN). By using GRN, a hierarchical model consisted of a two-layer morphogenetic controller for modular robots has been proposed in [105] and [106], where the layer 1 controller is a rule-based pattern generator that generates appropriate patterns for the current environment and assigns tasks, and the layer 2 controller is a GRN-based controller that automatically generates reconfiguration plans that converge the current configuration into patterns generated by the layer 1 controller.

Protoplasmic Streaming: This method is inspired from slime molds in exhibiting collective behaviors. A cellular slime mold is a primitive organism that fully employs decentralized control for self-organization based on reaction–diffusion of a substance called *Cyclic Adenosine Monophosphate* (AMP). Slime molds are unusual creatures which sometimes live individually as monocellular organisms but at other times form multicellular bodies. As monocellular organisms, they are amoebae that move individually. When environmental conditions deteriorate (e.g. when food is running out), many amoebae gather to form a slug-like entity, which then moves as a whole [116]. A decentralized algorithm to control 2D translational DOF modular robot was introduced in [173] through focusing on the primitive organism of slime mold, in which 'coupled nonlinear oscillators' were utilized to simulate *Protoplasmic Streaming*. In this method, modules are arranged in a network of passive and real-time tunable springs and covered by an outer skin, and are filled with an incompressible fluid (i.e. protoplasm). This MRS could exhibit adaptive amoeboid locomotion by conforming to the principle of 'protoplasmic mass conservation' and by the help of long distance interaction among modules.

Physarum Robotics: Physarum polycephalum is a slime mold that inhabits in shady, cool, and moist areas and can exhibit transportation, navigation, and complex behaviors from very simple local interactions. Inspired by such slime molds, Jones et al. [82] studied complex behavior generation from simple components, and developed a particle-based model which mimicked plasmodium (a mass of protoplasm containing many nuclei) of Physarum polycephalum and could spontaneously generate oscillatory patterns. This model can exhibit self-reconfiguration behaviors which are morphologically adaptive, amenable to external influences, and robust to environmental assaults. Through this model modules are able to perform different forms of controllable motions such as linear, rotational, helical, reciprocal, and amoeboid movements.

2.4.3. Analysis of Bio-inspired methods

Bio-inspired methods of self-reconfiguration originate from biological self-organizing systems and aim to realize self-reconfiguration through *Emergence*, which means to exhibit complex behaviors through simulating simple local interactions between individuals in absence of a central commander. Unlike the conventional top-down approach in which a high-level problem is decomposed into simpler subproblems solvable by recognized algorithms, an emergent behavior is realized through a bottom-up approach. A challenge that must be addressed in developing Bio-inspired methods is to control the Emergence phenomenon such that the desired behavior is achieved [142]. *Robustness* and *Adaptability* issues of Bio-inspired methods have roots in local and distributed nature of algorithms inspired from biological systems. In fact, the redundancy in organic systems is central to robustness realization and failure compensation. However, these methods need a policy to optimize *distribution of redundancy* (the way redundant components are distributed within the structure). For this purpose, Adaptive Deductive Cause-Consequence Analysis (ADCCA) technique was proposed to calculate minimal combinations of failures [69], and Fault Tree Analysis was introduced to identify single- or n-point(s) of failure(s) [179].

Generally, Bio-inspired approaches are patterned after the growth process of multicellular organisms. Virtual embryogenesis (VE) is devised to simulate the EvoDevo-like process, an observable phenomenon in the nature during the development phase of multicellular life forms, in which a multi-modular robot morphology is developed in parallel with controller evolution. Rule sets in VE resemble to Artificial Homeostatic Hormone System (AHHS) rule sets and reflect reactions of a cell to different concentrations of morphogens in VE and flow of hormones in AHHS. Schmickl [144] envisaged *Artificial Life* (ALife) as the ultimate goal of bio-inspired robotic systems, which tries to achieve creation of life-like structures and mechanisms in order to act in the same way biological organisms perform tasks. However, a complication toward achieving ALife is the significant drop in performance of bio-inspired approaches when they are implemented in real robotic systems (i.e., embodiment of the software agent on which the bio-inspired method has been implemented) compared to their well-working computer models and simulations. For example, intra-agent forces and collisions among modules require introduction of complementary algorithms to avoid such physical interferences, which gradually deviates a particular bio-inspired algorithm from its original mechanism. However, performance decline can be overcome by (1) adapting bio-inspired mechanisms to handle issues associated with embodiment of agents through artificial evolution of controllers, (2) enhancing mechanics and the level of bio-mimicry in the forms of agent embodiment through improving hardware engineering approaches and testing of new materials, and (3) incorporating 'mechanical intelligence' into hardware of robots in order to the adverse effects of physical embodiment.

Another impediment on the way of developing Bio-inspired methods for self-reconfiguration is the fact that robotic systems lack self-reproduction ability, which is crucial in biological organisms and their underlying mechanisms. Indeed, while some attempts have been made for self-reproduction and self-evolution of software agents and controllers, such an idea is still far from reach in the case of hardware of physical robots in general and modular robots in particular.

Table 4
Analysis of Bio-inspired solution methods for Self-reconfiguration.

Methods	Advantages	Weaknesses
Virtual Embryogenesis Morphogenetic Robotics Protoplasmic Streaming Physarum Robotics	• Can develop specialized controllers for limbs within the body • Provide a platform for engineering emergent behaviors [144]	• Significant performance drop due to 'embodiment of agents' when implemented on real robots [129] • Mechatronic systems using these methods suffer from lack of reproduction ability inherent in biological mechanisms

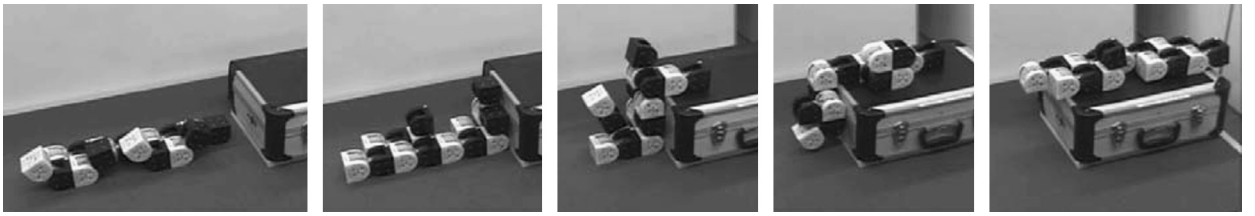


Fig. 5. Illustration of Flow motion by M-TRAN III modules [95].

Schmickl [144] suggests that a trend to solve replication problems can be through utilization of bio-molecules and bacteria in the formation of robots; however, the problem will not be completely resolved as such robotic systems will not be capable enough to form macro-scale modules and perform real-world tasks.

In Table 4, we have summarized the main characteristics of Bio-inspired approaches, as well as their future research potentials.

3. Flow methods

Modular robots can mimic the flow of fluids usually through concurrent reconfiguration of modules, similar to the way that spilt water traverses a terrain toward a sink [22]. Lattice-based modules have the ability to move on the surface of their neighbors along any direction relative to the MRS structure, while the connectedness of the whole body is preserved. Therefore, inspired from the way fluids run toward a sink, such modular robots can relocate through simulating this behavior. In the Flow, modules change their morphology in conformance with the obstacles they meet during locomotion, just like fluids conforming to their stream bed. This way of locomotion is called *water-flow* motion as it simulates the flow of water on the ground, in which modules assume the shape of their underlying terrain while moving forward [22]. While in water-flow motion, Flow is planned at the level of modules, in *Cluster-flow* it is planned for a *block* of modules: Yoshida et al. [198] introduced the *Cluster-flow* motion for a cluster (meta-module) of M-TRAN modules, in which blocks of modules trace a desired trajectory, such that for instance the 'tail' of a cluster translates on the structure to reach the front of the cluster and create a new 'head' (Fig. 5). The cluster-flow motion is similar to the amoeboid movement, in which modules move in parallel through repetitive local reconfigurations [115].

The Flow operation is realized through changes in the morphology of the modular robot; thus it can inherently be modeled as an SRP and solved using the solution methods described in Section 2. However, a subtle point is that consecutive target configurations in Flow are defined subjectively, i.e. they are defined to attain a desired functionality (which is to translate the robot's center of mass along a particular direction) rather than an explicit geometrically-defined shape. Hence, the solution methods of the Flow operation slightly vary from the SRP solution methods.

Besides, since the Flow operation is generally implemented through distributed interactions of modules, it is necessary to adopt synchronization methods to guarantee that the whole cluster of modules remains connected while flowing, and modules are sufficiently coordinated such that 'overcrowding' situations (where several modules intend to enter the same lattice position) are avoided. In the following, we address both the Control and Synchronization methods used in the Flow operation.

3.1. Control methods

Cellular Automata (CA): The local nature of transition rules in CA fits well with Flow locomotion, where modules should decide on their next state according to their local state, local configurations of surrounding modules, and limited sensory data about the environment (e.g. presence of obstacles). Xu et al. [188] developed a set of rules for the class of planar lattice modules based on modules' local perception about their immediate neighboring lattice cells. When executed on the Finite-State Machine of modules, the rules can result in flow-like locomotion behavior in presence of obstacles on a 2D grid. Butler et al. [22] devised a set of transition rules for realizing water-flow motion by MoleCube, M-TRAN, and Crystal modules, in which a modular robot is considered as a particular type of Cellular Automata which runs local rules in each individual cell. Local rule sets are based only on the local configuration around a particular module, and consist of five and eight rules for water-flow in obstacle-free and

obstacle-filled environments, respectively. The rules exhibit a behavior similar to a tank tread, by which modules move in turn from the back of a module cluster over its top and eventually place at the front, either on the ground or on another module. Wu et al. [187] developed CA for flow locomotion in presence of obstacles by the M-Cubes lattice modules. In that work, the state of a cell is defined by the ID of modules located in front of its connectors and the next system state is computed by a transition rule generated by a two-layer artificial neural network with seven inputs (one for the cell state and six for states of neighboring cells) and one output (for the next state of cell).

Murata and Kurokawa [115] employed CA for generating the Flow operation based on an abstract model for a specific meta-module of M-TRAN called 'tile' model, in which a planar regular structure is considered as a plane filled with 2×2 tiles (cells). Their work considered these cells as objects of control (which are fixed in the environment) as opposed to conventional CA implementations that consider moving modules as control objects. Obstacle avoidance is realized by setting a vector field in the cell space through a process similar to gradient generation which places sources and sinks right next to the cells on edges of the structure. Then, Flow on the contour of the environment is realized by using modules that can detect obstacles in their neighborhood. To conclude, the CA methods employed for the Flow operation have roots in fluid flow simulation: for example, the difference between the two approaches of considering cells or modules as control objects stems from the difference between Eulerian and Lagrangian viewpoints in fluid simulating [17]. The former looks at fixed points in the cell space and measures changes of those points, while the latter treats an MRS as a system of particles, and studies the motion for each particle.

Distributed Planning: It is possible to fulfill the Flow operation by parallel execution of locomotion paths planned per each individual module: for instance, the PacMan algorithm by Butler and Rus [24] can be utilized for generating surface-moving systems, and is applicable to unit-compressible systems like the 'Crystalline' module, in which paths of modules are planned in parallel using the depth-first search strategy and path conflicts are resolved in the actuation phase to let modules flow among obstacles or even climb up them (also see the Gradient-based solution method in Section 2.2.2 for more details on PacMan). Fitch and Butler [48] employed distributed dynamic programming for planning an individual locomotion path for each module, where modules use local constant-time search and a module-locking scheme in order to ensure physical integrity of the robot, while following their paths toward the goal of locomotion which has been specified by a simple bounding box.

Hierarchical Planning: Flow can be considered as a two-level planning problem: at the higher level gross locomotion of the modular robot's body is planned, and at the lower level detailed local movements of modules, coordination of their actions and conflict resolution issues are addressed. For example, for cluster-flow of a class of regular structures and especially M-TRAN modules, Yoshida et al. [197] proposed a centralized two-layered planning method where the upper layer (called *global flow planner*) plans the overall cluster motion along a desired trajectory, and the lower layer (called *motion scheme selector*) locally determines low-level module motions by means of a database of If-Then rules. As a result, a block of modules from the tail is transferred toward a given heading direction. The lower layer checks whether paths found by the global planner are valid for each module in the block by repeatedly applying rules which include local motion sequences and are defined according to the initial local configuration of the module. Another instance is the two-layered motion planning approach by Ababsa et al. [1] in which at the higher level a parallel GA search is used for finding the most suitable morphology that a lattice based modular robot (particularly 'Crystalline') must assume during its Flow among obstacles toward a goal position. The genome data structure in the proposed GA contains coordinates of modules in a grid environment, and the fitness function evaluates the Euclidean distance of the robot's center of mass to the destination position. Since each module runs the same copy of the GA algorithm, the population can be divided among modules so that the planning is done in decentralized manner. Once the next morphology of the modular robot is determined, the lower level plans motions of modules using a PacMan-like algorithm to transform the current morphology to the target morphology identified by the higher level.

Reinforcement Learning: The RL method (discussed in Section 2.3.2) can also be utilized for exhibiting flow-like motions. Varshavskaya et al. [177] implemented RL on a 2D lattice modular robot with a learning objective of displacing the center of mass towards a particular direction. Varshavskaya [176] also employed *Distributed Reinforcement Learning* on a 2D lattice-based modular system and set the learning objective to move modules in presence of obstacles such that the robot' center of mass is translated toward a given direction while the connectivity of modules is preserved. Consequently, each action that moves the center of mass toward the desired direction gains a reward. The problem was modeled as a *Partially Observable* MDP (POMDP) since each module could identify the presence of obstacles or modules in its eight immediate neighbors. The modules learned proper reconfigurations through centralized and decentralized methods based on direct search of parameterized state-space called *Gradient Ascent in Policy Space* (GAPS).

3.2. Synchronization methods

Messaging: Modules can coordinate their motions through passing messages between themselves. For example, messaging can be used to assure that the destination of a module is not occupied by another module, and if so, to ask

Table 5
Analysis of Flow-based control methods for locomotion.

Methods	Advantages	Weaknesses
Cellular Automata	• Can exhibit locomotion on the contour of the environment and obstacles • Is based on local rules that are executed in distributed manner	• Rule sets are designed manually and thus require correctness tests for deadlocks, system balance preservation, and connectivity preservation
Distributed Planning	• Modules can flow in parallel and distributed manner	• Prone to generate multiple intersecting paths • The set of paths that can be performed in parallel without having the modular robot disconnected is extremely small [24]
Hierarchical Planning	• Can handle intractable state-space sizes through rough planning at the high level and local execution of plans at the lower level	• Modules move one by one and not simultaneously in the 'global flow planner' system
Reinforcement Learning	Refer to Table 3	Refer to Table 3

the blocking module to leave that location. However, such a coordination scheme requires the messages to be exchanged between all modules of an MRS, which not only is bandwidth-intensive, but also needs a centralized controller for coordination. Murata and Kurokawa [115] introduced 'grouping' of modules as an approach for resolving communication and coordination problems as locally as possible. In that method, modules are grouped together and communication paths between two adjacent groups are determined automatically. A module aiming to occupy a particular location within an adjacent group verifies the vacancy of its destination through messages exchanged at the group level. As a result, by considering meta-modules as groups, flow motion is realized in a distributed manner by means of a hierarchical control structure, in which the lower level controls the motion of meta-modules via coordinating their constituting modules, while the higher-level is responsible for the distributed control of meta-modules.

