A survey of autonomous self-reconfiguration methods for robot-based programmable matter

Pierre Thalamy, Benoit Piranda, Julien Bourgeois

A Survey of Autonomous Self-Reconfiguration Methods for Robot-Based Programmable Matter

Pierre Thalamy[a,*], Benoît Piranda[a], Julien Bourgeois[a]

[a] *Univ. Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS*
1 cours Leprince-Ringuet, 25200, Montbéliard, France.

Abstract

While researchers envision exciting applications for metamorphic systems like programmable matter, current solutions to the shape formation problem are still a long way from meeting their requirements. To dive deeper into this issue, we propose an extensive survey of the current state of the art of self-reconfiguration algorithms and underlying models in modular robotic and self-organizing particle systems. We identify three approaches for solving this problem and we compare the different solutions using a synoptic graphical representation. We then close this survey by confronting existing methods to our vision of programmable matter, and by discussing a number of future research directions that would bring us closer to making it a reality.

Keywords: Self-Reconfiguration, Modular Robots, Programmable Matter, Distributed Algorithms, Self-Organizing Particle Systems

1. Introduction

Modular robots are robotic systems composed of interconnected individual electro-mechanical modules that can rearrange in order to best adapt to their task-environment or recover from failures. Their promise is to realize robotic systems that are more versatile, affordable, and robust than their conventional counterparts, at the cost of a probable reduced efficacy for specific task. This concept was first introduced in the late 1980's as

*Corresponding author.
Email addresses: pthalamy@femto-st.fr (Pierre Thalamy), bpiranda@femto-st.fr (Benoît Piranda), jbourgeois@femto-st.fr (Julien Bourgeois)

cellular robotic systems by T. Fukuda, later physically realized in the CEBOT modular robot by Fukuda and Kawauchi (1990). Since then, the field has been renamed *modular robotics*, and various robotic architectures have been proposed (Ahmadzadeh et al., 2016). Modular robots differ from traditional swarm robotic systems by the fact that all individual robots in a system must remain connected to each other at all times, whereas swarm robots are usually mobile robots with full autonomy — albeit some of these systems are sometimes referred to as *mobile* modular robotic systems (e.g., Kilobot (Rubenstein et al., 2012)).

The first type of architecture is *chain-type* modular robots, where chains of modules make up the structure of the robots, with the modules always arranged in a tree-like fashion. This architecture has been extensively studied over the years, and the control problem for such systems is now largely well-understood. The other main architecture type is *lattice-type* modular robots, composed of an ordered arrangement of modules, residing on a regular structure named a *lattice*. There exist a number of lattice structures, each based on a particular geometry of their cells, and differing by their packing-density, number of dimensions, and number of neighboring positions at a given location (Naz et al., 2018; Piranda and Bourgeois, 2018). In contrast with chain-type modular robots, there are still many open problems regarding lattice-based systems, especially on the control side. This will be the topic of this paper, where the state of the art in the self-reconfiguration of 3D lattice-based metamorphic systems will be brought into focus. Finally, some modular robots can both exhibit the features of lattice-type and chain-type modular robots, and are commonly referred to as *hybrid*.

Independently of their architecture, the most striking feature of modular robots is their ability to *reconfigure* their morphology, a process called *self-reconfiguration* (SR). More specifically, self-reconfiguration is the process undergone by a modular robot whereby its initial arrangement of modules I (and therefore initial shape, also referred to as *configuration*) is altered into a goal configuration G. Self-reconfiguration is effectively realized by the self-organization of individual modules constituting the robot, through motion and communication. Self-reconfiguration differs from *self-assembly* (Tucci et al., 2018) by having modules occupying the structure in an orderly manner throughout the reconfiguration process. Although there are several scenarios in which self-reconfiguration can turn useful, such as locomotion (Fitch and Butler, 2007), task adaptation (Bojinov et al., 2000), collective mechanical actuation (Holobut et al., 2014; Campbell and Pillai,

2008), or self-repair (Stoy and Nagpal, 2004), our analysis will focus solely on the problem of shape formation. While great efforts have been produced to solve the *Self-Reconfiguration Problem* (SRP) since its inception in the mid-1990's, we want to make the point that realistic solutions having all the desirable properties of a self-reconfiguration algorithm have yet to be introduced.

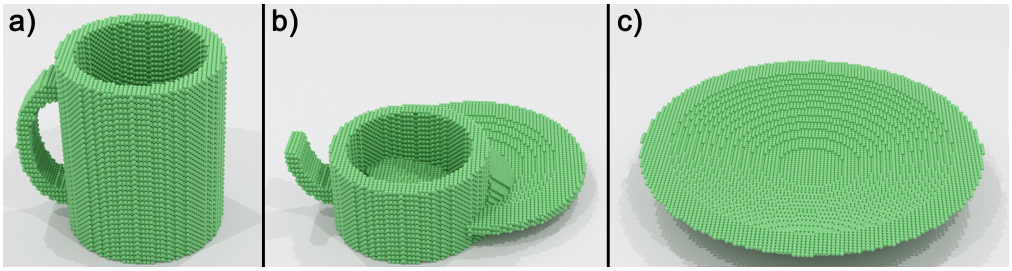

Figure 1: Sample self-reconfiguration of about 38,500 3D Catom modules from a cup into a plate. a) *Cup* initial configuration; b) An intermediate configuration from the self-reconfiguration process; c) *Plate* goal configuration.

A promising application of metamorphic systems is to achieve *Programmable Matter* (PM), matter that is able to dynamically alter its state—usually its shape— as a response to internal or external stimuli. Though many technologies could potentially be used as the basis for programmable matter, building PM using modular self-reconfigurable robots (MSR) represents the most promising endeavor, as it is the only technology that manifests all four desired properties: evolutivity, programmability, autonomy, and interactivity (Bourgeois et al., 2016). The self-reconfiguration problem evidently stands as one of the major foundational issues of PM. In this context, algorithmic solutions are required to exhibit some particular properties that will be discussed further on through an analysis of the state of the art of SR juxtaposed with the expectations of PM.

Self-reconfiguration planning is a hard problem, as traditional search methods are ineffective due to the size of the configuration space increasing exponentially with the number of modules in the system. It has been proved to be NP-complete for chain-type MSR by reduction to 3-PARTITION (Hou and Shen, 2010) and PSAT (Gorbenko and Popov, 2012), and is suspected to be at least NP-complete for lattice MSR.

Previous surveys in this field have focused mainly on the hardware problem. One exception is a survey by Ahmadzadeh and Masehian (2015), which

broadly focused on the software challenges of these systems, giving an extensive review of existing methods and algorithms for reconfiguration planning, locomotion control, and synchronization. Though many details were given on the topic of SR and current abstraction and solution methods, we contend that a more in-depth review and comparison of existing methods on this particular topic ought to be proposed.

This paper therefore aims to deliver a detailed account of the current research on modular robotic self-reconfiguration for shape formation especially in its three-dimensional lattice-based variant as well as more theoretical works. We proceed by outlining the various high-level methodologies present in the literature, then dive down into the specifics of the algorithms and their underlying models, before focusing on their application towards PM. This translates into the following paper organization: First, we identify three different approaches that researchers in the field have adopted to tackle the self-reconfiguration problem. For each approach, we outline its inherent characteristics and constraints, and briefly mention the main models and works that it encompasses. Then, in Section 3 and based on Figure 7, we provide an in-depth summary and comparison of the self-reconfiguration algorithms from the previous section. Finally, we re-frame the self-reconfiguration problem within the context of programmable matter, discuss properties particularly relevant for this applications, and point out promising opportunities for future research.