Locking: Controllers must also care about the arrangement of modules within the robot's structure in order to prevent modules from taking actions that may lead to collisions. The aim of a *locking mechanism* is to synchronize access of modules to the free space in the same way 'critical sections' handle simultaneous access to a shared resource in Concurrent Programming: in MRS, rather than defining 'critical sections' as blocks of program codes, Lund et al. [104] defined them as regions in the free space, and instead of constraining a critical section to be executed by only on module at a time, they devised a locking mechanism in which the module that wants to use the critical section must 'lock' the module that 'controls' the critical section. The controlling module is a module that has to be accessed exclusively in order to avoid collisions (like a module used as a base for a movement). Fitch and Butler [48] utilized the locking mechanism for assuring the connectedness of the modular robot structure during parallel actuation of modules. Connectivity is preserved by locking the modules located on the path searched toward the goal position by the MDP method (see Section 2.3.2), and collisions between modules are prevented through locking the immediate vacant destination position of a module in order to stop other modules from occupying it. After that the module relocates, locked modules are unlocked and allowed to follow their own plans.

3.3. Analysis of Flow methods

Inspired from the fluid flow metaphor, modules in the Flow operation are deemed as liquid particles that slide on the surface of each other while conforming to the shape of the environment and obstacles. A challenge in Flow methods is to produce simultaneous motions of modules while it is guaranteed that there is no chance for deadlock conditions, collision between modules, and risk of fragmentation in the modular robot. Among the three main control methods of motion through Flow, the Cellular Automata and Distributed Planning methods realize Flow through simultaneous motions of modules, while Flow through the *global flow planner* in the Hierarchical Planning method is achieved by serial movements of modules. On the other hand, it is crucial for Flow methods to consider real-world conditions so that the effects of external forces such as weight, friction, and inertia are considered for maintaining the balance and stability of intermediate configurations during the operation. This issue can be addressed in Cellular Automata by devising transition rules that take such external forces into account. A summary of characteristics for different Flow methods is provided in Table 5.

4. Gait methods

Although the Flow locomotion is applicable to structures with many modules, they are extremely inefficient in practice due to the substantial time and energy required in order for the modules to assist other modules in moving on the faces of each other. Thus, researchers incorporated a more energy-efficient robot locomotion technique into modular robots that is locomotion through *Gait*, in which scheduled rotations of joints and adjustments of joint angles leads to translation of a

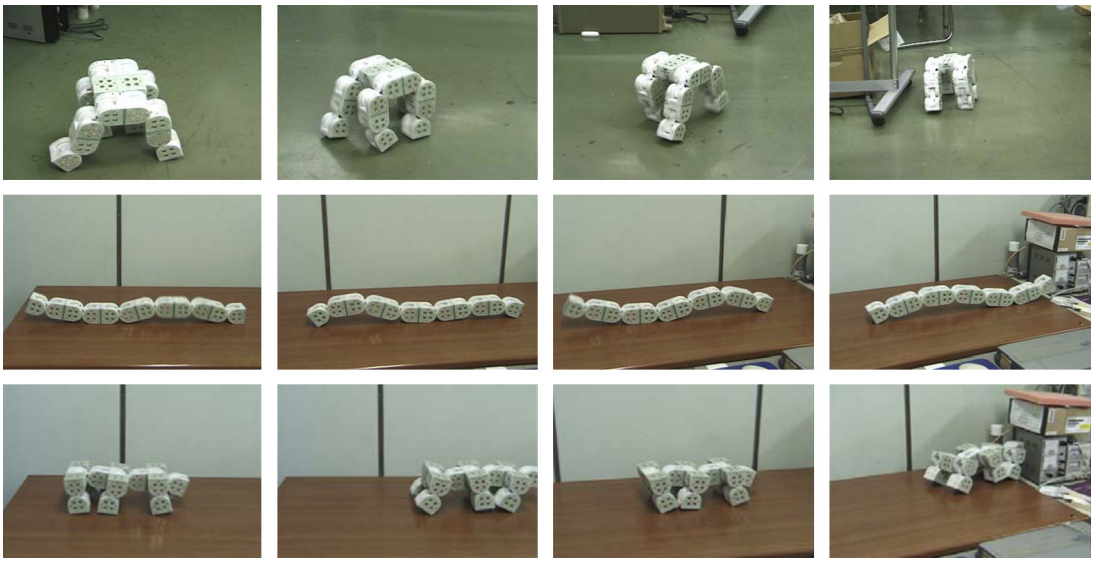


Fig. 6. Examples of Gait locomotion by M-TRAN modules (from left to right) [83].

limbed or limbless body along a desired direction. In Gait motion, since the joints merely rotate at their positions and no attachment/detachment action takes place, the morphology of the modular robot remains fixed during locomotion. Despite the abundance of developed gaits for bipedal, multi-legged, crawler, caterpillar, and snake robots, their direct implementation to modular robots is not straightforward because in most cases 'conventional' robots are designed to suit a particular locomotion gait while modular robots are normally designed to be general purpose.

In addition, modular robots are expected to exhibit several types of gaits depending on various situations they face. Hence, achieving locomotion by gait in modular robots imposes more challenges than in conventional robots. Nonetheless, numerous gait types have been successfully implemented on modular robots, each being inspired from human or animal motion patterns. Some examples are walking gait (either biped or multiped) [149,94,46], climbing gait [101,68,146], rolling and side-winding gaits [63,18,124], looping gait [110,147,192], caterpillaring gait [149,94], and inchworming gait [138,18]. Fig. 6 illustrates some locomotion gaits.

In order to realize a particular gait in modular robots it is necessary to characterize two basic elements of *controller* and *synchronization* methods. In fact, a gait is a set of cyclic actions that need a controller to tell each individual module what action must be done at each time-step. On the other hand, synchronization is crucial for creating harmony between movements of modules so that discrete movements of each module lead to a continuous and smooth gait. In the following, we first cover various gait controllers developed for modular robots and then study their synchronization methods.

4.1. Control methods

In general, gait controllers can be devised either manually (by a designer) or automatically (by the controller itself). In the first type a mapping between states and actions is constructed using lookup tables, periodic functions, and event-driven state machines, etc. as employed in *Control Tables* and *Phase Automata* methods. However, the difficulties associated with hand-designing of state–action mappings instigated methods that can automatically develop gait controllers, including methods that solely automate the controller development process, such as *Central Pattern Generator* (CPG) and *Neuroevolution*, and methods that automate both the development of body and controller so that optimal gait locomotion is realized, such as *Brain and Body Coevolution*.

Control Tables: This is the simplest yet useful control method in which modules are equipped with a list of predesigned mappings from a set of *states* (either motion-step or time-step) into a set of *actions* which tells the module what action to adopt according to its current state. Various gaits for rolling-track, slinky-like, cartwheel, earth-worming, and snake-like locomotion have been implemented by means of mapping from motion-steps to actions, as studied in [189,193,192]. Control Tables is basically an open-loop controller; that is, its adaptation to environmental changes is limited to the flexibility incorporated in the mappings of actions [158]. Another approach in using Control Tables is to map time-steps of each module to actions through a cyclic function with period T. Time-based control tables are implemented in [162] for sidewinder, rolling, and caterpillar gaits.

Distributed Reinforcement Learning: Christensen et al. [37] developed a distributed RL strategy for learning simple gait control tables in which the velocity of the whole modular robot is considered as a global shared reward signal to individual learning modules. Each module selects its action from an action set at random based on ε-greedy policy for

exploration (i.e. choosing various actions) and exploitation (i.e. choosing promising actions with higher probability). Although such a learning strategy is independent of the robot's morphology, it converges slowly to a meaningful behavior. Hence, in order to accelerate the learning process, a heuristic was proposed to bias the exploitation toward the action that has received a reward greater than the maximum expected value of other actions. While this accelerated strategy may converge more quickly, it is prone to remain trapped in local optima conditions longer than the normal ε-greedy policy. These two strategies were experimentally employed in learning gait control tables for realizing typical gaits in ATRON and M-TRAN modules such as snake, walker, and crawler, which each module independently learns what action to do at each time interval.

Phase Automata: This methods was introduced in [204] as a formal model for programming locomotion gaits in chain-based modular robots. It is a compact representation of locomotion gaits in systems with high-level *event-driven state automata* (in which a transition from one state to another is driven by events) and low-level continuous characteristics. Each phase automaton is equipped with an initial real-valued phase delay $\delta \in [0, 1]$ which indicates a *phase-shift* and *initial delay* in periodic and non-periodic gaits, respectively. Phase automaton has been utilized in [194,204,141] for developing distributed and scalable gait controllers. It also has been utilized in an XML-based scripting language to define complex and scalable gaits of modular robots with many degrees of freedom [203]. Phase Automata is a generalization of a similar approach called *Role-based* control model introduced in [163], which has been successfully used for implementing quadruped and hexapod gaits. In Role-based control, each part of a modular robot plays a specific role in the body and follows a particular periodic and continuous time-dependent function which is delayed with respect to its parent module by means of a reset signal sent at certain times.

Central Pattern Generator (CPG): CPGs are biologically-inspired neural networks capable of producing coordinated patterns of rhythmic motor actions while being initiated and modulated by simple input signals even in isolation from motor and sensory feedbacks [39]. In [63] a simple CPG is employed in a robot composed of eight modules in which each module is equipped by a sinusoidal CPG equation that controls the rotation angle of each module. Through the interactions between CPGs and by manual tuning of each individual CPG parameters, five different gaits were developed, namely, sinusoidal, turning, rolling, rotating, and lateral shift. In [111] sensory information feedback was incorporated in tuning CPG parameters so that enhanced locomotion movements such as traveling through an obstacle-filled uneven terrain could be achieved. The Genetic Algorithm was employed in [83,112,96] for automatic optimization of parameters in a network of interconnected CPG oscillators towards finding a stable walking gait where four state variables belonging to each CPG and the connection weights among CPGs were evolved using GA. Furthermore, CPGs have been successfully implemented in developing adaptive gaits of M-TRAN [84], YaMoR [110,152], and Roombots [133] modular robots.

Neuroevolution: The Neuro-Evolution of Augmented Topologies (NEAT) is a Genetic Algorithm for evolving Artificial Neural Networks (ANNs) by altering both weighting parameters and structures of networks [153]. As an extension to the NEAT, HyperNEAT evolves a particular type of ANN called Compositional Pattern Producing Network (CPPN) which in contrast to traditional ANNs can employ a mixture of many activation functions in addition to the widely-used sigmoid function. As a generative encoding description, the HyperNEAT was employed in [71] for generating genotypes that give rise to controllers that work appropriately in different positions of a given 'organism', and for developing gait controllers for locomotion and obstacle avoidance in a corridor with some bricks randomly placed on the terrain.

Brain and Body Coevolution: Throughout implementing conventional gait control methods in modular robots, a subtle presumption is transferred as well; that is, the body of the robot (i.e. the morphology of the modular robot) remains fixed during locomotion. However, this presumption has roots in conventional robotics where controllers are developed to best fit into the fixed morphologies of robots, which is not the case with modular robots which enjoy a reconfigurable and versatile body. In order to develop more sophisticated gait controllers and generate optimal locomotion patterns, and as a generic control scheme that can well-adapt to various morphologies, the concept of Coevolution of 'Brain and Body' (corresponding to the controller and morphology, respectively) was employed in [33,132,45] for coevolving both morphology and controllers through bringing flexible robot morphology, controller development, and environment dynamism together. Brain and Body Coevolution results in generation of 'developmental modular robots' that change their body and controller automatically in harmony with the situations they encounter. Yet again, the *Morphogenetic Robotics* method (which was discussed in Section 2.4.2) can be used for locomotion purposes and for modeling neural and morphological development in single- and multi-robot systems. Jin and Meng [81] showed how the Brain and Body Coevolution method was used for both guiding the flow of a rectangular block of 16 CrossCube modules through a narrow passage (i.e., Flow operation), and forming a vehicle shape with a gait controller which traversed a flat terrain with freely rotating wheels (i.e., Gait operation). A more comprehensive study on Brain and Body Coevolution for locomotion controllers of morphogenetic robotics is presented in [106], and for swimming controller of an underwater module is presented in [180].

4.2. Synchronization methods

Synchronization methods are crucial for coordinating the motion of modules so that a smooth gait motion is attained. These methods can be classified into two categories of *Blocking* (such as Master Control, Leader–Follower, and Hormone-based methods) and *Non-blocking* (such as Synched Internal Clocks and Delayed Signals methods). Although Blocking synchronization methods keep modules synchronized all the time, they are prone to end up with discrete movements accompanied by pauses and delays. Hence, Non-blocking methods are used in order to avoid interruptions in module movements due to late arrival of synchronization messages.

Master Control: In this Blocking method, a module halts its action until a signal conveying information about its next step is received from a synchronization reference. For example, in [189], whenever a module reaches its joint limit, a signal is sent to a *Master-Control* node. After that, the module halts its action until a signal carrying the next behavioral mode is arrived.

Leader–Follower: This Blocking synchronization method was employed by Shen et al. [146] in which one module is chosen as leader and the others as followers. The leader sends a message containing its current state to the module that is connected to its front dock. This follower then updates its status accordingly and by acting as a temporary leader, sends the status message to the next module connected to its head dock.

Hormone-Based: Inspired from chemical signaling in biological cells, the *Hormone-based* synchronization method was introduced in [148], in which 'hormones' are considered as special messages that can trigger different actions in modules. Hormones can be utilized to coordinate motions in presence of limited communication facilities and dynamic network topologies. The *Artificial Homeostatic Hormone System* (AHHS) introduced by Hamann et al. [73] constitutes artificial hormone messages which are diffused in the body of modular robots conveying parameters that control behavior of actuators at each time-step. As a result, movements of modules are synchronized due to diffusion of hormones in the whole body. Moreover, hormone-based synchronization is robust to addition or removal of modules, since (1) the action of each module is basically a reaction to the hormone level it has absorbed, and (2) unlike lookup tables, hormone-based synchronization is independent of modules' IDs. Hormones are used in synchronizing controllers either individually, as in [149], or in conjunction with other techniques, as in [111], in which a hybrid gait control strategy based on hormone-based messaging and CPGs was proposed. In that work, CPGs are responsible for generating motor primitives while hormones propagate sensory feedback information to CPGs enabling CPG network to achieve complex tasks such as obstacle avoidance and traversing across uneven terrains.

Synched Internal Clocks: A simple Non-blocking synchronization technique is through Synchronized Internal Clocks, in which each module is equipped with an internal clock occasionally synched by a master clock. Incorporating synched clocks into a time-based control table results in distributed gait controllers that are not vulnerable to Clock Drift (a phenomenon in which a clock 'drifts apart' from other clocks after some time due to difference in running speed), are free from communication bottlenecks, and scalable to many-module systems [189].

Delayed Signals: A distributed synchronization algorithm has been introduced in [162] which is minimal and robust to loss of synchronization signals and changes in the number of modules. In this method, each module has a cyclic sequence of actions with a period T, but when a fraction of T (e.g. d) has elapsed, a signal is sent to all child modules telling them to make their action sequence delayed by d relative to their parent. This synchronization method was implemented on a robot with eight CONRO modules for creating caterpillar, sidewinder, and rolling wheel gaits.

4.3. Analysis of Gait methods

Being the integral activity in almost all applications, Locomotion plays a vital role in accomplishing tasks. The more efficient the locomotion algorithms are, the more effective the utilization of modular robots in tasks will be. However, regarding that modules in MRS have limited computational, sensing, and moving capabilities, fitting various gaits (such as biped and multiped walking, climbing, rolling, side-winding, looping, caterpillaring, inchworming, etc.) into the hardware and software architectures of modular robots is a main challenge on the way of achieving locomotion through the Gait operation. On the other hand, conventional gait locomotion approaches can be enhanced by taking advantage of the unique properties of MRS (such as modularity, redundancy, and scalability) through introducing gait controllers that can handle addition or removal of modules from the body during the operation. Also, development of fault-tolerant gait controllers which can assimilate failure in modules while keeping the harmony of whole body motion is a research prospect in locomotion through gait by modular robots.

Table 6 summarizes characteristics of different methods employed in generating locomotion gaits.

5. Self-assembly methods

The *Self-assembly* operation is a means for fulfilling Shape-formation function, in which modules aggregate spontaneously to a final formation (configuration). More precisely, Self-assembly enables modular robots to transform into desired

Table 6
Analysis of Gait control methods for locomotion.

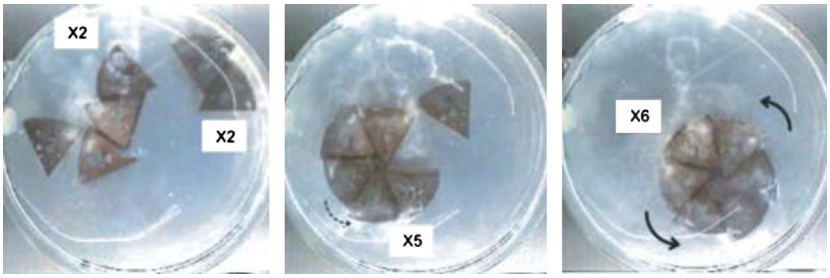
Method	Advantages	Weaknesses
Control Tables	• Simple and easy to implement especially when implemented in centralized manner	• Hard to design control tables • Faulty modules impair harmony of the MRS • Modules cannot be added/ removed during runtime • Not scalable as the number of modules increases [204] • Cannot generate non-periodic gaits • Not adaptable to environment changes
Distributed Reinforcement Learning	• Can learn simple control tables • Can recover the gait in case of module failure or configuration change	• Learning convergence is slow • Acceleration heuristics may keep the learning process longer at local minima states
Phase Automata	• Produces periodic and non-periodic gaits • Able to generate complex gaits • Generates scalable gaits • Applicable to MRS with many DOFs • Can generate terrain-adaptive gaits	• Needs manual design of transition rules
Central Pattern Generators (CPG)	• Produces periodic and non-periodic gaits • Automates gait controller development • Can produce coordinated patterns in absence of feedbacks • Can generate terrain-adaptive gaits	• Parameters optimization becomes challenging as DOF increases [133]
Neuroevolution	• Can develop controllers in conformance with the module position within the body • Can evolve reactive gait controllers that take sensor feedbacks into account, e.g. gaits with obstacle avoidance capability	• The method has been implemented only on a specific quadruped morphology, and its effectiveness in different morphologies and scales needs to be assessed
Brain and Body Coevolution	• Develops body and gait controller simultaneously • The GRN generates global behaviors automatically through implicit local interaction rules [80] • The Morphogenetic method provides a unified framework for multirobot shape formation [81] • The Morphogenetic method can automatically generate adaptive controllers for dynamic environments [107]	• Many underlying genetic and cellular mechanisms in biological morphogenesis remain undiscovered • Morphogenetic robotics is currently limited to simulation

morphologies and concerns motion planning of several dispersed, initially-detached modules which move freely in the environment and can establish multiple bilateral connections to other modules in such a way that a coherent configuration is built up at the end [86]. Fig. 7 shows snapshots of a Self-assembly process. Self-assembly can be used in forming final configurations for both self-actuated modular robots, as studied in [11,172,88,87], and for modules that lack innate actuation ability, like stochastically-driven modules in liquid environment [183,168,169,171]. *Self-repairing* can be considered as an extension to Self-assembly, in which a modular robot repels faulty modules autonomously and replaces them with working ones, whether they are constituted in the current configuration or not [113,47].

As pointed out by Whitesides and Grzybowski [185], *"The concept of self-assembly is used in many disciplines with a different flavor and emphasis in each"*, which leads to lack of a formal definition on the term 'self-assembly' as it embraces a variety of processes ranging from coalition of organic molecules to growth of semiconductor quantum dots on solid substrates. However, we consider self-assembly as a reversible process by which a set of scattered and distinct components can generate a desired pattern or structure. In essence, the study of self-assembly is not limited to modular robotics as it has originated from chemistry, materials science, and biology, where formation of molecular components such as crystals, colloids, lipid bilayers, and phase-separated polymers are concerned [184]. The literature of self-assembling systems is rich with a number of valuable surveys like [185,184,13,66,60], and [4], each of which reviews such systems with regard to particular characteristics. In the following, we concisely review types and control methods of self-assembly and refer the interested reader to the above surveys for more details.

Whitesides and Grzybowski [185] categorized self-assembling systems ranging from molecular to planetary scales according to energy perspective into *Static* and *Dynamic* types: the former involves systems that do not dissipate energy and remain at a global or local equilibrium (such as atomic, ionic, and molecular crystals), while in the latter the dissipation of energy plays a central role in the formation of structures and patterns between components (as in biological cells, swarms, and bacterial colonies). They also introduced two other *Templated* and *Biological* types of self-assembly systems: in the former the final structure is determined by means of interactions between self-assembly components and environment features (as in fluidic self-assembly), while the latter is characterized by variety and complexity of the functions it produces (as in bacterial colonies).

Whitesides and Boncheva [184] maintained that while focus on self-assembly as a strategy for synthesis has been largely confined to molecules, where chemists concern manipulation of molecular structures of matters, this trend is changing by the advent of nano- and micro-scale structures which has drawn interest in self-assembly as a way of aggregating



(a) (b) (c)

Fig. 7. Examples of shape formation through Self-assembly of Tribolon modules [109]: (a) Modules firstly form clusters of two units (i.e., X_2); (b) A cluster of five circular sectors (X_5) is shaped; (c) Modules eventually assemble to a six-unit cluster (X_6) that rotates counter-clockwise.

components larger that molecules. They also mentioned that rules for self-assembly in different scales are similar but not identical, and indicated five characteristics that mainly affect the success of self-assembly in molecular, and analogously, in mesoscopic to macroscopic systems: (1) components, (2) interactions, (3) reversibility, (4) environment, and (5) agitation.

Boncheva et al. [13] studied how self-assembly of components with sizes ranging from micrometers to millimeters can be utilized in constructing functional systems, and enumerated the most important challenges that a self-assembly process faces in building functional systems, especially in nano and micro scales, including: (1) *nature of the function*: to notice that not every function is possible in every size scale, (2) *fabrication of components*: to denote that applications of self-assembly are often limited due to lack of no general methodology of fabricating small nano and micro 3D components, (3) *Interaction between components*: which must be chosen wisely even though wider range of forces such as gravitational, magnetic, hydrophobic and so on can be utilized for interacting μm- to mm-sized components, and finally (4) *Connection between components*: to underline the 'recognition challenge' between components (i.e. the process through which a component 'knows' how and to which component must be connected), which is mainly governed by the design, surface chemistry, and the topology of interacting surfaces.

Groß and Dorigo [66] extended previous surveys on molecular to micro-scale general self-assembling systems to macro-scaled components, and thus to modular robots. They reviewed 21 different self-assembling MRS and presented a taxonomy for the underlying design principles and functions of such systems. They characterized self-assembling systems from four perspectives of (1) physical and electrical design, (2) outcome of analysis of self-assembly experimentation, (3) process control, and (4) functionality. Furthermore, Gilpin and Rus [60] surveyed the history of modular robots and focused on self-assembling robotics from hardware and software perspectives.

Generally, stochasticity of modules' motion is the key factor in characterizing the way a Self-assembly operation is fulfilled. According to the way modules are stirred in the environment and connect to each other, Tolley and Lipson [170] categorized MRS self-assembly systems into (1) those that produce assemblies through interactions between modules and semi-assembled structures while they move stochastically in a stirred environment, (2) those in which stochastically-moving modules deposit onto fixed assemblies, and (3) those in which the final configuration is assembled by means of several 'manipulating modules' working together in parallel.

5.1. Control methods

Regardless of the self-assembly system type, the most common self-assembly approach is to grow the final shape from a so-called 'seed module', which is a dedicated module that initiates a Self-assembly operation and guides the growth process by attracting other modules. The final configuration is then achieved as a result of interactions not only between the seed module and other modules, but also between semi-assembled structures (interaction products) and the modules in the environment. Thus, it is crucial to devise methods that control the interactions between modules (including communication, motion, and connection) according to the state of modules and towards the final configuration so that the likelihood of reaching '*assembly yields*' (the desired assemblies) is maximized.

Simulated Chemical Kinetics: An analogy between chemical kinetics and dynamics of self-assembling systems was drawn by Hosokawa et al. [75], which maintains that (dis)assembly processes can be accelerated or decelerated by controlling the rate of (dis)assembly reactions through manipulating probabilities that stochastically control the bonding policies of modules. Inspired from this analogy, Miyashita et al. [109] studied the self-assembly behavior of Tribolon modules (see Fig. 7) and represented the composition of an intermediate product (cluster) by a state variable X_i, in which i denotes the number of modules in the cluster. The state transition of the system is then expressed in the form of an easily-interpretable chemical reaction (e.g., $X_1 + X_4 \rightarrow X_5$) regarding the constraint that no more than two units can aggregate into a cluster at the same time. In [88] reaction rates were used for exhibiting at what speed assemblies were forming or decaying. Also, global performance of the system was tuned and optimized by tuning the probabilities associated with each reaction. Klavins et al. [89] utilized programmed self-assembly for maximizing the assembly yields through tuning the rates of experimentally-determined self-assembly

reaction pathways. In that work, the stochastic rate of a reaction is determined by an experimentally-measured constant $k(.)$, and the rate of reaction $\mathbf{v} \rightarrow \mathbf{v}+\mathbf{a}$ is defined by $K(\mathbf{v},\mathbf{a}) = k(\mathbf{a})M(\mathbf{v},\mathbf{a})$, where $\mathbf{v}$ and $\mathbf{a}$ are vector representations of a *macrostate* and a *reaction*, respectively, and $M(\mathbf{v},\mathbf{a})$ denotes the number of ways that $\mathbf{a}$ can occur in $\mathbf{v}$. Macrostates illustrate the number of module types in a system; e.g., if the module set is $C = \{C_1, C_2, C_3, \ldots\}$, then the macrostate $\mathbf{v} = (2, 3, 0, \ldots)$ denotes a state in which there are two C_1 modules, three C_2 modules, no modules of type C_3, etc. in the system. This system can be interpreted as a continuous-time discrete-state Markov process, which can, by utilizing the 'Master Equation' from chemical kinetics, determine both the probability of being at a particular macrostate at time t, and identify the steady state of the system (when $t \rightarrow \infty$).

Graph Grammar: Graphs are convenient tools for expressing relations between arbitrary sets of pattern primitives, in which each labeled node represents the pattern primitives and each labeled edge represents relations between pattern primitives. *Graph Grammar* is a formalism to facilitate processing of multi-dimensional patterns through grammatical manipulation of graphs, and is synonymous with 'graph rewriting systems'. Graph Grammar concerns with how to create a new graph from an original graph, in which at every sequential rewriting step, a subgraph of a host graph is replaced by another subgraph while the main problem is on determining how to embed the inserted graph into the 'restgraph' (i.e. the host graph minus the replaced subgraph) [120]. Graph Grammar was first introduced for image processing purposes and then found its applications in areas like biology, mechanical engineering, software engineering, and computer science [44]. In Graph Grammar, the state of the system is shown by means of labeled graphs $G = (V, E, l)$ in which V and E respectively denote vertices (modules) and edges (interconnections between modules), and l is a labeling function to map an alphabetical label set $\Sigma = \{a, b, c, \ldots\}$ into each vertex. For example, l can be a function that assigns the labels 'a', 'b' and 'c' to modules with 0, 1, and 2 bonds (connections). Then, a transition from a state G_k into a state G_{k+1} (which are labeled graphs with identical vertex sets) can be represented by the rule (grammar) $G_k \rightarrow G_{k+1}$, meaning that a copy of the state G_k can be replaced by G_{k+1}. A collection of rules forms a rule set Φ which is applied to the system with a probability set by the designer. Therefore, an assembly is reachable if a chain of rules transit the system from an initial state into a goal state [88]. Adjustment of the probability of each rule leads to *kinetics-based* interpretation of Graph Grammars in which by tuning the probability of rules (i.e. the rate of each reaction) the likelihood of reaching a desired assembly is maximized. Such kinetic-based interpretation of Graph Grammars was employed in [11,88] for generating hexagonal assemblies by six 'Programmable parts' modules.

Graph Grammar was also employed by Klavins et al. [90] for modeling distributed self-assembly of robotic systems. They devised rule sets with at most two modules on the left-hand side of transition rules (also called 'binary rule sets') for generating acyclic graph assemblies and devised rule sets with at most three modules on the left-hand side (also called 'ternary rule sets') for generating general graph assemblies. Pickem and Egerstedt [130] proposed automatic generation of Graph Grammar rule set by means of a two-stage approach that is capable of forming 3D morphologies by modular robots represented by the sliding cube model. In that work, the paths planned by A* between vacant lattice positions in the goal configuration and modules located off those positions are automatically rewritten into a rule set Φ in such a way that the modular robot reconfigures into a goal configuration by decentralized execution of the rules.

Finite State Machines (FSM): In this method each module is equipped with an internal logic that determines its docking behavior according to the sequence of states. Tolley and Lipson [171] suggested an open-loop FSM for self-assembly with the goal of minimizing the required feedback and consisting of four fundamental assembly operations corresponding to the four main states of Attract, Align, Latch, and Release in the system. Wei et al. [182] designed an FSM controller for self-assembly of Sambot modules and categorized modules based on their role into three types of: (1) SEED to denote modules that initiate the Self-assembly operation; (2) Docking Sambot (DSA) to mention modules that participate in growth of the assembly; and, (3) Connected Sambot (CSA) to refer to modules that have connected to the assembly. The proposed FSM was devised for DSA modules and included Wandering, Navigation, Docking, and Locking states. Experiments show that the FSM is scalable and capable of self-assembling into snake, quadruped, H-shape, etc. without any modification in the controller.

Cellular Automata: The CA method (discussed in Section 2.3.2) is also applicable to the Self-assembly operation and can produce distributed controllers that can generate desired structures using local rules. Kotay and Rus [93] studied development of self-assembly controllers using CA and devised a set of ten transition rules for creating a cubic assembly with n modules in each dimension. Stoy [157] proposed a seed-based self-assembly system in which gradients in the system were generated by seeds in order to produce growth in the system. Once gradient paths are generated, the automatically generated cellular automata (according to 3D CAD model description of the final configuration) were utilized to guide the growth process. Simulations showed that time to complete a configuration scaled almost linearly with the number of modules.

Gene Regularity Networks (GRN): Bongard [14] employed GRNs (discussed in Section 2.3.2) for automatic evolving assemblies of hypothetical cylindrical agents by employing *transcription factors* that affect the expression of genes along the genome. In that work, 23 pre-defined phenotypic transformations such as increasing the length, dividing a unit into two, and deleting or modifying the properties of the agent's neurons or synapses were initiated. Kernbach et al. [86] utilized the GRN method and its algorithmic inspirations to propose a self-assembly scheme which can produce functional assemblies from heterogeneous modules. In that work, regularity networks were employed to

Fig. 8. Self-disassembly of Miche modules [58].

generate environment-dependable topologies (Φ^S) with a process analogous to gene expression process. Then, the expressed topologies were optimized subject to the number of on-hand modules in the assembly, availability of docking ports, assembling dynamics (e.g. the number of collisions), etc. Scalability tests showed that this approach can scale well to systems with 5 to 30 modules.