2. Classification of Self-Reconfiguration Approaches

Historically, researchers in the field of modular self-reconfigurable robotics have initially focused their efforts on the hardware problem of building metamorphic robots; then, research interest in the generic control of classes of these systems gradually emerged and numerous software frameworks were proposed; more recently, researchers started showing interest in what could be considered as a theoretical kind of metamorphic system, in the form of *Self-Organizing Particle Systems*.

In fact, from these three classes we can derive three approaches to SR algorithm design, that we will thereafter refer to as *Bottom-Up*, *Top-Down*, and *Theoretical*, as shown on Figure 2. These approaches differ by how they relate to their target execution platform (Is the target platform designed to accommodate the algorithm, or is it the other way around?), and, therefore, by the nature of the constraints that make up the model used by the algo-

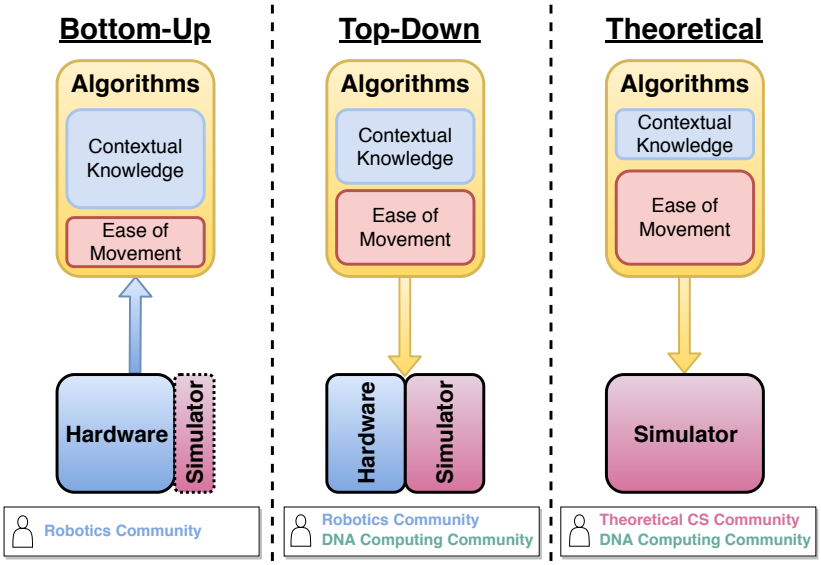

Figure 2: REVISED The three approaches to designing self-reconfiguration methods and their characteristics. (Best seen online with colors.)

rithm. Accordingly, the *Theoretical* approach deals with SR in the abstract, where the complexity and capabilities of the underlying model are often reduced to their minimum. *Bottom-Up* expresses the fact that the hardware systems were designed originally and algorithmic solutions for these specific systems have been subsequently proposed; the control software is hence inherently constrained by the particularities of the specific target platforms and lacks generic features in most cases. Conversely, *Top-Down* expresses the inverse relation, in which algorithms tend to be more generic, using models of the robots in which their specificities are abstracted and that are thus applicable to wide varieties of MSR. This approach will receive most of our interest, as comparison between generic algorithms is more enlightening. The essential difference with the Theoretical approach is that Top-Down works are generally based on more complex and powerful models for which they attempt to solve specific problems, while Theoretical works would instead attempt to solve problems as efficiently as possible by reducing the power of the model to its minimum (e.g., constant memory, no identification of modules, no synchronisation, etc.).

Begin Revision Furthermore, as made visible on Figure 2, various research communities are involved in the different approaches:

- Roboticists have the exclusivity of the Bottom-Up approach because of their hardware expertise.
- Top-Down works, while essentially also produced by the robotics community, also counts works from the DNA computing and molecular programming community.
- Lastly, the Theoretical approach is followed both by the DNA computing and molecular programming community and then theoretical computer science community—mostly through the lens of combinatorial geometry and distributed computing.

End Revision

In this section, we will successively explore the three aforementioned approaches to self-reconfiguration, discussing their leading fundamental models or MSR (summarized on Figure 3 below)), and introducing the corresponding solutions that have been proposed.

2.1. Bottom-Up Approach

2.1.1. Overview

As we have just learned, the Bottom-Up approach translates into an initial focus on the modular robotic hardware. Researchers represented in this approach rather unsurprisingly tend to belong to the research community of roboticists. They have come up with numerous module designs, from Unit-Compressible Modules (UCM) like Telecube (Vassilvitskii et al., 2002) and Crystalline (Butler and Rus, 2003), to hybrids like M-TRAN (Fitch and McAllister, 2013) and Roombot (Spröwitz et al., 2010), the Fracta self-reconfigurable structure (Yoshida et al., 1998), as well as bi-partite systems like the Robotic Molecule (Kotay and Rus, 2000) and I-Cubes (Ünsal and Khosla, 2001; Ünsal et al., 2001). Many more designs can be found in the literature, but these are the ones that are used in the algorithms concerned by this analysis.

This method credibly results in the most difficult self-reconfiguration planning, due to the intricacy of the geometry of hardware modules or their motion capabilities. These systems usually have strong non-holonomic motion constraints, complicating the reconfiguration process as a result. Motion constraints can either be local: induced by the geometry of the modules and by blocking constraints; or they can be global: like the *connectivity constraint* which states that the entire system's graph has to remain connected

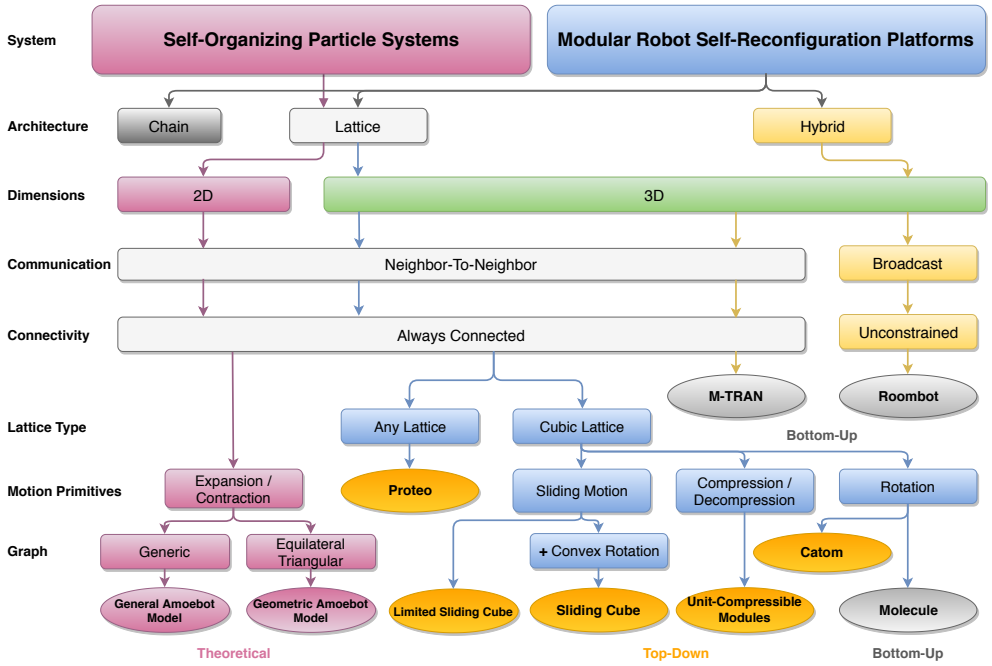

Figure 3: REVISED Overview of common self-reconfiguration models and select hardware systems. (Best seen online with colors.)