For in-depth analyses of Self-assembly methods, readers are referred to the surveys [13] which covers 26 examples of self-assembled systems at millimeter scale, and [66] which studies 21 macroscopic self-assembly systems.

6. Other operations

In the MRS literature, Self-disassembly, Enveloping, Grasping, Self-adaptation, and Collective Actuation operations are addressed relatively much less than the Self-reconfiguration, Gait, Flow, and Self-assembly operations which were reviewed in previous sections. This fact is not only due to the underlying complexities of such operations, but also because of their highly demanding sensing, actuation, and communication capabilities which mostly surpass current hardware and software specifications of modular robots. In this section we review the most important works on each of these operations.

6.1. Self-disassembly

Self-disassembly refers to the process of reaching a goal configuration via detaching some extra modules from a unified initial configuration, similar to a sculpting process. Through the Self-disassembly operation, a desired shape is produced by removing unnecessary modules from a close-packed configuration such as a cubic block of modules (as shown in Fig. 8). As stated by Gilpin et al. [57] "*a limited amount of past research has focused specifically on self-disassembling systems as a basis for shape formation*". Self-disassembly has some advantages over existing self-assembly approaches [58], including: (1) disconnection is relatively simpler than establishing a connection, and so requires simpler actuation mechanisms in modular robots, and (2) making a connection (i.e., assembly) requires a module to seek and align with a connection port on another module, while in disassembly a module can simply detach unnecessary connections. However, there are two tradeoffs: firstly, the modular robot must be pre-assembled prior to self-disassembly process, and secondly, exertion of external forces is required for repelling unwanted modules from the system.

Gilpin et al. [58] proposed an interactive Self-disassembly method for MICHE modules based on local communications between modules through five steps: (i) *Neighbor discovery*: modules are added to an initial assembly one-by-one, and upon insertion, each module immediately commences to find its neighboring modules and initiates magnetic connections to them so that a solid initial block is formed; (ii) *Localization*: each module determines its coordinates in the assembly and sends its location data to a host computer where a 3D model of the initial configuration is displayed; (iii) *Sculpting*: at the host computer, an operator virtually sculpts the block into the desired configuration and sends back the shape information to the system; (iv) *Shape distribution*: an 'inclusion' message is sent to a single 'root' module which then gets propagated among modules in the system telling them whether to remain or leave the system; and (v) *Disassembly*: any module that is located off the grid of final configuration disconnects itself from its neighbors and is ultimately repelled from the system. Later, an enhanced version of this method was implemented on *Smart Pebbles* modules [57,59], without the restriction of forming only a single shape with a unique 'root' during self-disassembly process. In addition, the new algorithm is able to produce multiple shapes, either contiguous or separated, with no restriction in the number of modules that must be repelled from the initial assembly. The enhanced algorithm incorporated a new way for propagating the inclusion message at step (iv) of the disassembly algorithm (i.e. *Shape distribution*) in order to avoid message flooding in the system: inclusion messages carry both *hop count* and *branch direction* pieces of information and upon traveling a specified number of hops from the root, the message branches off of the chain in the specified branch direction.

6.2. Self-adaptation

The purpose of *Self-adaptation* is to utilize reconfigurability of modular robots in recovering system functionality once equilibrium situation is lost. It is applicable in constructing environmentally-adaptive structures such as self-balancing

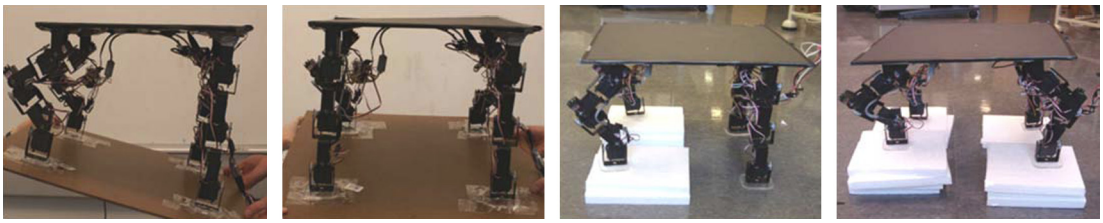


Fig. 9. An example of Self-adaption operation by modular robots in creating a Self-balancing table [201].

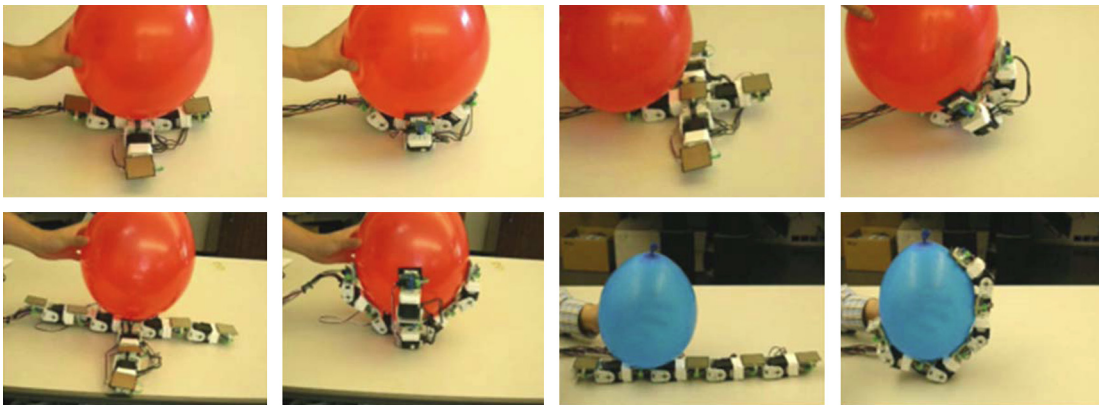


Fig. 10. Examples of grasping objects by modular robots through Modular grippers [200].

tables [201], adaptive columns to withstand external forces [200], and terrain adaptive bridges [199]. An example of a self-balancing table is depicted in Fig. 9. Self-adaptation utilizes reconfigurability of modular robots in constructing structures that reconfigure themselves in response to environmental changes, so that a particular Function (such as Manipulation, Supporting, or Balancing) is achieved. Realization of this operation is convenient for Truss-based modules as they can control the length of linear actuators precisely and hence adapt the structure of modular robots to environmental conditions. A challenge in self-adaption is to reach a global perception of the current MRS state based on local and limited sensing of individual modules nearly in real-time. This problem is mainly addressed by algorithms that propagate sensory data among modules and then utilize these data for deciding upon actuation feedbacks.

Yu and Nagpal [200] proposed a stepwise algorithm for producing a 'pressure-adaptive column' system from modular robots in three steps: (i) modules detect the presence of unknown objects on the structure, (ii) each module transmits the value of its pressure sensor to its neighbors, and (iii) each module controls its actuator parameters based on sensor feedbacks received from neighbors. Another study on creating a 'flexible surface' with supporting legs was conducted by Yu et al. [201] in which a distributed feedback control algorithm continuously iterates between two steps: (i) collecting tilt sensor information from neighboring surfaces and computing an *aggregation feedback* in 'pivot' modules (those that connect legs to the surface), and (ii) controlling actuation of each module in the legs according to the received aggregated feedback.

6.3. Grasping

The aim of *Grasping* operation is to hold an object via grippers that are formed by particular arrangement of modules. In other words, the reconfigurability of modular robots can be utilized for developing versatile grippers able to grasp objects with unknown sizes or shapes. This is especially beneficial when a fixed and pre-designed morphology for gripping hands does not cover our grasping needs. Fig. 10 illustrates some examples of Grasping by modular robots. Bojinov et al. [12] proposed a gradient-based control method for lattice modules which uses seeds for growing 'emergent' structures that can exhibit correct properties associated with grasping an object. In that work, two types of seed modules are introduced: SEED modules, which are regular seed modules that guide the growth process, and TOUCHSEED modules, which are SEED modules that have changed their status upon touching an object. Introduction of these two seed types allows wandering modules to identify whether they are growing toward the object or are about to grasp it. Y. Guan et al. [68] introduced mechanics for a modular robot equipped with two special grippers not only for grasping but also for climbing like chimpanzees.

A 'modular gripper' that can manipulate fragile objects was studied in [200], in which through employing distributed sensing and actuation, chain modules actuate in order to grasp an object in a fixed morphology (see Fig. 10). A three-step algorithm was devised for this purpose, in which: (i) modules reach for an object and once a module senses the object, it sends a message to its neighbors which propagates within the structure, (ii) each module sends the value of its pressure sensor to its neighbors, and (iii) each module computes its actuator parameters based on received sensory feedbacks from its neighbors. The second and third steps of the algorithm are repeated until all modules apply equal pressure on the object.

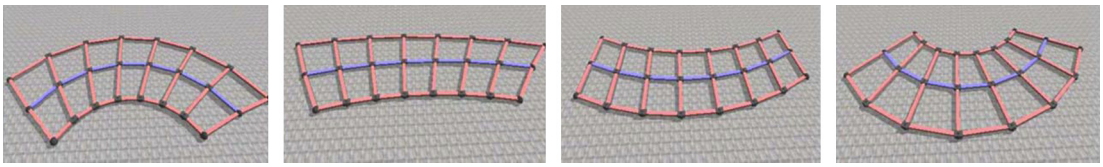


Fig. 11. Collective Actuation of Morpho modules to exhibit Inversion [199].

Fig. 12. Enveloping operation by M-TRAN modules.

Gonzalez-Gomez et al. [62] proposed a novel touch sensor mechanism based on touch strips which was implemented on Y1 modules. They made a snake configuration by 30 Y1 modules, and simulated the operation of grasping a cylindrical object by the snake (the same way a python wraps around its prey) through which contact points and the applied forces between the snake and the cylinder were obtained. Also, based on the *Gene Regulatory Network* bio-inspired method (Section 2.4.2), coevolution of hand morphology concurrent with its controller was addressed by Jin and Meng [81], in which the shape, number of fingers, number of finger segments, and even the number of arms can be evolved together with the controller in order to grasp considerably different-shaped objects.

6.4. Collective Actuation

Effective interaction with objects in the real environments calls for strong and powerful manipulation arms. *Collective Actuation*, as a way of achieving this goal, has been developed to overcome the problem of less powerful modules as a result of reduction in modules' size. Through this scheme, modular robots composed of tiny modules can build strong and powerful manipulation arms, and synergistically attain larger forces, torques, and movements than what is attainable by the sum of the capabilities of individual modules, thus creating more effective and powerful bodies [26]. Fig. 11 shows a sequence of shapes generated by Collective Actuation of Morpho modules.

A solution to this problem is through exploiting mechanical advantage principles in the design of modular robot configurations. For instance, Yim [189] designed mechanics of Polypod actuators in order to benefit from near singularity conditions in the Jacobian matrix of the joint matrix representation, in which the actuation arm becomes near-perpendicular to load forces when it is fully extended. Under such circumstances, the mechanical advantage will be near infinity, which when combined with a ratcheting mechanism having very little backlash, can move heavy loads in small distances. Yim et al. [190] devised configurations with near singularity Jacobian matrix that could produce very large mechanical advantage over a small range. The effective range of such large mechanical advantage can be further extended through adding locking and ratcheting mechanisms to the modules. Another solution is through inspiration from the anatomy of animals in making configurations that can withstand heavy loads. Following this scheme, Stoy [155] introduced 'Deformatron' modules in which each module can play one of three roles of *Muscle*, *Tendon*, and *Bone* in the structure of the modular robot: Muscle robots can be considered as robotic equivalent of biological muscle fibers with actuation power proportional to the number of parallel module chains in the muscle, Tendon is referred to a module connected to a neighbor using an extendible unactuated connector to transform translational movements of muscles into rotational movements, and Bone structures are responsible for transferring movements of muscles over longer distances. Similarly, Christensen et al. [36] introduced 'ATRON-anatomy' which simulates muscle, bone, and joint parts in the configurations of ATRON modules for generating modular robots that are able to scale up their diversity of functionality with the number of modules. Yu et al. [199] simulated collective actuation of Morpho modules for exhibiting 'Inversion' behavior of volvox (a genus of chlorophytes), which is a bio-inspired way for generating self-deformation in the body. As illustrated in Fig. 11, Inversion occurs as a result of actively contraction of outer arc links and simultaneously expansion of inner arc links.

6.5. Enveloping

A target object is said to be *Enveloped* by a closed configuration if it is confined within a number of modules and cannot escape from the boundary of surrounding modules unless by crossing them (as shown in Fig. 12). Enveloping is useful when a dynamic 'cage' is required around one or more objects, just like shepherding or herding behaviors in flocking robots to guide a number of objects to a goal region without touching them, but through repelling forces between modules and subjects of guidance [74]. Miao et al. [108] proposed a distributed algorithm for enveloping an object inside a hexagonal

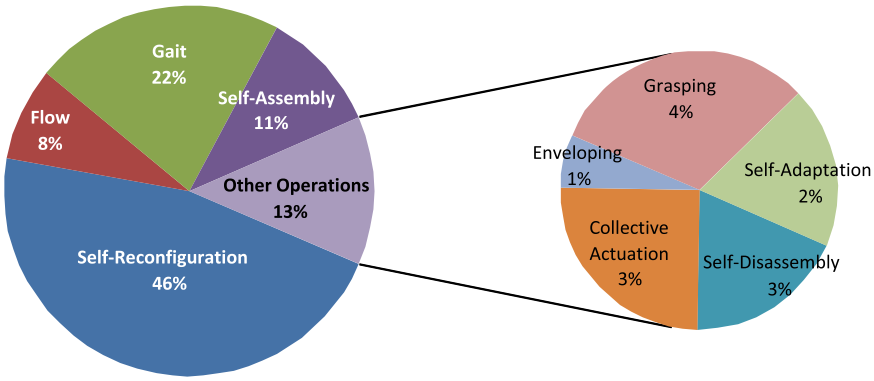


Fig. 13. Percentage of publications dedicated to various modular robotic operations based on data extracted from the selected 110 works.

lattice environment based on local communications among neighboring modules and between modules and the lattice node containing the target object. In that work, a stepwise algorithm based on the Lattice Distance $LD(m_i, G)$ between module to the target object G has been proposed for updating the position of modules. Module m_i observes whether its neighboring *feasible* cells (a lattice cell that can host modules) are closer to the target G than itself: if there is any such a cell, the module waits for a while to ensure that the cell is still available and then updates its position. Otherwise, the waiting period will be longer if there are feasible cells that keep $LD(m_i, G)$ unchanged. In the case that no feasible cell is available after the waiting period, the module m_i will remain still. Finally, an *isolated* module (a module with all empty cells as neighbors) is not allowed to move at all. It is proved that this algorithm can fulfill the Enveloping operation, and an upper bound for the convergence time of the Enveloping configuration is provided.

7. Discussion

In the previous sections we attempted to present a thorough and methodical review on abstraction, solution, control, and synchronization methods for various operations in modular robotic systems, covering 125 most-relevant publications (summarized in Table 7), of which 15 works mainly focused on abstraction or synchronization methods and the remaining 110 presented novel solution or control methods, though a number of them contained original abstraction or synchronization methods as well. We identified 64 distinctive solution/control methods dedicated to various basic operations in MRS such as Self-reconfiguration, Gait, Flow, etc. and plotted the number of publications on each operation in Fig. 13. It is observed that Self-reconfiguration and Gait operations are the two most-addressed subjects in development of MRS algorithms, and together with Self-assembly, they constitute nearly 79% of the total 110 considered publications. In Fig. 14 a chronological diagram for different solution methods is provided, where the number, range of publication years, and average publishing year of the research papers related to each solution method are superimposed in a single diagram.