at all times. Several techniques for achieving holonomy at the cost of the granularity of the system have been devised, through the use of module aggregates with higher holonomy (*meta-modules*) (Ünsal and Khosla, 2001) or by having the system organized into a porous structure (Kotay and Rus, 2000) through which modules can flow unconstrained (*a scaffold*). While the kinematics are usually more complex in the *Bottom-Up* approach, modules are likely to assume a wider knowledge of their environment. These environmental facts come from sensor information about their orientation, position in the system, neighborhood, etc. Generally, as in the old saying *knowledge is power*, extensive environmental knowledge in individual modules allows for more straightforward algorithmic solutions, as learning the necessary facts could otherwise require a massive amount of communication.

2.1.2. Hardware Specific self-reconfiguration Methods

In this subsection, we review the hardware specific self-reconfiguration methods proposed in the literature for modular robots residing in 3D lattice

Figure 4: A snake-like formation of Roombot modular robots. (Courtesy of Prof Auke Jan Ijspeert, Biorobotics Laboratory, École Polytechnique Fédérale de Lausanne)

environments.

Kotay and Rus (2000) proposed a centralized solutions for their bipartite Molecule robot based on a hierarchical planner consisting of three levels. Task-level planning stands as the highest level of planning and selects a configuration that suits the task at hand. It then uses configuration planning to decide on a motion plan for Molecules to transform the initial configuration into the goal one. On the lower level, trajectory planning is used by the configuration planner to move individual modules to their goal position. They also introduced the aforementioned concept of *scaffolding*, to ensure that Molecules would converge into the goal configuration, though it made the granularity of their system extremely high as 54 modules constituted a single scaffold tile.

A similar approach was proposed by Ünsal and Khosla (2001) for the I-Cubes bipartite system, consisting of three-degree-of-freedom links used for communication and actuation, and passive cubes for the modules. They used a centralized two-level planner in which the high level planner decides on the position of modules in the goal configuration by using the low level planner to search for a feasible plan of individual link motions, that would move the module to the desired location. Several iterative improvements were later made on this approach, by introducing *meta-modules* to simplify planning and therefore adding another layer of planning on top of the existing two, at the meta-module level. Another work with I-Cubes used a centralized divide and conquer approach, where the problem of planning the motion of

a module from a position to another was divided into a sequence of local subproblems. They also used a two-level hierarchical planner in this work, where (1) solutions to subproblems were searched on the low level planner while (2) the high level planner was concerned with the actual motion of the module, combining solutions from the lower level (Ünsal et al., 2001).

Although these early works using centralized planner laid the groundwork for much of the field and introduced valuable problem simplification techniques, they are inherently lacking the robustness, scalability, and autonomy that is so critical in self-reconfiguration. Therefore, and as we will see below, researchers eventually turned to decentralized SR method, and thus also had to face the challenges of distributed algorithm design.

Yoshida et al. (1998) proposed a distributed algorithm based on local information for 3D reconfigurable structures with star-shaped modules. They put forward a description of the goal shape using connection types, as previously used in some of their works on 2D hardware. Their approach used local rules with added randomness, in the form of stochastic relaxation based on simulated annealing.

Spröwitz et al. (2010) designed the Roombot hybrid modular robot, which relaxes some of the strong constraints imposed on MSR that greatly complicated planning. Roombots can communicate with other modules through broadcast instead of the traditional neighbor-to-neighbor communication, and does not require the robot to remain connected at all times (which is a major constraint of nearly all other systems), though it requires the presence of a structured ground surface with passive connectors. They proposed a decentralized self-reconfiguration algorithm using meta-modules made of two stacked Roombots, which guarantees that individual modules can always move. Their approach relies on the locomotion of disconnected structures of Roombot meta-modules that converge into the desired configuration thanks to the attraction of a force-field and a predetermined assembly order.

Finally, a number of self-reconfiguration methods for unit-compressible modules—square or cubic (in 3D) modules that can contract and expand on each of their sides— have been proposed. However, MSR made of unit-compressible modules can somewhat both be thought of as specific hardware (e.g., Crystalline in 2D and Telecube in 3D) and a class thereof. We decided to consider the later and cover SR algorithms for these systems under the *Top-Down* approach in Section 2.2.4, as they could potentially be used generically on any future hardware with a similar actuation mechanism.

2.2. Top-Down Approach

Begin Revision

2.2.1. Algorithmic Conventions and Metrics

We will introduce in this section a systematic convention that will be used to compare the further introduced algorithms.

Number of Modules. Self-reconfiguration performance is usually evaluated relative to the number of modules in the configuration. Let n be this number.

Resolution. We can also introduce a resolution parameter k, for which n is proportional to k^d, with $d = 2$ or $d = 3$ depending on the d-dimensional space. The resolution expresses the size of the modules (or *meta-modules*, in relation to the size of the shape. Therefore a low-resolution configuration would mean that each pixel (or voxel) of the goal shape corresponds to a meta-module made of many individual modules, while in a *full-resolution* configuration each individual module corresponds to a single pixel (or voxel).

Complexity. The complexity of reconfiguration algorithms is generally expressed as number of reconfiguration steps and number of motions. The number of reconfiguration steps is expressed in number of time steps. These time steps can be either synchronous, in which case a single time step commonly corresponds to the time required by the rotation of a single module (to which we will refer to as *reconfiguration time* throughout this survey), and where modules are often assumed to perform synchronously; or they can be asynchronous, in which case the duration of a single time step is defined by the authors. On the other hand, the number of motions represents the total number of motions performed by all modules during the entire reconfiguration. Both these complexities are expressed relative to the number of modules in the system n.

Furthermore, the complexity of the total number of messages exchanged during reconfiguration is another notable indicator of how efficient an algorithm performs at the network level, also expressed relative to n.

Additional complexities that could be worth investigating are the *real reconfiguration time*, in seconds, which would require the actuation time of modules to be known, or the *number of CPU operations* (global or per module depending on the underlying architecture).

Architecture. Finally, there are number of different architectures that can be used for modular robotic systems. The first main distinction to be made is between centralized systems, in which all computation is performed on a single module of the system or on an external computer, and distributed systems, where the computation is performed distributedly across all modules in the system. If a distributed architecture is in use, the system can either be synchronous or asynchronous, depending on whether or not the modules rely on a global synchronization of the system to perform their tasks. Furthermore, communication between modules can be either local, in which case modules can only communicate with their immediate neighbor, or global, where any module can communicate with any other module, through unicast or broadcast communications. Finally, memory access can also be local to the module or their immediate neighbors, or global to the whole system.