Fig. 14 is also accompanied by citations to the 110 relevant references, and thus can serve as a quick guide to the publications on each solution method. Needless to say, some papers might have addressed more than one operation at the same time, and so a single work may have been cited in two or more operations.

The chronological diagram reveals that among the four approaches to Self-reconfiguration, researchers were initially focused on Control-Based and Search-Based methods, but the other two, i.e. Agent-based and Bio-inspired approaches (especially the latter), emerged quite later. Despite this fact, the number of works on the Agent-based approach surpassed Control-based self-reconfiguration works in a much shorter time interval. This might be due to the capability of Agent-based methods to solve SRP in distributed manner on one hand, and the fact that they are inherently more 'intelligent' than Control-based methods on the other hand. Regarding Gait, although early research was focused on manually designed Control Tables, later it shifted to methods that can generate various gaits automatically, and particularly to the promising area of CPG-based methods that produce periodic or non-periodic terrain adaptive gaits even in the absence of feedbacks. The relatively new paradigm of *Coevolution of Brain and Body* has proved to be very successful in diverse operations like Self-reconfiguration (Section 2.4.2), Gait (Section 4.1), Self-assembly (Section 5.1), and Grasping (Section 6.3), mainly due its ability in evolving controllers considering the characteristics of modular robots in different morphologies. However, this research area is "*very much limited to computational simulations*" and "*appropriate hardware for morphogenetic robotics, including programmable materials and adaptable sensors and actuators, is to be studied*" [81].

7.1. Improvement outlooks

As mentioned earlier, the gap between achievements in modular robotics and their real-world applications is significant such that many researchers in the field have noted in their papers, books, or lectures at conferences. Indeed, a part of this gap is due to the drawbacks or failures of implemented algorithms to carry out complicated tasks, especially with large number of modules. By investigating the weaknesses of the most well-known methods in MRS, we were able to

Table 7
Summary of main Abstraction, Solution, Control, and Synchronization methods used in MRS Operations.

Operation	Approach			Method name	Employed by
Self-Reconfiguration Planning	Search-based	Abs.	1	Graph Representation	Casal and Yim [27], Baca et al. [8]
			2	Connector Graph	Hou and Shen [78]
			3	Assembly Incident Matrix	Chen and Burdick [29]
			4	Lattice Connectivity Graph	Pamecha et al. [126]
			5	Configuration Coupling Model	Dong and Li [41]
			6	Graph Signature	Asadpour et al. [6]
			7	Logical Modeling	Gorbenko and Popov [64,65]
			8	Sliding Cube	Fitch et al. [50]
			9	Proteo Model	Yim et al. [196]
			10	Meta-Module	Kotay and Rus [92], Zhang et al. [202], Dewey et al. [40]
		Sol.	11	Uninformed Searches	Kotay and Rus [92], Rus and Vona [138], Fitch et al. [49], Liu et al. [102], Larkworthy and Ramamoorthy [98], Wei et al. [182]
			12	Informed Searches	Pamecha and Chirikjian [125], Pamecha et al. [126], Asadpour et al. [6], Asadpour et al. [5]
			13	Hierarchical Task Network	Bihlmaier et al. [10]
			14	Dynamic Programming	Nourollah and Razzazi [122]
			15	Divide and Conquer	Casal and Yim [27], Ünsal et al. [174], Bhat et al. [9], Aloupis et al. [3]
			16	Rapidly-exploring Random Tree	Brandt [15], Golestan et al. [61]
			17	Distributed and Local Search	Christensen [34,35]
			18	Simulated Annealing	Pamecha et al. [126], Chiang and Chirikjian [31]
			19	Genetic Algorithm	Chen [28], Lal et al. [97], Dong and Li [41]
	Control-based	Abs.	20	Coordinates Map	Chirikjian [32]
			21	Connection Information	Murata et al. [118]
			22	Volume Approximation	Fitch et al. [49], Aloupis et al. [2], Gilpin et al. [58]
			23	Surface Approximation	Fitch and McAllister [51]
		Sol.	24	Random Action	Murata et al. [118]
			25	Predefined Rules	Christensen [34], Wong and Walter [186]
			26	Self-evolving Controllers	Christensen [34], Zykov et al. [205], Hamann et al. [72], Pouya et al. [133], Dong and Li [41]
			27	Gradient-based	Stoy [154,156], Stoy and Nagpal [160]
			28	Distributed Planning	Butler and Rus [25,24]
	Agent-based	Abs.	29	Abstract Module Model	Shiba et al. [150]
			30	Sliding Cube	Wu et al. [187], Varshavskaya et al. [177], Fitch and Butler [48], Suzuki et al. [166]
		Sol.	31	Cellular Automata	Butler et al. [20], Stoy [154]
			32	Markov Decision Process	Fitch and Butler [48], Sadjadi et al. [139], Fitch and McAllister [51]
			33	Reinforcement Learning	Shiba et al. [150]
			34	Game Theory	Ramaekers et al. [134], Dutta et al. [43,42]
			35	Leader-Follower	Suzuki et al. [166], Liu and Wu [103], Rubenstein and Nagpal [137], Hou and Shen [77]
			36	Reconfigurable Software	Ko and Cheng [91]
	Bio-inspired	Abs.	37	Genome Data Structure	Kernbach et al. [87]
			38	Morphogen Concentration Gradients	Ashe and Briscoe [7]
		Sol.	39	Virtual Embryogenesis	Thenius et al. [167]
			40	Morphogenetic Robotics	Jin and Meng [81], Meng and Jin [105], Meng et al. [106]
			41	Protoplasmic Streaming	Umedachi et al. [173]
			42	Physarum Robotics	Jones et al. [82]
Flow		Cont.	43	Cellular Automata	Xu et al. [188], Butler et al. [22], Wu et al. [187], Murata and Kurokawa [115]
			44	Distributed Planning	Butler and Rus [24], Fitch and Butler [48]
			45	Hierarchical Planning	Yoshida et al. [197], Ababsa et al. [1]
			46	Reinforcement Learning	Varshavskaya [176], Varshavskaya et al. [177]
		Syn.	47	Messaging	Murata and Kurokawa [115]
			48	Locking	Lund et al. [104], Fitch and Butler [48]
Gait		Con.	49	Control Tables	Yim [189], Yim et al. [193], Stoy et al. [162,163], Yim et al. [192]
			50	Distributed Reinforcement Learning	Christensen et al. [37]
			51	Phase Automata	Yim et al. [194], Zhang et al. [204,203], Salemi et al. [141]

Table 7 (*continued*)

Operation	Approach		Method name	Employed by
		52	Central Pattern Generators	Kamimura et al. [83], Murata et al. [112], Kamimura et al. [84], Gonzalez-Gomez et al. [63], Kurokawa et al. [96], Möckel et al. [110], Sproewitz et al. [152], Pouya et al. [133], Moreno and Gomez [111]
		53	Neuroevolution	Stanley and Miikkulainen [153], Haasdijk et al. [71]
		54	Brain and Body Coevolution	Chocron [33], Jin and Meng [81], Meng et al. [106], Pouya et al. [132], Faíña et al. [45]
	Syn.	55	Master control	Yim [189]
		56	Leader–follower	Shen et al. [146]
		57	Hormone-based	Shen et al. [148,149], Hamann et al. [73], Moreno and Gomez [111]
		58	Synched Internal Clock	Stoy et al. [158]
		59	Delayed Signals	Stoy et al. [162]
Self-Assembly	Con.	60	Simulated Chemical Kinetics	Klavins [88], Klavins et al. [89], Miyashita et al. [109]
		61	Graph Grammar	Klavins et al. [90], Bishop et al. [11], Klavins [88], Pickem and Egerstedt [130]
		62	Finite State Machines	Tolley and Lipson [171], Wei et al. [182]
		63	Cellular Automata	Stoy [157,154]
		64	Gene Regularity Network	Bongard [14], Wei et al. [182]
Other Operations	Self-disassembly			Gilpin et al. [58,57,59]
	Self-adaptation			Yu et al. [201,199], Yu and Nagpal [200]
	Grasping			Bojinov et al. [12], Guan et al. [68], Yu and Nagpal [200], Gonzalez-Gomez et al. [62], Jin and Meng [81]
	Collective Actuation			Yim [189], Yim et al. [190], Stoy [155], Campbell and Pillai [26]
	Eveloping			Miao et al. [108]

come up with a list of about sixty open problems or improvements outlooks, which are believed to alleviate the existing shortcomings or inefficiencies of some solution methods, in case of being addressed in the future. In this subsection, these items are highlighted under the three most-researched operations of Self-reconfiguration, Flow, and Gait, categorized by their respective solution/control method.

7.1.1. Self-reconfiguration methods

Uninformed Searches: • Devising techniques to reduce the complexity of the C-space; • Developing efficient isomorphism checking techniques; • Proposing 'Anytime' algorithms to offer a path in C-space.

Informed Searches: • Proposing powerful and admissible heuristics; • Employing Anytime heuristic searches for solving SRP; • Developing low-cost isomorphism check.

Hierarchical Task Network (HTN): • Reconfiguration planning by HTN for 3D (non-planar) morphologies; • Utilizing distributed HTN for solving SRP [10].

Dynamic Programming: • Approximating the optimal self-reconfiguration cost using dynamic programming.

Divide and Conquer: • Identifying promising intermediate configurations.

Rapidly-exploring Random Tree: • Developing low-cost isomorphism checking; • Introducing more informed distance metrics for better avoidance from local minima.

Distributed and Local Search: • Developing coordination mechanisms; • Performing Connectivity checks in distributed manner; • Introducing mechanisms for preventing Deadlock situations (i.e., Hollow and Solid configurations).

Simulated Annealing: • Developing more informed distance metrics; • Devising novel definitions for neighboring configurations; • Developing 'Branching'-proof distance metrics.

Genetic Algorithm: • Developing uncertainty-proof GA-based controllers [97].

Random Action: • Utilizing low-cost connectedness checking techniques; • Implementing distributed Random Actions while the connectedness of the MRS is preserved.

Predesigned Rules: • Developing generic rules independent from type of modules; • Reusing predesigned rules in similar MRS problems; • Devising coordination mechanisms to resolve inconsistencies between rules.

Self-evolving Controllers: • Achieving simultaneous evolution of morphology and controller [33].

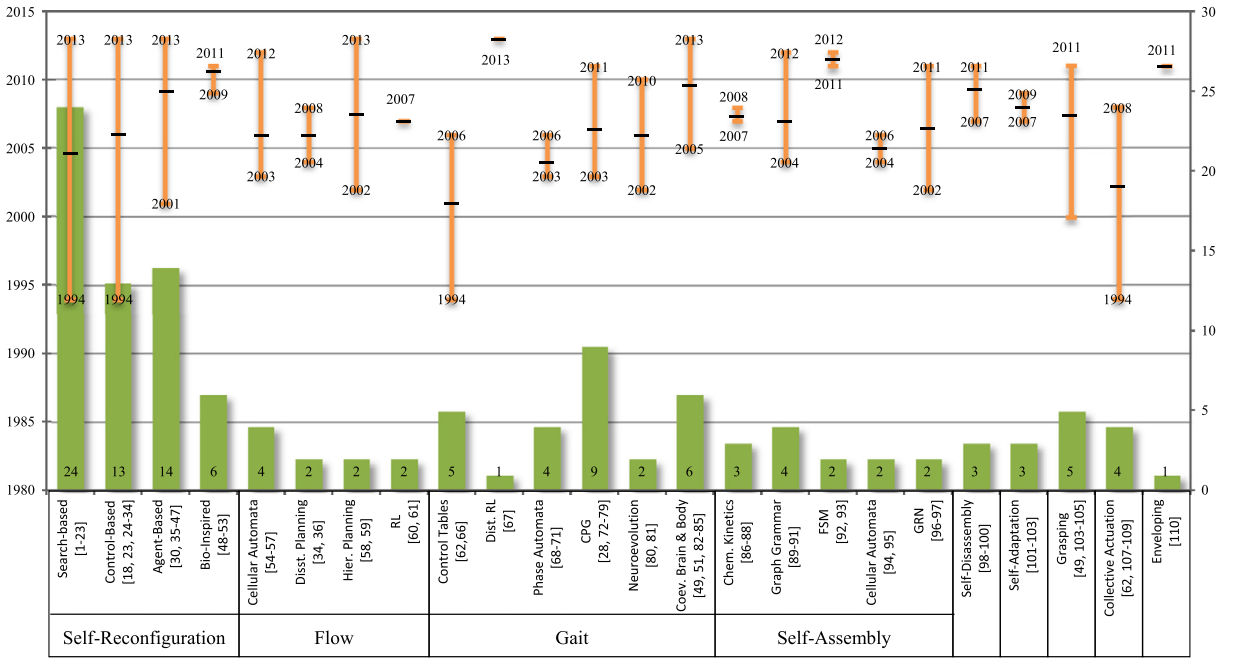
Gradient-Based: • Optimizing gradient propagation radius; • Considering motion constraints in gradient generation [158].

Recruitment: • Utilizing distributed breadth-first search for recruiting wandering modules close to seeds [158].

Cellular Automata: • Developing complex rule sets using learning techniques [22]; • Developing rule sets for complex tasks.

Markov Decision Process: • Devising mechanisms to prevent disconnection in modular robot body [48]; • Developing algorithms that efficiently find safe-to-move modules in sparse configurations.

Reinforcement-Learning: • Employing RL in MRS with many modules; • Learning cooperation in self-reconfiguration through learning joint actions; • Utilizing Multi-Agent Reinforcement Learning (MARL) methods in self-reconfiguration; • Considering world's partial observability in the learning process of modules.