Most of the works that will receive the focus of this paper assume distributed systems using local communications and memory accesses.

End Revision

2.2.2. Overview

With versatility being a major concern of researchers when designing self-reconfiguration algorithms, the *Top-Down* approach has a crucial role: creating shape formation methods that are not tied to a specific hardware implementation, and that can be applied to various MSR in a generic fashion. Moreover, a software-first approach where algorithms can help point out interesting requirements to include in the hardware platforms. These algorithms usually operate on models with particular kinematic capabilities and constraints that represent classes of robots, as can be observed on the map of usual models and select hardware systems on Figure 3.

Due to geometrical and mechanical attributes of robots being more generic, motion planing for these models tends to be slightly less demanding than for the specific hardware platforms found in the *Bottom-Up* approach. Conversely, these models have weaker assumptions about the environmental knowledge of the modules on average, and computation therefore tends to be heavier.

2.2.3. Generic Algorithms

A number of self-reconfiguration methods that can be found in the literature are truly independent of any particular hardware implementation whatsoever. These are the most generic algorithms, either at the level of

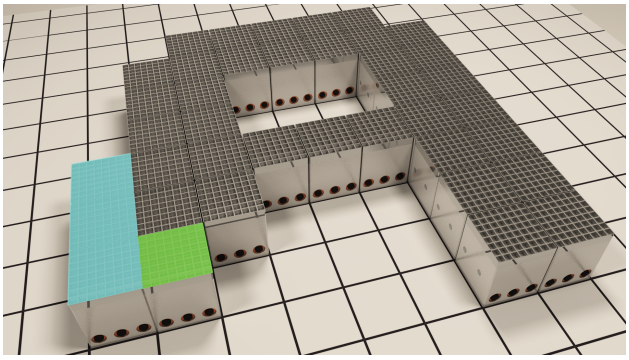

Figure 5: A sample configuration of modules from the Sliding-Cube model performing reconfiguration into a 2D *A* shape. (From the Smart Blocks project (Piranda et al., 2013))

modular robots in general, or for a particular class thereof as in the following work.

Dewey et al. (2008) designed a system of meta-modules for lattice-based modular robots named *Pixel*, that could considerably simplify reconfiguration planning in massive modular robots. Their main idea is to divide the reconfiguration problem into a planning task and a resource allocation task. The role of the former is to decide what meta-module positions in the goal configuration have to be filled next, and the one of the latter to decide where the meta-modules filling that position should be picked from. They achieve holonomy on their meta-modules by allowing them to be in two states: a *filled* state, and an *empty* state. Modules are able to internally flow from a meta-module in the *filled* state to a meta-module in the *empty* state, hence performing a swap, and moving through the structure in predetermined manner—though the fundamental problem of local planning for module flow is not addressed in their paper. The difficulty is thus shifted from actual reconfiguration planning to creating meta-modules that have the desired holonomic features, and designing local rules for internal module flow between meta-modules. Finally, they show that their planner is complete and demonstrates a reconfiguration time that scales linearly with the diameter of the system.

Another approach to fully generic algorithm (for any architecture) can be found in (Fitch and McAllister, 2013), in which the authors use a two-level hierarchical framework where the planning problem is formulated as a distributed *Markov Decision Process* (MDP). An MDP is defined by a 4-tuple $\langle S, A, T, R \rangle$, where: S is the set of *states*, represented by open positions

to be filled by modules—which is relative to the number of faces of the modules; A is the set of *actions*, represented by the disconnection of connector from a neighbor module and the reconnection to another, potentially using a different connector; T is a stochastic or deterministic *transition function* that decides on the next action to perform; R is the expected *reward*, set to -1 as a way to minimize the number of moves. The authors solve this MDP using a distributed implementation of dynamic programming using message passing. The MDP operates on the higher level of the planner, determining for each mobile module (i.e., that **can** move) on which other module and connector it should attach during the next time-step. Then the low level planner computes the sequence of individual module motions that the moving module should follow in order to disconnect from its current neighbor and reconnect at its new anchor point. Modules search through the structure to ensure that they are not an articulation point of the system's graph to decide whether or not they are mobile—so as to satisfy the connectivity constraint—and lock a portion of it during their motion if mobile. As several modules can lock the same portion of the structure, they can also move in parallel, hence quickening the reconfiguration process. In this scenario, the complexity lies in designing an efficient kinematic planner to act as the transition function T.

2.2.4. Unit-Compressible Modules

Butler and Rus (2003) generalized their PacMan self-reconfiguration algorithm for 2D unit-compressible modules to 3D systems. An advantage of UCM is that they are able to travel through the volume of the structure, hence potentially benefiting from a higher number of parallel movements, and a shorter distance to their target compared to surface moving modules. In this work, the authors use a technique called *virtual relocation* to move modules from one end of the configuration to the other, swapping their identity with the modules compressing and decompressing along the path to their target position. PacMan is based on a two-stage distributed planning algorithm, wherein: (1) modules locally compute the difference between the current shape and the goal shape in order to decide on which modules should move; (2) a distributed search (depth-first search or deepening iterative search) for a mobile module is performed from the target position, dropping *pellets* along the way to mark the path to be followed by the selected module. A specific actuation protocol is then followed by the modules to make their way through the path without causing deadlocks or disconnec-

tions. An interesting aspect of this work is its high parallelism and efficiency, though it requires all modules to be have a unique label.

In a similar work, Vassilvitskii et al. (2002) also used an algorithm in two phases, this time to reconfigure systems of Telecube unit-compressible modules. Their method had not only local decision-making, but also some degree of parallelism and completeness. Distributed planning was performed on cubic meta-modules made of eight Telecube modules, with: (1) a path planning phase inspired by the original *PacMan* algorithm for 2D unit-compressible modules with an exponential deepening search to find a mobile module as close to the goal as possible; (2) an execution phase where meta-modules would translate their motion plan into a sequence of individual meta-module motion primitives and execute it. The whole reconfiguration could be performed in worst-case $O(n^2)$ time.

2.2.5. The Proteo Model

Introduced by Yim et al. (2001), the *Proteo* model includes constraints on the configuration space of the modules and their movements. Several of the recent works on 3D self-reconfiguration are using models that are different variants of *Proteo*. Metamorphic systems from the *Proteo* class share the following properties: (1) **homogeneity**: all modules share the same electro-mechanical and physical structure—as opposed to heterogeneity, where modules constituting a single MSR can be of various types; (2) **connectivity**: the system must remain connected at all times; (3) **mobility**: each module has motion capabilities; (4) **locality**: only neighbor-to-neighbor communication is allowed and each module is embedded with a local processor. Furthermore, *Proteo* modules only reside in lattice environments, where movements are only allowed from one cell to an open adjacent one, and with the help of a support module acting as pivot—for rotation or sliding motion. These individual movements are treated as discrete steps. Though it is assumed that connectors between modules are strong enough to support all possible movements and configuration—hence ignoring structural mechanical constraints—a moving module cannot carry another with it. Additionally, another noteworthy aspect of the *Proteo* model is how modules deal with motion constraints. When a motion occurs it must not result in a collision, nor split the robot into two disconnected structures. While these constraints are characteristic of SR models, it is assumed that *Proteo* modules can proactively sense both local and global violations (through embedded sensors) related to: (1) the movement of their motion pivot which would forbid their

subsequent motion; (2) a motion that would result in a collision or deadlock; (3) a motion that would result in a violation of the connectivity constraint. As we will later discuss, these are quite strong assumptions, as preventing collisions, deadlocks, and preserving the connectivity of the systems through communication or coordination greatly hinder motion planning.