LEGEND:

1. (Kotay and Rus 2000)
2. (Rus and Vona 2001)
3. (Fitch *et al.* 2005)
4. (Liu *et al.* 2007)
5. (Larkworthy and Ramamoorthy 2010)
6. (Pamecha and Chirikjian 1996)
7. (Pamecha *et al.* 1997)
8. (Asadpour *et al.* 2008)
9. (Asadpour *et al.* 2009)
10. (Bihlmaier *et al.* 2013)
11. (Nourollah and Razzazi 2011)
12. (Casal and Yim 1999)
13. (Ünsal *et al.* 2001)
14. (Bhat *et al.* 2006)
15. (Aloupis *et al.* 2008)
16. (Brandt 2006)
17. (Golestan *et al.* 2010)
18. (Christensen 2006)
19. (Christensen 2007)
20. (Chiang and Chirikjian 2001)
21. (Chen 1994)
22. (Lal *et al.* 2008)
23. (Dong and Li 2011)
24. (Murata *et al.* 1994)
25. (Brandt and Ostergaard 2004)
26. (Wong and Walter 2013)
27. (Zykov *et al.* 2007)
28. (Pouya *et al.* 2010)
29. (Hamann *et al.* 2010a)
30. (Stoy 2004)
31. (Stoy 2006b)
32. (Stoy and Nagpal 2007)
33. (Butler and Rus 2003)
34. (Butler and Rus 2004)
35. (Butler *et al.* 2001)
36. (Fitch and Butler 2008)
37. (Sadjadi *et al.* 2009)
38. (Fitch and McAllister 2010)
39. (Shiba *et al.* 2009)
40. (Ramaekers *et al.* 2011)
41. (Dutta *et al.* 2012)
42. (Dutta *et al.* 2013)
43. (Suzuki *et al.* 2009)
44. (Liu and Wu 2010)
45. (Rubenstein and Nagpal 2010)
46. (Hou and Shen 2013)
47. (Ko and Cheng 2010)
48. (Thenius *et al.* 2011)
49. (Jin and Meng 2011b)
50. (Meng and Jin 2011)
51. (Meng *et al.* 2011a)
52. (Umedachi *et al.* 2009)
53. (Jones *et al.* 2011)
54. (Xu *et al.* 2003)
55. (Butler *et al.* 2004)
56. (Wu *et al.* 2005)
57. (Murata and Kurokawa 2012c)
58. (Yoshida *et al.* 2002)
59. (Ababsa *et al.* 2013)
60. (Varshavskaya *et al.* 2007)
61. (Varshavskaya 2007)
62. (Yim 1994)
63. (Yim *et al.* 2001b)
64. (Stoy *et al.* 2002a)
65. (Stoy *et al.* 2002b)
66. (Yim *et al.* 2006)
67. (Christensen *et al.* 2013)
68. (Zhang *et al.* 2003)
69. (Yim *et al.* 2003)
70. (Y. Zhang *et al.* 2004)
71. (Salemi *et al.* 2006a)
72. (Kamimura *et al.* 2003)
73. (Murata *et al.* 2004)
74. (Kamimura *et al.* 2005)
75. (Gonzalez-Gomez *et al.* 2006)
76. (Möckel *et al.* 2006)
77. (Sproewitz *et al.* 2007)
78. (Kurokawa *et al.* 2006)
79. (Moreno and Gomez 2011)
80. (Stanley and Miikkulainen 2002)
81. (Haasdijk *et al.* 2010)
82. (von Haller *et al.* 2005)
83. (Chocron 2007)
84. (Pouya *et al.* 2011)
85. (Faíña *et al.* 2013)
86. (Klavins 2007)
87. (Klavins *et al.* 2007)
88. (Miyashita *et al.* 2008)
89. (Klavins *et al.* 2004)
90. (Bishop *et al.* 2005)
91. (Pickem and Egerstedt 2012)
92. (Tolley and Lipson 2011b)
93. (Wei *et al.* 2012)
94. (Kotay and Rus 2004)
95. (Stoy 2006c)
96. (Bongard 2002)
97. (Kernbach *et al.* 2011)
98. (Gilpin *et al.* 2007)
99. (Gilpin *et al.* 2010)
100. (Gilpin *et al.* 2011)
101. (Yu *et al.* 2007)
102. (Yu *et al.* 2008)
103. (Yu and Nagpal 2009)
104. (Bojinov *et al.* 2000)
105. (Y. Guan *et al.* 2007)
106. (Gonzalez-Gomez *et al.* 2010)
107. (Yim *et al.* 2001a)
108. (Stoy 2006a)
109. (Yu *et al.* 2008)
110. (Miao *et al.* 2011)

Fig. 14. Illustration of the number, range and of publication years, and the mean of publication years for the studied 110 research papers in each operation.

Game Theory: • Designing approximation algorithms for coalition formation in large-size agent sets [76].

Leader-Follower: • Enhancing leader election mechanisms; • Utilization of local coordination approaches such as Coordination Graphs [70].

Reconfigurable Software: • Developing methods for intelligent migration of agents to the neighboring hosts [91].

Bio-inspired solution methods: • Creating life-like structures and mechanisms (Artificial Life); • Adapting bio-inspired techniques to resolve embodiment issues; • Incorporating 'mechanical intelligence' into hardware design; • Solving replicator-related problems through bacteria and bio-molecules [144].

7.1.2. Flow methods

Cellular Automata: • Designing CA transition rules that consider external forces such as weight, friction, and inertia.

Distributed Planning: • Using various path-planning algorithms for generating an initial path for each module.

Hierarchical Planning: • Devising more complete and compact rule sets [197]; • Considering connection/disconnection costs in the plans and rule sets so that motion by hardware modules is optimized.

Reinforcement Learning: • Reducing explorations in the state-space through sharing experience of other learning modules [178]; • Considering world's partial observability in the learning process of modules.

7.1.3. Gait methods

Control Tables: • Developing distributed control tables immune from drifted local timers [158]; • Incorporating closed-loop control into individual actions [158].

Distributed Reinforcement Learning: • Speeding up convergence through new policies of balancing between exploration and exploitation.

Phase Automata: • Automatic design of phase automata through self-evolution of transition rules.

Central Pattern Generators (CPG): • Using sensory feedback to create robust CPG gait controllers [133].

Neuroevolution: • On-line adaptation of controllers for morphologies not known a priori [71].

Brain and Body Coevolution: • Reconstruction of gene expression patterns using computational models [81]; • Research on hardware requirements of Morphogenetic robotics such as growing materials, adaptable structures, adaptable sensors and actuators [81].

8. Perspective and conclusions

Since their inception in 1988, the main motivation for introducing modular robots has been to create versatile robots constructed from a number of building blocks called modules so that they can assume different morphologies to conform to changes in their tasks and environment. Challenges of enhancing MRS not only are limited to designing reliable, responsive, robust, and scalable hardware components, but also include developing solvers in the form of planning/control methods and algorithms that can contribute to versatility of such systems.

Considering the ultimate expectations from modular robots to perform various sophisticated tasks for real-world applications in different morphologies, the software aspect of modular robots needs to be extended beyond mere planners of self-reconfiguration, self-assembly, locomotion, etc. On the other hand, developing a truly universal solver that 'knows' how to perform various tasks in different morphologies is very challenging, and hence, currently impractical. Such a solver must be able to exhibit self-configuration, self-optimization, and self-evolving in its computation unit. For such future perspective, a viable resolution might be through implementing *Self-Adaptive Software* [123,140] in MRS, as a result of which behaviors or attributes of the software can be adapted during the runtime to the changes in the *self* (the whole body of the software) and the *context* (the environment). Such an effort can be seen in a recent work by Roehr et al. [136] which aims to realize adaptability and flexibility of modular robots through making changes in both hardware and software dimensions and take advantage of using configuration properties of software modules for generating more sophisticated reconfiguration approaches. This cutting edge vision in the software side of MRS has the potential to play key roles in shaping the future of modular robot algorithms, expanding the science and practice of MRS, and facilitating their deployment in real world, along with serving as an advanced research subject in the field.

In conclusion, this paper attempts to provide an up-to-date and comprehensive survey of algorithms and solution methods for performing nine basic operations of self-reconfiguration, flow, gait, self-assembly, self-disassembly, self-adaptation, grasping, collective actuation, and enveloping by modular robots. The methods' capabilities in modeling and simplifying problems, as well as their contribution toward developing controllers and synchronization mechanisms have been particularly emphasized. We identified 64 major abstraction, solution, control, and synchronization methods and algorithms that were developed in 125 research papers and employed in different operations by modular robotic systems, and through a consistent classification, studied their underlying theoretic and technical properties verified by supporting citations. In addition, the main challenges that these methods may confront while dealing with a particular problem are mentioned and their strengths and weaknesses are investigated, and a number of open problems or improvements outlooks are suggested to overcome their drawbacks. The paper has been written with the hope of providing a handy guide to researchers for studying the state-of-the-art of this exciting discipline and exploring promising research directions for the future.

References

[1] T. Ababsa, N. Djedi, Y. Duthen, S.C. Blanc, Decentralized approach to evolve the structure of metamorphic robots, in: IEEE Symposium on Artificial Life, ALIFE, 2013, pp. 74–81.

[2] G. Aloupis, S. Collette, M. Damian, E.D. Demaine, R. Flatland, S. Langerman, J. O'Rourke, S. Ramaswami, V. Sacristán, S. Wuhrer, Linear reconfiguration of cube-style modular robots, in: International Symposium on Algorithms and Computation, ISAAC, 2007, pp. 208–219.

[3] G. Aloupis, S. Collette, E. Demaine, S. Langerman, V. Sacristán, S. Wuhrer, Reconfiguration of cube-style modular robots using O(log n) parallel moves, in: International Symposium on Algorithms and Computation, ISAAC, 2008, pp. 342–353.

[4] C. Anderson, G. Theraulaz, J.-L. Deneubourg, Self-assemblages in insect societies, Insectes Soc. 49 (2) (2002) 99–110.

[5] M. Asadpour, M. Ashtiani, A. Sproewitz, A. Ijspeert, Graph signature for self-reconfiguration planning of modules with symmetry, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2009, pp. 5295–5300.

[6] M. Asadpour, A. Sproewitz, A. Billard, P. Dillenbourg, A.J. Ijspeert, Graph signature for self-reconfiguration planning, in: International Conference on Intelligent Robots and Systems, IROS, 2008, pp. 863–869.

[7] H.L. Ashe, J. Briscoe, The interpretation of morphogen gradients, Development 133 (3) (2006) 385–394.

[8] J. Baca, A. Yerpes, M. Ferre, J. Escalera, R. Aracil, Modelling of modular robot configurations using graph theory, in: Hybrid Artificial Intelligence Systems, 2008, pp. 649–656.

[9] P. Bhat, J. Kuffner, S. Goldstein, S. Srinivasa, Hierarchical motion planning for self-reconfigurable modular robots, in: International Conference on Intelligent Robots and Systems, IROS, 2006, pp. 886–891.

[10] A. Bihlmaier, L. Winkler, H. Worn, Automated planning as a new approach for the self-reconfiguration of mobile modular robots, in: 9th Workshop on Robot Motion and Control, RoMoCo, 2013, pp. 60–65.

[11] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, T. Nguyen, Programmable parts: a demonstration of the grammatical approach to self-organization, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2005, pp. 3684–3691.
[12] H. Bojinov, A. Casal, T. Hogg, Emergent structures in modular self-reconfigurable robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2000, pp. 1734–1741.
[13] M. Boncheva, D.A. Bruzewicz, G.M. Whitesides, Millimeter-scale self-assembly and its applications, Pure Appl. Chem. 75 (5) (2003) 621–630.
[14] J.C. Bongard, Evolving modular genetic regulatory networks, in: Congress on Evolutionary Computation, 2002, pp. 17–21.
[15] D. Brandt, Comparison of A* and RRT-connect motion planning techniques for self-reconfiguration planning, in: International Conference on Intelligent Robots and Systems, IROS, 2006, pp. 892–897.
[16] D. Brandt, E.H. Ostergaard, Behaviour subdivision and generalization of rules in rule based control of the ATRON self-reconfigurable robot, in: The International Symposium on Robotics and Automation, ISRA, 2004, pp. 67–74.
[17] R. Bridson, Fluid Simulation for Computer Graphics, AK Peters/CRC Press, 2008.
[18] A. Brunete, M. Hernando, E. Gambao, J. Torres, A. Castro-González, MDL: a module description language for chained heterogeneous modular robots, in: International Conference on Robotics and Biomimetics, 2011, pp. 2706–2711.
[19] L. Busoniu, R. Babuska, B. De Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. 38 (2) (2008) 156–172.
[20] Z. Butler, K. Kotay, D. Rus, K. Tomita, Cellular automata for decentralized control of self-reconfigurable robots, in: Workshop on Modular Robots International Conference on Robotics and Automation, ICRA, 2001.
[21] Z. Butler, K. Kotay, D. Rus, K. Tomita, Generic decentralized control for a class of self-reconfigurable robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2002, pp. 809–816.
[22] Z. Butler, K. Kotay, D. Rus, K. Tomita, Generic decentralized control for lattice-based self-reconfigurable robots, Int. J. Robot. Res. 23 (9) (2004) 919–937.
[23] Z. Butler, A.A. Rizzi, Distributed and cellular robots, in: Springer Handbook of Robotics, Springer-Verlag, New York, 2008, pp. 911–920.
[24] Z. Butler, D. Rus, Distributed motion planning for 3D modular robots with unit-compressible modules, in: Algorithmic Foundations of Robotics, 2004, pp. 435–452.
[25] Z. Butler, D. Rus, Distributed planning and control for modular robots with unit-compressible modules, Int. J. Robot. Res. 22 (9) (2003) 699–715.
[26] J. Campbell, P. Pillai, Collective actuation, Int. J. Robot. Res. 27 (3–4) (2008) 299–314.
[27] A. Casal, M. Yim, Self-reconfiguration planning for a class of modular robots, in: International Symposium on Intelligent Systems and Advanced Manufacturing (SPIE), 1999, pp. 246–257.
[28] I.M. Chen, Theory and applications of modular reconfigurable robotic systems, PhD thesis, California Inst. of Tech., Pasadena, CA, 1994.
[29] I.M. Chen, J.W. Burdick, Enumerating the non-isomorphic assembly configurations of modular robotic systems, Int. J. Robot. Res. 17 (7) (1998) 702–719.
[30] I.M. Chen, G.L. Yang, Automatic model generation for modular reconfigurable robot dynamics, J. Dyn. Syst. Meas. Control 120 (3) (1998) 346–352.
[31] C.J. Chiang, G.S. Chirikjian, Modular robot motion planning using similarity metrics, Auton. Robots 10 (1) (2001) 91–106.
[32] G.S. Chirikjian, Kinematics of a metamorphic robotic system, in: IEEE International Conference on Robotics and Automation, ICRA, 1994, pp. 449–455.
[33] O. Chocron, Evolving modular robots for rough terrain exploration, in: Mobile Robots: The Evolutionary Approach, Springer, 2007, pp. 23–46.
[34] D.J. Christensen, Evolution of shape-changing and self-repairing control for the ATRON self-reconfigurable robot, in: International Conference on Robotics and Automation, ICRA, 2006, pp. 2539–2545.
[35] D.J. Christensen, Experiments on fault-tolerant self-reconfiguration and emergent self-repair, in: IEEE Symposium on Artificial Life, ALIFE, 2007, pp. 355–361.
[36] D.J. Christensen, D. Brandt, K. Stoy, Towards artificial ATRON animals: scalable anatomy for self-reconfigurable robots, in: The RSS Workshop on Self-Reconfigurable Modular Robots, 2006.
[37] D.J. Christensen, U.P. Schultz, K. Stoy, A distributed and morphology-independent strategy for adaptive locomotion in self-reconfigurable modular robots, Robot. Auton. Syst. 61 (9) (2013) 1021–1035.
[38] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, MIT Press, Cambridge, 2001.
[39] F. Delcomyn, Neural basis of rhythmic behavior in animals, Science 210 (4469) (1980) 492–498.
[40] D.J. Dewey, M.P. Ashley-Rollman, M. De Rosa, S.C. Goldstein, T.C. Mowry, S.S. Srinivasa, P. Pillai, J. Campbell, Generalizing metamodules to simplify planning in modular robotic systems, in: International Conference on Intelligent Robots and Systems, IROS, 2008, pp. 1338–1345.
[41] B. Dong, Y. Li, Multi-objective-based configuration generation and optimization for reconfigurable modular robot, in: International Conference on Information Science and Technology, ICIST, 2011, pp. 1006–1010.
[42] A. Dutta, P. Dasgupta, J. Baca, C. Nelson, A block partitioning algorithm for modular robot reconfiguration under uncertainty, in: European Conference on Mobile Robots, ECMR, 2013, pp. 255–260.
[43] A. Dutta, P. Dasgupta, J. Baca, C. Nelson, A fast coalition structure search algorithm for modular robot reconfiguration planning under uncertainty, in: International Symposium on Distributed Autonomous Robotic Systems, DARS, 2012.
[44] H. Fahmy, D. Blostein, A survey of graph grammars: theory and applications, in: International Conference on Pattern Recognition, Conference B: Pattern Recognition Methodology and Systems, 1992, pp. 294–298.
[45] A. Faíña, F. Bellas, F. López-Peña, R.J. Duro, EDHMoR: evolutionary designer of heterogeneous modular robots, Eng. Appl. Artif. Intell. 26 (10) (2013) 2408–2423.
[46] Y. Fang, H. Zhang, X. Li, S. Chen, The mathematical model and control scheme of a four-legged robot based on GZ-I and note module, in: H. Ding, Z. Xiong, X. Zhu (Eds.), Intelligent Robotics and Applications (ICIRA), in: LNAI, vol. 6424, 2010, pp. 300–309.
[47] Y. Fei, C. Wang, Self-repairing algorithm of lattice-type self-reconfigurable modular robots, J. Intell. Robot. Syst. 75 (2) (2013) 193–203.
[48] R. Fitch, Z. Butler, Million module march: scalable locomotion for large self-reconfiguring robots, Int. J. Robot. Res. 27 (3–4) (2008) 331.
[49] R. Fitch, Z. Butler, D. Rus, Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2005, pp. 117–124.
[50] R. Fitch, Z. Butler, D. Rus, Reconfiguration planning for heterogeneous self-reconfiguring robots, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2003, pp. 2460–2467.
[51] R. Fitch, R. McAllister, Hierarchical planning for self-reconfiguring robots using module kinematics, in: 10th International Symposium on Distributed Autonomous Robotic Systems, DARS, 2010.
[52] R. Fitch, K. Stoy, S. Kernbach, R. Nagpal, W.M. Shen, Reconfigurable modular robotics, Robot. Auton. Syst. 62 (7) (2014) 943–944.
[53] A. Fuggetta, G.P. Picco, G. Vigna, Understanding code mobility, IEEE Trans. Softw. Eng. 24 (5) (1998) 342–361.
[54] T. Fukuda, Y. Kawauchi, Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator, in: IEEE International Conference on Robotics and Automation, ICRA, 1990, pp. 662–667.
[55] T. Fukuda, S. Nakagawa, Approach to dynamically reconfigurable robotic system, in: IEEE International Conference on Robotics and Automation, ICRA, 1988, pp. 1581–1586.
[56] M. Ghallab, D. Nau, P. Traverso, Automated Planning: Theory & Practice, Elsevier, 2004.
[57] K. Gilpin, A. Knaian, D. Rus, Robot pebbles: one centimeter modules for programmable matter through self-disassembly, in: IEEE International Conference on Robotics and Automation, ICRA, 2010, pp. 2485–2492.