In the same paper, Yim et al. proposed a distributed self-reconfiguration algorithm for their class of modules based on local information and a coordination mechanism that they name *goal-ordering*. Two methods for attracting modules to goal positions are put forward. In both of them, the mechanism of *goal ordering* ensures that modules avoid overcrowding around a single goal position by allowing them to reserve one if they satisfy a set of constraints, and implements some coordination mechanisms to help nearby modules get into position. In the *distance-based* method, modules are attracted to the closest unfilled goal position using Euclidean distance, whereas the *heat-based* method uses a heat flow technique with accessible unfilled positions acting as heat sources, and modules not yet in position acting as sinks. Modules climb the gradient by moving towards positions with higher temperature. Furthermore, in order to prevent modules from getting trapped, randomness is used as a temperature tiebreaker and for adding noise to the goal ordering process—by regularly picking the second-best open position. A combination of the two gradient methods is shown to provide the best results, where the algorithm defaults to the distance-based method and switches to heat-based when stuck. Experiments show that reconfiguration time scales roughly linearly with the number of modules, albeit never converging into the goal shape in some cases, generally because of overcrowding issues.

2.2.6. The Sliding-Cube Model

The *Sliding-Cube* model has a lot in common with *Proteo*, from which it differs mainly by an absence of homogeneity constraint and less powerful embedded sensors. This model has been extensively studied within the context of self-reconfiguration due to its simple kinematics. Modules under the *Sliding-Cube* model can be attached to up to six neighbors using connectors on each of their faces. They are capable of performing sliding motions on the surface of neighbor modules, as well as convex rotations along their edges. In contrast with the the Proteo model, *Sliding-Cube* modules are assumed to be fitted with sensors that can only sense local information (mutual exclusion and blocking issues), they cannot decide on the violation of the connectivity constraint through the same means. *Sliding-Cube* algorithms can be imple-

mented on varied hardware systems, potentially leveraging meta-modules to achieve cube-like structures with the proper kinematics. Some of the existing compatible hardware systems include UCM such as Telecube (Vassilvitskii et al., 2002) and Crystal (Butler and Rus, 2003), Molecule (Kotay and Rus, 2000), hexagonal lattice systems such as Fracta (Yoshida et al., 1998), and the M-TRAN hybrid MSR (Fitch and McAllister, 2013).

This model was first introduced by Fitch et al. (2003), in which they also proposed the *MeltSortGrow* algorithm, that can reconfigure an heterogeneous MSR in an out-of-place manner, through sequential module motions. The algorithm consists of three phases: (1) in the *Melt* phase, the initial configuration is disassembled into a line, used as intermediate configuration to simplify planning; (2) during the *Sort* phase, the line is sorted according to the type of the heterogeneous modules and their position in the goal configuration. The line is folded in two so as to avoid breaking the connectivity constraint; (3) with the final *Growth* phase, the goal configuration is sequentially assembled from the sorted line configuration, by repeatedly moving the module at the tail of the line into its goal position. Both a centralized and decentralized version of the algorithm were proposed. In the decentralized version, distributed planning is used to find a path for mobile modules to and from the intermediate configuration. A surprising finding they made is that reconfiguration planning for heterogeneous modular robots is not asymptotically harder than homogeneous reconfiguration as previously thought, as they achieved $O(n^2)$ and $O(n^3)$ reconfiguration time for the centralized and decentralized versions, respectively. Clearly, the main problem with this approach—aside from the sequential motion of modules—is the amount of free space that it requires. Therefore, the authors decided to investigate the effect of free space restriction on the self-reconfiguration of heterogeneous *Sliding-Cube* modules. In (Fitch et al., 2007), they introduced the *TunnelSort* algorithm for solving SR problems among obstacles in $O(n^2)$ time. Modules navigate on the interior of the structure through tunneling and on a one-module thick crust on its surface. This assumption on the presence of a free space crust is abandoned in (Fitch et al., 2005), in the *ConstrainedTunnelSort* algorithm, hence enabling reconfiguration in environments consisting of arbitrary obstacles, forming what they refer to as a *bounding region* around the system. Both algorithms consist of an homogeneous phase, in which modules organize into the desired goal shape independently of their module type, and a module swapping phase, wherein individual modules in the goal configuration are swapped through tunneling in order to attain the correct type specifications

of the goal shape. While *ConstrainedTunnelSort* is not complete, its homogeneous phase shows $O(n^2)$ reconfiguration time and moves and $O(n^4)$ time and moves ($\Theta(n^2)$ in practice for common cases) in the second phase.

Moreover, Zhu et al. (2017) combined Cellular Automata (CA) and Lindenmayer-systems (L-Systems) in order to efficiently reconfigure a *Sliding-Cube* modular robot into a class of goal shapes that can be described by branching structures, in a robust, distributed, and highly parallel manner. The CA rules are used to control the movements of the modules and enable the growth of the goal reconfiguration, from the turtle interpretation of the L-system. The desired structure is grown from an initial seed module, and a new seed module is added at each branching in the growing structure and can initiate the growth of a substructure in parallel. Their approach demonstrates a linear increase in reconfiguration time with the number of modules. As with many other algorithms experimenting with unconventional shape representation techniques, a major drawback of this approach is the difficulty of designing the L-system rules that correctly describe the desired configuration. It is nonetheless a clear example of a self-reconfiguration algorithm that is highly specialized and efficient for a specific class of goal configurations.

Additionally, Støy used unspecified modules with kinematics similar to those of the *Sliding-Cube* in (Støy, 2006) and (Støy and Nagpal, 2007). His first approach was to use a two-step approach based on a CA, scaffolding, and attraction gradients. It consists of an off-line preliminary step, wherein CA rules are generated from a Computer Aided Design (CAD) model or mathematical description of the goal shape. This description is first made porous so as to build a scaffold to ease reconfiguration. The self-reconfiguration starts from a seed module controlled by the CA, that attracts wandering modules (i.e., not in goal shape) by the means of attraction gradients for them to descend. Collisions are avoided thanks to the scaffolding structure and a local distributed algorithm is used for connectivity-checking. Their algorithm is convergent and reconfiguration time grows linearly in the number of modules. The main drawback of this method is that it is not systematic: a different cellular automaton needs to be generated for every goal configuration. To circumvent this problem, (Støy and Nagpal, 2007) proposed to replace the CA-based shape description method by a volume approximation of the goal shape using a set of overlapping bricks of different sizes, while still forming a scaffolding structure. The resolution of the volume approximation is adjustable, with higher resolutions requiring more modules. Local rules are used to replace the CA from their previous algorithm, and need only to be

created once, as they are not tied to any particular reconfiguration problem. They show that different kinds of attraction gradients can be used depending on whether one is seeking to minimize time and number of messages or the number of individual module motions.

2.2.7. The Limited Sliding-Cube Model

Kawano investigated a version of the *Sliding-Cube* model with increased kinematic constraints where convex rotations are not allowed. He demonstrated both homogeneous and heterogeneous self-reconfiguration of these systems, using algorithms based on local rules and meta-modules that guarantee the preservation of the connectivity and the existence of a mobile module in the structure. In (Kawano, 2015), the author showed that SR using homogeneous *Limited Sliding-Cube* modules could be performed in quadratic time using meta-modules and tunneling motions, even in environments with obstacles. This approach benefits from a high degree of motion parallelism. It is later extended in (Kawano, 2017) for heterogeneous reconfiguration. In (Kawano, 2016), a different approach is proposed for heterogeneous modules, using a compression and decompression mechanisms with virtual walls, that condenses the initial shape at the start of the reconfiguration in order to perform modules swaps through tunneling (for ensuring the proper placement of heterogeneous modules), and expands it into the goal shape at the end. However, this method seems to rely on the assumption that modules have enough force to carry or lift an arbitrary number of modules on one of their faces. Moreover, although the author uses 2×2 meta-modules during part of the reconfiguration to ease permutations by providing motion pivots to sliding modules, the goal shape is not described at the meta-module scale; therefore the granularity of the system is not increased—thus achieving what the author refers to as a *full-resolution* algorithm.

2.2.8. General Cubic-Lattice-Based Works

Lengiewicz and Holobut (2019) tackled reconfiguration by having modules flow through a porous structure with moving boundaries, incorporating interesting aspects from the scaffolding method in Støy (2006) and (Støy and Nagpal, 2007), as well as the multi-stage reconfiguration with parallel movements between boundaries from the *PacMan* algorithm (Butler and Rus, 2003). Their method uses maximum-flow searches to reconfigure massive ensembles of cubic modules on a cubic lattice through a scaffold formed by porous 7-module cubic meta-modules. Their approach decomposes the

reconfiguration problem into two partly disjoint subproblems: (1) trajectory planning from the current to the goal configuration (i.e., deciding how the boundaries of the current shape should evolve in order to reach the goal configuration); (2) finding an optimal flow of modules between the boundaries of the current shape and through its volume. Their algorithm is remarkably efficient as the number of module movements is proportional to the resolution of the robot, that is to say $O(\sqrt[3]{n})$ movements. There are a few caveats in their method however, that would require additional coordination measures for which they propose several solutions that could be investigated: (1) the existence of virtual modules acting as heat sinks and able to communicate while not physically present; (2) no connectivity preservation mechanism—that they propose to solve using a connection gradient; (3) the reliance on a global streamline planner, for which they propose an asynchronous and distributed version in the same work, which is quite efficient and based on local memory and local communication assumptions only. This work could possibly be extended to any other hardware systems capable of performing internal movements through a scaffold arrangement of these system.

2.3. Theoretical Approach

The third approach to the SRP is led by the theoretical computer science community. They are interested in distributed shape formation and programmable matter in their most fundamental form. Their central question is this: Once most of the constraints regarding the hardware have been stripped out, what can be said about the self-reconfiguration problem, and what sensor information and contextual knowledge is truly indispensable to shape formation?

Predictably, and as displayed on Figure 2, this approach operates solely at the software level, developing algorithms to be experimented on using simulation, without concern for the hardware. They also differ from other approaches by the nature of the assumptions that form the basis of their research. Here, researchers are less concerned with the kinematics of the computational units (referred to as *particles*), but are rather interested in the basic abilities of the particles in relation to their own state knowledge. The chief model that has been considered on this particular form of metamorphic systems, named *Self-Organizing Particle Systems* (SOPS), is the *Amoebot* model (Derakhshandeh et al., 2015b; Daymude et al., 2019), inspired by the biological behavior of Amoebae. In its most abstract version, the particles reside on a graph, that represents all the possible positions that a connected

set of particle may assume. However, it is the infinite equilateral triangular graph that is generally used in algorithms for practical purpose, where particles are arranged on a 2D triangular lattice. Besides, some important properties of the model must be highlighted. For instance, all particles have constant memory size, modest computational power and do not store any identifier. Furthermore, particles do not have access to global information and all their decision-making happens at the local level, in a synchronous fashion. Particle motions are constrained in a more traditional manner however, with individual units unable to carry another because of their limited physical strength. More importantly, all particles in the system are required to form a single connected component at all times—which is equivalent to the connectivity constraint. This motivates the choice of movement capability of the particles, where they perform motion through expansion and contraction primitives, effectively occupying either one or two adjacent graph positions at all times. More complex forms of motions are also introduced, such as *handovers*, where a particle contracts out of a node while another expands into it simultaneously.

While it is true that other theoretical models of Programmable Matter exist besides Self-Organizing Particle Systems as the *Amoebot* model, especially a number of works emerging from the DNA computing and molecular programming communities, we believe that the lack of real reconfigurability of these models implies that they are too far removed from modular robotic systems to be pertinent to this survey. This is because this models are either passive systems that cannot autonomously decide on their motion but are instead pre-set to reach a desired configuration such as for instance *DNA Tile Assembly* models (Doty, 2012; Patitz, 2014), *Population Protocols* (Angluin et al., 2006), or in the case of *Hybrid Programmable Matter* systems of active robots acting on passive tiles (Gmyr et al., 2017). We may nonetheless note that the *NuBot* active model has been used to perform simple shape formation through self-assembly by Woods et al. (2013), which makes it more pertinent than the aforementioned works to the topic of this survey, and can be of interest to the reader. Furthermore, programmable matter can take the form of programmable self-folding molecules, able to transform into any shape by folding, though these systems resemble more chain-type modular robots than lattice-based modular robots. This has been even realized in hardware (Knaian et al., 2012), therefore shifting this particular work towards the Top-Down class.

2.3.1. Shape Formation in Self-Organizing Particle Systems (SOPS)

In this section we will exclusively focus on works concerning the geometric version of the Amoebot model, due to its similarities with lattice-based modular robots. A number of distributed and local algorithms have been proposed for the purpose of shape formation, which will be mentioned below. For simplicity, early works on the problem of shape formation in SOPS have focused on having particles self-organize into primitive shapes. It was first demonstrated by Derakhshandeh et al. (2015b) with the example of a line. The particles had to self-organize to form a line on the triangular grid, with an additional condition: all the particles constituting the line have to be in a contracted state. Their approach assumes the existence of a seed particle (for which the authors also propose a leader-election algorithm in the same paper) that defines the starting point of the line. Other particles organize themselves into a *spanning forest*—essentially a spanning set of disjoint trees. The root of each tree represents a leader that will rotate around the goal shape in a predetermined direction, while followed by all particles in the tree, in a snake-like manner. Trees of particles hence rotate around the growing line until they reach one of the ends on each side of the seed, and add themselves to it one at a time. This process is guided by local rules determining the next valid position and be filled, and can reconfigure any connected set of particles into a line with worst-case $O(n)$ rounds (where a round involves a single motion from all particles) and $O(n^2)$ moves (contractions and expansions). The *spanning forest* component used for implementing *leader-follower* motions is one of the fundamental components of all the shape formation works in SOPS mentioned below.