[58] K. Gilpin, K. Kotay, D. Rus, I. Vasilescu, Miche: modular shape formation by self-disassembly, in: IEEE International Conference on Robotics and Automation, ICRA, 2007, pp. 2241–2247.
[59] K. Gilpin, K. Koyanagi, D. Rus, Making self-disassembling objects with multiple components in the robot pebbles system, in: IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 3614–3621.
[60] K. Gilpin, D. Rus, Modular robot systems from self-assembly to self-disassembly, IEEE Robot. Autom. Mag. 17 (3) (2010) 38–55.
[61] K. Golestan, M. Asadpour, H. Moradi, A new graph signature calculation method based on power centrality for modular robots, in: Distributed Autonomous Robotic Systems, in: Springer Tracts in Advanced Robotics, vol. 83, 2010, pp. 505–516.
[62] J. Gonzalez-Gomez, J. Gonzalez-Quijano, H. Zhang, M. Abderrahim, Toward the sense of touch in snake modular robots for search and rescue operations, in: IEEE Workshop on Modular Robots: The State of the Art, IEEE International Conference on Robotics and Automation, ICRA, 2010, pp. 63–68.
[63] J. Gonzalez-Gomez, H. Zhang, E. Boemo, J. Zhang, Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules, in: 9th International Conference on Climbing and Walking Robots, 2006.
[64] A.A. Gorbenko, V. Popov, On the optimal reconfiguration planning for modular self-reconfigurable DNA nanomechanical robots, Adv. Stud. Biol. 4 (2) (2012) 95–101.
[65] A.A. Gorbenko, V. Popov, Programming for modular reconfigurable robots, Program. Comput. Softw. 38 (1) (2012) 13–23.
[66] R. Groß, M. Dorigo, Self-assembly at the macroscopic scale, Proc. IEEE 96 (9) (2008) 1490–1508.
[67] E. Guan, Z. Fu, W. Yan, D. Jiang, Y. Zhao, Self-reconfiguration path planning design for M-lattice robot based on genetic algorithm, in: Intelligent Robotics and Applications, ICIRA, 2011, pp. 505–514.
[68] Y. Guan, L. Jiang, X. Zhang, Mechanical design and basic analysis of a modular robot with special climbing and manipulation functions, in: International Conference on Robotics and Biomimetics, ROBIO, 2007, pp. 502–507.
[69] M. Gudemann, F. Ortmeier, W. Reif, Safety and dependability analysis of self-adaptive systems, in: Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA, 2006, pp. 177–184.
[70] C. Guestrin, D. Koller, R. Parr, Multiagent planning with factored MDPs, in: NIPS, 2001, pp. 1523–1530.
[71] E. Haasdijk, A.A. Rusu, A. Eiben, HyperNEAT for locomotion control in modular robots, in: Evolvable Systems: From Biology to Hardware, Springer, 2010, pp. 169–180.
[72] H. Hamann, J. Stradner, T. Schmickl, K. Crailsheim, Artificial hormone reaction networks: towards higher evolvability in evolutionary multi-modular robotics, arXiv:1011.3912, 2010.
[73] H. Hamann, J. Stradner, T. Schmickl, K. Crailsheim, A hormone-based controller for evolutionary multi-modular robotics: from single modules to gait learning, in: IEEE Congress on Evolutionary Computation, CEC, 2010, pp. 1–8.
[74] J. Harrison, C. Vo, J.M. Lien, Scalable and robust shepherding via deformable shapes, in: R. Boulic, Y. Chrysanthou, T. Komura (Eds.), Motion in Games, in: LNCS, vol. 6459, Springer, Berlin/Heidelberg, 2010, pp. 218–229.
[75] K. Hosokawa, I. Shimoyama, H. Miura, Dynamics of self-assembling systems: analogy with chemical kinetics, Artif. Life Robot. 1 (4) (1994) 413–427.
[76] S. Hossain, C.A. Nelson, P. Dasgupta, Hardware design and testing of ModRED: a modular self-reconfigurable robot system, in: J.S. Dai, M. Zoppi, X. Kong (Eds.), Advances in Reconfigurable Mechanisms and Robots, Springer, London, 2012, pp. 515–523.
[77] F. Hou, W.M. Shen, Graph-based optimal reconfiguration planning for self-reconfigurable robots, Robot. Auton. Syst. 62 (7) (2013) 1047–1059.
[78] F. Hou, W.M. Shen, On the complexity of optimal reconfiguration planning for modular reconfigurable robots, in: International Conference on Robotics and Automation, ICRA, 2010, pp. 2791–2796.
[79] X. Jiang, H. Bunke, Optimal quadratic-time isomorphism of ordered graphs, Pattern Recognit. 32 (7) (1999) 1273–1283.
[80] Y. Jin, Y. Meng, Morphogenetic robotics—an evolutionary developmental approach to morphological and neural self-organization of robotic systems, in: Bio-Inspired Self-Organizing Robotic Systems, Springer, 2011, pp. 3–23.
[81] Y. Jin, Y. Meng, Morphogenetic robotics: an emerging new field in developmental robotics, IEEE Trans. Syst. Man Cybern., Part C, Appl. Rev. 41 (2) (2011) 145–160.
[82] J. Jones, S. Tsuda, A. Adamatzky, Towards Physarum robots, in: Bio-Inspired Self-Organizing Robotic Systems, Springer, 2011, pp. 215–251.
[83] A. Kamimura, H. Kurokawa, E. Toshida, K. Tomita, S. Murata, S. Kokaji, Automatic locomotion pattern generation for modular robots, in: International Conference on Robotics and Automation, ICRA, 2003, pp. 714–720.
[84] A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, S. Murata, Distributed adaptive locomotion by a modular robotic system, M-TRAN II, in: International Conference on Intelligent Robots and Systems, IROS, 2005, pp. 2370–2377.
[85] L.E. Kavraki, P. Svestka, J.-C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (4) (1996) 566–580.
[86] S. Kernbach, B. Girault, O. Kernbach, On self-optimized self-assembling of heterogeneous multi-robot organisms, in: Y. Meng, Y. Jin (Eds.), Bio-Inspired Self-Organizing Robotic Systems, Springer, 2011, pp. 123–141.
[87] S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, D. Laneri, L. Winkler, T. Schmickl, R. Thenius, Symbiotic robot organisms: REPLICATOR and SYMBRION projects, in: The 8th Workshop on Performance Metrics for Intelligent Systems, PerMIS, 2008, pp. 62–69.
[88] E. Klavins, Programmable self-assembly, IEEE Control Syst. Mag. 27 (4) (2007) 43–56.
[89] E. Klavins, S. Burden, N. Napp, Optimal rules for programmed stochastic self-assembly, in: Robotics: Science and Systems II, 2007, pp. 9–16.
[90] E. Klavins, R. Ghrist, D. Lipsky, Graph grammars for self-assembling robotic systems, in: IEEE International Conference on Robotics and Automation, ICRA, 2004, pp. 5293–5300.
[91] D. Ko, H.H. Cheng, Reconfigurable software for reconfigurable modular robots, in: Proc. of ICRA 2010 Workshop Modular Robots: State of the Art, 2010, p. 100.
[92] K. Kotay, D. Rus, Algorithms for self-reconfiguring molecule motion planning, in: International Conference on Intelligent Robots and Systems, IROS, 2000, pp. 2184–2193.
[93] K. Kotay, D. Rus, Generic distributed assembly and repair algorithms for self-reconfiguring robots, in: International Conference on Intelligent Robots and Systems, IROS, 2004, pp. 2362–2369.
[94] H. Kurokawa, A. Kamimura, E. Yoshida, K. Tomita, S. Kokaji, S. Murata, M-TRAN II: metamorphosis from a four-legged walker to a caterpillar, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2003, pp. 2454–2459.
[95] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, S. Murata, Distributed self-reconfiguration of M-TRAN III modular robotic system, Int. J. Robot. Res. 27 (3–4) (2008) 373–386.
[96] H. Kurokawa, E. Yoshida, K. Tomita, A. Kamimura, S. Murata, S. Kokaji, Self-reconfigurable M-TRAN structures and walker generation, Robot. Auton. Syst. 54 (2) (2006) 142–149.
[97] S. Lal, K. Yamada, S. Endo, Emergent motion characteristics of a modular robot through genetic algorithm, in: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, in: LNCS, vol. 5227, 2008, pp. 225–234.
[98] T. Larkworthy, S. Ramamoorthy, An efficient algorithm for self-reconfiguration planning in a modular robot, in: International Conference on Robotics and Automation, ICRA, 2010, pp. 5139–5146.
[99] S.M. LaValle, Rapidly-exploring random trees: a new tool for path planning, Tech. report, Computer Science Department, Iowa State University, 1998.
[100] H. Lipson, J.B. Pollack, Automatic design and manufacture of robotic lifeforms, Nature 406 (6799) (2000) 974–978.

[101] J.G. Liu, Y. Wang, S. Ma, B. Li, Analysis of stairs-climbing ability for a tracked reconfigurable modular robot, in: Security and Rescue Robotics Workshop, International Safety, 2005, pp. 36–41.
[102] J.G. Liu, Y.C. Wang, B. Li, S.G. Ma, D.L. Tan, Center-configuration selection technique for the reconfigurable modular robot, Sci. China, Ser. F 50 (5) (2007) 697–710.
[103] J.G. Liu, J. Wu, Multiagent Robotic Systems, CRC Press, 2010.
[104] H.H. Lund, R.L. Larsen, E.H. Østergaard, Distributed control in self-reconfigurable robots, in: Evolvable Systems: From Biology to Hardware, Springer, 2003, pp. 296–307.
[105] Y. Meng, Y. Jin, Morphogenetic self-reconfiguration of modular robots, in: Bio-Inspired Self-Organizing Robotic Systems, Springer, 2011, pp. 143–171.
[106] Y. Meng, Y. Zhang, Y. Jin, Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model, IEEE Comput. Intell. Mag. 6 (1) (2011) 43–54.
[107] Y. Meng, Y. Zhang, A. Sampath, Y. Jin, B. Sendhoff, Cross-ball: a new morphogenetic self-reconfigurable modular robot, in: IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 267–272.
[108] Y. Miao, G. Yan, Z. Lin, A distributed reconfiguration strategy for target enveloping with hexagonal metamorphic modules, in: IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 4804–4809.
[109] S. Miyashita, M. Kessler, M. Lungarella, How morphology affects self-assembly in a stochastic modular robot, in: IEEE International Conference on Robotics and Automation, ICRA, 2008, pp. 3533–3538.
[110] R. Möckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, A. Ijspeert, YaMoR and Bluemove – an autonomous modular robot with bluetooth interface for exploring adaptive locomotion, in: M.O. Tokhi, G.S. Virk, M.A. Hossain (Eds.), Climbing and Walking Robots, Springer, Berlin/Heidelberg, 2006, pp. 685–692.
[111] R. Moreno, J. Gomez, Central pattern generators and hormone inspired messages: a hybrid control strategy to implement motor primitives on chain type modular reconfigurable robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2011, pp. 1014–1019.
[112] S. Murata, A. Kamimura, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, Self-reconfigurable robots: platforms for emerging functionality, in: Embodied Artificial Intelligence, Springer, 2004, pp. 312–330.
[113] S. Murata, H. Kurokawa, Artificial self-assembly and self-repair, in: Self-Organizing Robots, in: STAR, vol. 77, Springer, Berlin/Heidelberg, 2012, pp. 77–103.
[114] S. Murata, H. Kurokawa, Prototypes of self-organizing robots, in: Self-Organizing Robots, Springer, 2012, pp. 105–130.
[115] S. Murata, H. Kurokawa, Robotic metamorphosis, in: Self-Organizing Robots, Springer, 2012, pp. 131–171.
[116] S. Murata, H. Kurokawa, Self-organization of biological systems, in: Self-Organizing Robots, Springer, 2012, pp. 19–35.
[117] S. Murata, H. Kurokawa, Self-reconfigurable robots: shape-changing cellular robots, IEEE Robot. Autom. Mag. 14 (1) (2007) 71–78.
[118] S. Murata, H. Kurokawa, S. Kokaji, Self-assembling machine, in: IEEE International Conference on Robotics and Automation, ICRA, 1994, pp. 441–448.
[119] L. Murray, Fault tolerant morphogenesis in self-reconfigurable modular robotic systems, Thesis, University of York, 2013.
[120] M. Nagl, A tutorial and bibliographical survey on graph grammars, in: Graph-Grammars and Their Application to Computer Science and Biology, 1979, pp. 70–126.
[121] D.S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, F. Yaman, SHOP2: an HTN planning system, J. Artif. Intell. Res. 20 (2003) 379–404.
[122] A. Nourollah, M. Razzazi, Minimum cost open chain reconfiguration, Discrete Appl. Math. 159 (14) (2011) 1418–1424.
[123] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, A.L. Wolf, An architecture-based approach to self-adaptive software, IEEE Intell. Syst. Appl. 14 (3) (1999) 54–62.
[124] L. Paez, K. Melo, C. Parra, Center of mass displacements using rolling gaits for modular robots on the outside of pipes, in: Colombian Conference on Automatic Control and Industry Applications, LARC, 2011, pp. 1–6.
[125] A. Pamecha, G. Chirikjian, A useful metric for modular robot motion planning, in: International Conference on Robotics and Automation, ICRA, 1996.
[126] A. Pamecha, I. Ebert-Uphoff, G. Chirikjian, Useful metrics for modular robot motion planning, IEEE J. Robot. Autom. 13 (4) (1997) 531–545.
[127] M. Park, S. Chitta, A. Teichman, M. Yim, Automatic configuration recognition methods in modular robots, Int. J. Robot. Res. 27 (3–4) (2008) 403.
[128] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Longman Publishing Co., 1984.
[129] R. Pfeifer, J. Bongard, How the Body Shapes the Way We Think: A New View of Intelligence, MIT Press, 2007.
[130] D. Pickem, M. Egerstedt, Self-reconfiguration using graph grammars for modular robotics, in: Conference on Analysis and Design of Hybrid Systems, IFAC, 2012.
[131] D. Pickem, M. Egerstedt, J.S. Shamma, Complete heterogeneous self-reconfiguration: deadlock avoidance using hole-free assemblies, in: Estimation and Control of Networked Systems, 2013, pp. 404–410.
[132] S. Pouya, E. Aydin, R. Möckel, A.J. Ijspeert, Locomotion gait optimization for modular robots; coevolving morphology and control, Proc. Comput. Sci. 7 (2011) 320–322.
[133] S. Pouya, J. Van Den Kieboom, A. Spröwitz, A. Ijspeert, Automatic gait generation in modular robots: to oscillate or to rotate? That is the question, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2010, pp. 514–520.
[134] Z. Ramaekers, R. Dasgupta, V. Ufimtsev, S. Hossain, C. Nelson, Self-reconfiguration in modular robots using coalition games with uncertainty, in: Automated Action Planning for Autonomous Mobile Robots, 2011.
[135] D. Ray, A Game-Theoretic Perspective on Coalition Formation, Oxford University Press, 2008.
[136] T.M. Roehr, F. Cordes, F. Kirchner, Reconfigurable Integrated Multirobot Exploration System (RIMRES): heterogeneous modular reconfigurable robots for space exploration, J. Field Robot. 31 (1) (2014) 3–34.
[137] M. Rubenstein, R. Nagpal, Kilobot: a robotic module for demonstrating behaviors in a large scale (2^{10} units) collective, in: IEEE Workshop on Modular Robots: The State of the Art, IEEE International Conference on Robotics and Automation, ICRA, 2010.
[138] D. Rus, M. Vona, Crystalline robots: self-reconfiguration with compressible unit modules, Auton. Robots 10 (1) (2001) 107–124.
[139] H. Sadjadi, M. Al-Jarrah, K. Assaleh, Morphology for planar hexagonal modular self-reconfigurable robotic systems, in: International Symposium on Mechatronics and Its Applications, ISMA, 2009, pp. 1–6.
[140] M. Salehie, L. Tahvildari, Self-adaptive software: landscape and research challenges, ACM Trans. Auton. Adapt. Syst. 4 (2) (2009) 14.
[141] B. Salemi, M. Moll, W.M. Shen, SUPERBOT: a deployable, multi-functional, and modular self-reconfigurable robotic system, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2006, pp. 3636–3641.
[142] B. Salemi, M. Moll, W.M. Shen, SUPERBOT: a deployable, multi-functional, and modular self-reconfigurable robotic system, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2006, pp. 3636–3641.
[143] H. Schmeck, C. Mller-Schloer, E. Cakar, M. Mnif, U. Richter, Adaptivity and self-organization in organic computing systems, ACM Trans. Auton. Adapt. Syst. 5 (3) (2010) 1–32.
[144] T. Schmickl, How to engineer robotic organisms and swarms?, in: Bio-Inspired Self-Organizing Robotic Systems, Springer, 2011, pp. 25–52.
[145] U.P. Schultz, Distributed control diffusion: towards a flexible programming paradigm for modular robots, in: The 1st International Conference on Robot Communication and Coordination, 2007, p. 15.
[146] W.M. Shen, H.C. Chiu, M. Rubenstein, B. Salemi, Rolling and climbing by the multifunctional superbot reconfigurable robotic system, in: Proceedings of the Space Technology International Forum, Albuquerque, New Mexico, 2008, pp. 839–848.