Based on this work, a general framework for shape formation in SOPS was later proposed in (Derakhshandeh et al., 2015a), wherein the authors demonstrate shape formation into scale adjustable triangular and hexagonal structures from any connected set of particles in $O(n^2)$ moves. This algorithms also relies on a leader-follower approach, with particles rotating around the shape being formed (initially only made of the leader particle, or *seed*) in a *snake-formation*, until they reach the next position to be filled in the growing shape.

This is later extended to support the formation of any shape, with a general SOPS shape formation framework in (Derakhshandeh et al., 2016). This framework relies on a few assumptions however: (1) the particles initially form a connected set arranged in a not-necessarily-complete triangle; (2) the

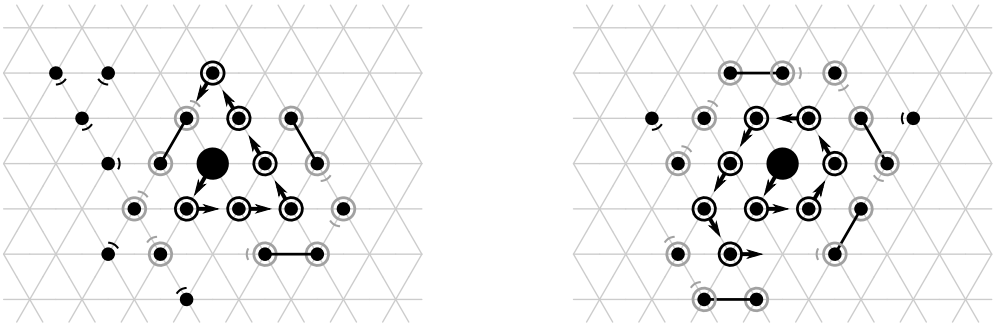

Figure 6: Shape formation of a triangle (left) and of an hexagon (right) on a 2D triangular lattice with the Amoebot model. (Courtesy of Prof Andrea Richa, Self-organizing Particle Systems Lab, Arizona State University)

goal shape can be described as a constant number of equilateral triangles, whose scale depends on the number of particles in the system; (3) particles know their orientation and move in a clockwise manner; (4) particles can use randomization; (5) particles motions are scheduled in a sequential manner. Under these assumptions, this shape formation algorithm can form any shape using only local information and in a distributed fashion with parallel movements, and in worst-case $O(\sqrt{n})$ rounds. The algorithm first reconfigures the particle system into a line formed by equilateral triangles, each containing a *triangle coordinator* particle, that can direct expansion and contraction motions of the entire triangle, in a way reminiscent of meta-module motions in MSR. The goal shape is then built by a series of triangles motions in a specific assembly order computed using a set of rules.

Since then, another approach to a general SOPS shape formation framework with an equilateral triangle approximation has been proposed by Di Luna et al. (2018b), that relaxes some of the assumptions made in the previous algorithm by Derakhshandeh et al., albeit with an $O(n^2)$ moves and rounds complexity. An interesting finding is that predetermining the orientation of movement is not a necessary condition. The authors demonstrate that their algorithm is complete if randomization is allowed, without requiring a specific initial arrangement of particles. Their proposed shape formation method consists of a sequence of seven phases, including spanning forest construction, agreement on the direction of movement, intermediate lines formation, and the final goal shape construction phase.

Finally, several problems derived from shape formation have been investigated. Firstly, the problem of forming a shape that achieves the maximum

compression for a given set of particles, solved using a stochastic approach based on Markov chains in (Cannon et al., 2016). Then, the problem of shape recovery, in which particles are assembled in an arbitrary shape and some defective particles in the system must be discarded while maintaining the current shape. It was demonstrated in (Di Luna et al., 2018a), with a line recovery technique resulting in a scaled-down version of the line after defective particles have been removed.

3. Analysis of 3D Lattice Self-Reconfiguration Algorithms

In this section, we examine how the works mentioned in the previous section compare against each other on a variety of aspects, in order to provide a clearer overview of successful methods and their inherent compromises. This discussion is based on Figure 7 (which depicts a comparison of modular self-reconfiguration and particle self-organization methods in a tree-style manner), but goes one step further as some characteristics of the works had to be left out of the diagram for the sake of clarity. Furthermore, even though they appear in Figure 7, works previously classified as part of the *Bottom-Up* and *Theoretical* approaches are purposely left out of the discussion, due to the impediment to comparison caused by their lack of genericity and the different nature of their underlying system, respectively.

3.1. Planning Under Mechanical Constraints

Most of the SR algorithms presented here do not truly take into account physical constraints in their planning, such as those unavoidably imposed by gravity in 3D systems. It is however a salient requirement of SR algorithms is they are to be realized in actual hardware systems and on a large scale in the future. Some researchers have recently shown interest in this problem. In (Hołobut and Lengiewicz, 2017), the authors mention two types of mechanical failures that can occur in a MSR: (1) *loss of stability* due to a shift in the center of mass of the system, which might be caused by the movement of modules; (2) *structural failure*, induced by the breaking of a bond between modules due to an excessive load imposed on a connector. The configuration stability problem was mentioned in (Butler and Rus, 2003), in which the authors state that stability can be insured under a few conditions, and for reconfiguration on a small class of shapes that they call *stem cells*, thanks to UCM traveling through the volume of the structure. Besides, failures due to overstressed connectors are investigated in (Hołobut and Lengiewicz,

2017), where the authors present a distributed procedure for predicting if the next reconfiguration step will cause a structural failure. Ideally, procedures of this sort could be added to the planning process of self-reconfiguration algorithms so as to further constrain possible motions to mechanically safe ones exclusively.

3.2. Free-Space Requirements and Obstacles

Most algorithms do not explicitly factor in the amount of free space required by the reconfiguration. *Free space* requirement is defined in (Fitch, 2004) as "the total amount of space occupied by intermediate configurations during shape-changing". Within that context, the most desirable space related property for a given algorithm is that it can perform *in-place* reconfiguration, which means that it requires no more space than the union of the initial and goal configurations. As can be observed in Figure 7, it concerns most algorithms, though a slight degree of tolerance is granted in some cases. It follows naturally that algorithms requiring an arbitrary amount of free space perform *out-of-place* reconfiguration, which is evidently prohibitive to most applications. Some instances of out-of-place reconfigurations are: (Fitch et al., 2003), where a line of length n is used as intermediate configuration, hence requiring a massive amount of space in a given plane, and (Kawano, 2016) in which the compression and decompression mechanism inevitably uses more space than is desirable. Fitch et al. further investigated self-reconfiguration under free space constraints: (1) in (Fitch et al., 2005), wherein their algorithm only assumes a one-module thick *crust* around the intermediate configurations—which might also help in environments with obstacles; (2) in (Fitch et al., 2007) where the reconfiguration is constrained by an arbitrary shape (a *bounding region*) that blocks the motion of modules. The latter is analogous to performing reconfiguration among obstacles, a problem which was also studied in (Kawano, 2015) and (Kawano, 2017). The difficulty of motion planning in environments with obstacles naturally comes from the restriction it imposes on module movements, even preventing reconfiguration altogether in some cases.

3.3. Collision and Deadlock Prevention Mechanisms

The path-planning of modules constitutes an extremely complex problem to solve by itself, but it becomes incredibly more tedious when motion parallelism is considered. Indeed, when considering concurrent module motions, two kinds of additional kinetic constraints appear: movement blocking and

deadlock. The former relates to the presence of one module preventing another to move, either while moving or simply by having a disadvantageous position. (This is especially a problem in rotating modules.) The latter can happen when two modules attempt to concurrently enter the same lattice position, and either results in a collision or requires additional coordination measures to be solved. Researchers have come up with several ways to tackle this collision avoidance problem.