[147] W.M. Shen, M. Krivokon, H.C.H. Chiu, J. Everist, M. Rubenstein, J. Venkatesh, Multimode locomotion via SuperBot robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2006, pp. 2552–2557.
[148] W.M. Shen, Y. Lu, P. Will, Hormone-based control for self-reconfigurable robots, in: The Fourth International Conference on Autonomous Agents, 2000, pp. 1–8.
[149] W.M. Shen, B. Salemi, P. Will, Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots, IEEE Robot. Autom. Mag. 18 (5) (2002) 700–712.
[150] S. Shiba, M. Uchida, A. Nozawa, H. Asano, H. Onogaki, T. Mizuno, H. Ide, S. Yokoyama, Autonomous reconfiguration of robot shape by using Q-learning, Artif. Life Robot. 14 (2) (2009) 213–218.
[151] K. Sims, Evolving 3D morphology and behavior by competition, Artif. Life Robot. 1 (4) (1994) 353–372.
[152] A. Sproewitz, R. Moeckel, J. Maye, M. Asadpour, A. Ijspeert, Adaptive locomotion control in modular robotics, in: Workshop on Self-Reconfigurable Robots/Systems and Applications International Conference on Intelligent Robots and Systems, IROS, 2007, pp. 81–84.
[153] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[154] K. Stoy, Controlling self-reconfiguration using cellular automata and gradients, in: The 8th International Conference on Intelligent Autonomous Systems, IAS-8, 2004, pp. 693–702.
[155] K. Stoy, The deformatron robot: a biologically inspired homogeneous modular robot, in: IEEE International Conference on Robotics and Automation, ICRA, 2006, pp. 2527–2531.
[156] K. Stoy, How to construct dense objects with self-recondfigurable robots, in: European Robotics Symposium, 2006, pp. 27–37.
[157] K. Stoy, Using cellular automata and gradients to control self-reconfiguration, Robot. Auton. Syst. 54 (2) (2006) 135–141.
[158] K. Stoy, D. Brandt, D.J. Christensen, Self-Reconfigurable Robots: An Introduction, The MIT Press, 2010.
[159] K. Stoy, H. Kurokawa, Current topics in classic self-reconfigurable robot research, in: IEEE Self-Reconfigurable Robotics Workshop, IEEE International Conference on Intelligent Robots and Systems, IROS, 2011.
[160] K. Stoy, R. Nagpal, Self-reconfiguration using directed growth, in: Distributed Autonomous Robotic Systems, vol. 6, Springer, 2007, pp. 3–12.
[161] K. Stoy, R. Nagpal, Self-repair through scale independent self-reconfiguration, in: International Conference on Intelligent Robots and Systems, IROS, 2004, pp. 2062–2067.
[162] K. Stoy, W.M. Shen, P. Will, Global locomotion from local interaction in self-reconfigurable robots, in: 7th International Conference on Intelligent Autonomous Systems, 2002.
[163] K. Stoy, W.M. Shen, P. Will, How to make a self-reconfigurable robot run, in: The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS): Part 2, 2002, pp. 813–820.
[164] R. Subramanian, M. Masek, C.S. Lee, Connectivity check for modular self reconfigurable robots, in: IEEE Region 10 Conference, TENCON, 2013, pp. 1–4.
[165] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[166] Y. Suzuki, N. Inou, M. Koseki, H. Kimura, Reconfigurable modular robots adaptively transforming a mechanical structure (numerical expression of transformation criteria of "CHOBIE II" and motion experiments), in: Distributed Autonomous Robotic Systems, vol. 8, 2009, pp. 393–403.
[167] R. Thenius, M. Dauschan, T. Schmickl, K. Crailsheim, Regenerative abilities in modular robots using virtual embryogenesis, in: Adaptive and Intelligent Systems, Springer, 2011, pp. 227–237.
[168] M.T. Tolley, M. Krishnan, D. Erickson, H. Lipson, Dynamically programmable fluidic assembly, Appl. Phys. Lett. 93 (25) (2008) 254105.
[169] M.T. Tolley, H. Lipson, Fluidic manipulation for scalable stochastic 3d assembly of modular robots, in: IEEE International Conference on Robotics and Automation, ICRA, 2010, pp. 2473–2478.
[170] M.T. Tolley, H. Lipson, On-line assembly planning for stochastically reconfigurable systems, Int. J. Robot. Res. 30 (13) (2011) 1566–1584.
[171] M.T. Tolley, H. Lipson, Programmable 3D stochastic fluidic assembly of cm-scale modules, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2011, pp. 4366–4371.
[172] E. Tuci, R. Groß, V. Trianni, F. Mondada, M. Bonani, M. Dorigo, Cooperation through self-assembly in multi-robot systems, ACM Trans. Auton. Adapt. Syst. 1 (2) (2006) 115–150.
[173] T. Umedachi, T. Kitamura, K. Takeda, T. Nakagaki, R. Kobayashi, A. Ishiguro, A modular robot driven by protoplasmic streaming, in: Distributed Autonomous Robotic Systems, vol. 8, Springer, 2009, pp. 193–202.
[174] C. Ünsal, H. Kiliççöte, P.K. Khosla, A modular self-reconfigurable bipartite robotic system: implementation and motion planning, Auton. Robots 10 (1) (2001) 23–40.
[175] K.S. Van Hornweder, A chronological survey of modular self-reconfigurable robots, Tech. report, Department of Electrical Engineering & Computer Science University of Tennessee, Knoxville, 2011.
[176] P. Varshavskaya, Distributed reinforcement learning for self-reconfiguring modular robots, Ph.D. thesis, Massachusetts Institute of Technology, 2007.
[177] P. Varshavskaya, L. Kaelbling, D. Rus, Automated design of adaptive controllers for modular robots using reinforcement learning, Int. J. Robot. Res. 27 (2007) 3–4, Special Issue on Self-Reconfigurable Modular Robots.
[178] P. Varshavskaya, L.P. Kaelbling, D. Rus, Efficient distributed reinforcement learning through agreement, in: Distributed Autonomous Robotic Systems, Springer, 2009, pp. 367–378.
[179] W.E. Vesely, F.F. Goldberg, N.H. Roberts, D.F. Haasl, Fault Tree Handbook, U.S. Nuclear Regulatory Commission, Washington, DC, 1981.
[180] B. Von Haller, A. Ijspeert, D. Floreano, Co-evolution of structures and controllers for Neubot underwater modular robots, in: Advances in Artificial Life, Springer, 2005, pp. 189–199.
[181] M. Vona, D. Rus, A physical implementation of the self-reconfiguring crystalline robot, in: IEEE International Conference on Robotics and Automation, ICRA, 2000, pp. 1726–1733.
[182] H. Wei, H. Li, J. Tan, T. Wang, Self-assembly control and experiments in swarm modular robots, Sci. China, Technol. Sci. 55 (4) (2012) 1118–1131.
[183] P.J. White, V. Zykov, J. Bongard, H. Lipson, Three-dimensional stochastic reconfiguration of modular robots, in: Robotics: Science and Systems, 2005, pp. 161–168.
[184] G.M. Whitesides, M. Boncheva, Beyond molecules: self-assembly of mesoscopic and macroscopic components, Proc. Natl. Acad. Sci. USA 99 (8) (2002) 4769–4774.
[185] G.M. Whitesides, B. Grzybowski, Self-assembly at all scales, Science 295 (5564) (2002) 2418–2421.
[186] S. Wong, J. Walter, Deterministic distributed algorithm for self-reconfiguration of modular robots from arbitrary to straight chain configurations, in: IEEE International Conference on Robotics and Automation, ICRA, 2013, pp. 537–543.
[187] Q. Wu, Y. Wang, G. Cao, Y. Fei, Locomotion control of distributed self-reconfigurable robot based on cellular automata, in: ICIC 2005, in: LNCS, vol. 3645, 2005, pp. 179–188.
[188] W. Xu, S.G. Wang, A.L. Wang, G.B. Wang, Towards an efficient self-organizing reconfiguration method for self-reconfigurable robots, J. Intell. Robot. Syst. 37 (4) (2003) 415–425.
[189] M. Yim, Locomotion with a unit-modular reconfigurable robot, Ph.D. thesis, Stanford University, 1994.
[190] M. Yim, D. Duff, Y. Zhang, Closed-chain motion with large mechanical advantage, in: International Conference on Intelligent Robots and Systems (IROS), 2001, pp. 318–323.
[191] M. Yim, D.G. Duff, K. Roufas, Modular reconfigurable robots, an approach to urban search and rescue, in: 1st International Workshop on Human Welfare Robotics Systems, HWRS, 2000, pp. 69–76.

[192] M. Yim, C. Eldershaw, Y. Zhang, D. Duff, Limbless conforming gaits with modular robots, in: M. Ang, O. Khatib (Eds.), Experimental Robotics, vol. IX, Springer, Berlin/Heidelberg, 2006, pp. 459–468.
[193] M. Yim, S. Homans, K. Roufas, Climbing with snake-like robots, in: IFAC Workshop on Mobile Robot Technology, 2001, pp. 21–22.
[194] M. Yim, K. Roufas, D. Duff, Y. Zhang, C. Eldershaw, S. Homans, Modular reconfigurable robots in space applications, Auton. Robots 14 (2) (2003) 225–237.
[195] M. Yim, W.M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G.S. Chirikjian, Modular self-reconfigurable robot systems [grand challenges of robotics], IEEE Robot. Autom. Mag. 14 (1) (2007) 43–52.
[196] M. Yim, Y. Zhang, J. Lamping, E. Mao, Distributed control for 3D metamorphosis, Auton. Robots 10 (1) (2001) 41–56.
[197] E. Yoshida, S. Matura, A. Kamimura, K. Tomita, H. Kurokawa, S. Kokaji, A self-reconfigurable modular robot: reconfiguration planning and experiments, Int. J. Robot. Res. 21 (10–11) (2002) 903–915.
[198] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, S. Kokaji, A motion planning method for a self-reconfigurable modular robot, in: International Conference on Intelligent Robots and Systems, IROS, 2001, pp. 590–597.
[199] C.H. Yu, K. Haller, D. Ingber, R. Nagpal, Morpho: a self-deformable modular robot inspired by cellular structure, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2008, pp. 3571–3578.
[200] C.H. Yu, R. Nagpal, Self-adapting modular robotics: a generalized distributed consensus framework, in: IEEE International Conference on Robotics and Automation, ICRA, 2009, pp. 1881–1888.
[201] C.H. Yu, F.X. Willems, D. Ingber, R. Nagpal, Self-organization of environmentally-adaptive shapes on a modular robot, in: IEEE International Conference on Intelligent Robots and Systems, IROS, 2007, pp. 2353–2360.
[202] L. Zhang, J. Zhao, H.G. Cai, A substructure based motion planning method for a modular self-reconfigurable robot, in: International Workshop on Robot Motion and Control, RoMoCo, 2004, pp. 371–376.
[203] Y. Zhang, A. Golovinsky, M. Yim, C. Eldershaw, An XML-based scripting language for chain-type modular robotic systems, in: IEEE Conference on Intelligent Autonomous Systems, 2004.
[204] Y. Zhang, M. Yim, C. Eldershaw, D. Duff, K. Roufas, Phase automata: a programming model of locomotion gaits for scalable chain-type modular robots, in: International Conference on Intelligent Robots and Systems, IROS, 2003, pp. 2442–2447.
[205] V. Zykov, E. Mytilinaios, M. Desnoyer, H. Lipson, Evolved and designed self-reproducing modular robotics, IEEE Trans. Robot. 23 (2) (2007) 308–319.