A first solution is to use what we name *kinematics simplification methods*. One of these methods is *scaffolding*, which consists in having modules in the configuration organize into a porous structure through which modules can flow to their destination without blocking, at the cost of a higher granularity. An additional consequence of using a scaffolding structure is that hollow, solid, and concave shapes become no harder to build than regular shapes, by suppressing local minima issues thanks to the free passage of modules across the whole system.

Another method aggregates modules into logical units named *meta-modules*. If carefully designed, meta-modules can have holonomic properties that greatly simplify planning. SR frameworks using meta-modules usually perform motion planning at the meta-module level, and use local rules to realize the transitions between meta-module states at the level of individual modules. Depending on their function, using meta-modules can have a negative impact on granularity, as shape description is also done at the meta-module level: while one voxel normally one voxel equals one module, then one voxel equals one meta-module when they are in use. It is common to have cubic $2 \times 2 \times 2$ meta-modules in 3D algorithms, but they can also be much larger as envisioned in (Dewey et al., 2008). As can be seen on Figure 7, the use of scaffolding and meta-modules in self-reconfiguration methods is now commonplace.

A complementary approach is to have modules rely on sensor information—thus assuming the presence of potentially very powerful sensors on modules—or communication to avoid collisions. Modules can either adopt a *proactive* collision solving mechanism, in which they detect movements that will results in collisions and abort or avoid planning them at all; or a *reactive* mechanism, in which the modules wait for a collision to occur and decide *a-posteriori* the way forward. A *proactive* stance involving sensors is often assumed, as in the *Proteo* and *Sliding-Cube* SR algorithms (Yim et al., 2001). Proactive detection through communication is never used in practice, because this process could be extremely costly in term of time and messages exchanged as query-

ing every module to ensure it is not blocking to the current motion would necessarily result in a complete flooding of the configuration graph on each verification. Reactive deadlock resolution is also avoided, as collisions could potentially put at risk the regularity of the lattice and jeopardize the entire self-reconfiguration.

In hardware models based on rotation-only primitives and surface motion, such as the quasi-spherical *Catom3D* (Piranda and Bourgeois, 2018), the problem of detecting potentially blocking modules becomes even more essential. Indeed, if sensor detection is not available, ensuring the safe rotation of a module must imply a full-traversal of the network to detect potential collisions, which is time- and message-prohibitive. This is an instance of a hardware requirement for relaxing collision avoidance computations, as discussed in Section 4.2.

3.4. Goal-Shape Representation

Researchers have come up with ingenious way to represent the goal configuration over the years. In most cases algorithms assume this representation to be globally known by modules, therefore motivating research on efficient shape description techniques so as to avoid overloading their limited memory. In lattice systems, one of the simplest representation of a goal shape is using a grid, which comes at a high memory cost as the size of the representation scales with the number of modules. A shared representation where the description is disseminated across the modules constituting the system could also be considered, but it depends on a number of challenging problems related to data dissemination and retrieval in distributed systems that would need to be solved first (Bourgeois et al., 2016).

Fitch and McAllister (2013) investigated representing the goal shape as a volume (which they termed *bounding box*) that modules have to fill in order for the reconfiguration to complete. This technique works nicely for convex shapes but requires additional assembly rules for other shapes.

Some works have used description of goal shapes with variable resolutions, where lower resolutions require fewer modules and are thus faster, whereas higher resolution provide a higher level of detail at the cost of longer reconfiguration times and an increased size of the robot. This has been investigated by Støy and Nagpal (2007), through a volume approximation of a CAD description of the goal shape using overlapping bricks of various sizes. A resolution parameter can be supplied that alters the positioning of the

bricks in order to reflect the desired resolution. This approach has the advantage of not having the size of the description increase with the number of modules, but rather with the complexity of the goal shape. It contrasts with a previous work by Støy (2006), where the shape description was embedded into the rules, and whose number would grow linearly in the number of modules. Lengiewicz and Holobut (2019) also used a variable resolution of the goal shape in their max-flow algorithm, though using a grid representation at the meta-module level.

Though it has not yet been used in a SR algorithm per se, a promising vectorial method for compact shape representation was introduced by Tucci et al. (2017), inspired by a common technique in image synthesis. It uses *Constructive Solid Geometry* (CSG) to describe an object as a tree of primitive geometrical objects, transformations, and set operations, thus having the size of the representation scale with the complexity of the shape and allowing for adjustable resolution at a negligible cost.

Goal shape representation can also be absolutely central to the algorithm, such as in the self-reconfiguration algorithm for branching structures by Zhu et al. (2017), which relies on a recursive description based on L-Systems, defining the goal shape with a rewriting system and formal grammar. A strong advantage of this approach is the compactness of the description, which comes at the expense of a lack of generality of the algorithm, which is narrowly specialized in self-reconfiguration for branching structures.

3.5. Solution Methods

Solution methods have already been detailed on a case-by-case basis in Section 2 for every algorithm covered in this paper. Nevertheless, it is worth noting that on a higher level, researchers have so far largely relied on three categories of approaches in order to have modules move into position in the goal shape: (1) searching through the configuration for a mobile module or reachable open position while building a motion path; (2) attracting wandering modules to open goal positions through gradient-like techniques; (3) emergent methods based on local rules and CA. An extensive survey of abstraction and solution methods in the context of modular robotics control was offered by Ahmadzadeh and Masehian (2015) and covers this topic in detail.

3.6. Surface Movements vs. Internal Movements

Two paradigms for module movements exist: surface movements and internal movements—i.e., through the volume of the object. Sometimes a combination of both is used as demonstrated in some *Sliding-Cube* algorithms. Internal movements currently exist in three flavors which are: (1) compression / decompression with UCM, (2) tunneling, and (3) motion through a scaffold. According to Rus and Vona (2001), this second mode of movement is advantageous compared to surface relocation because SR through the volume of the robots generally requires $O(n)$ fewer moves than by surface motions. The literature seems to evidence that internal motions allow for higher degrees of parallelism, at least using scaffolding or unit-compression, as it eases the avoidance of motion blocking and collisions. The difficulty then becomes trajectory planning through the volume of the object and avoiding internal collisions. It however remains an open question whether metamorphic systems involving a very large number of modules and using internal movements of any sort could be physically realized in practice, as it might turn out to be impracticable to maintain a perfect module alignment for tunneling in massive ensembles, or build modules with sufficient elasticity and connector strength to safely allow motions through a scaffold.

3.7. Motion Parallelism and Convergence

As discussed earlier in this paper, sequential motions are highly prohibitive in medium-to-massive self-reconfiguring ensembles, as they tend to dramatically increase the duration of the reconfiguration process. Yet, it greatly simplifies planning and reduces uncertainty by making deadlocks and collisions virtually impossible. Convergence into a goal shape under these conditions therefore only depends on whether or not a satisfiable assembly order supports the process. We believe that there is an inescapable tradeoff to be made between achieving a great degree of motion parallelism and being able to guarantee the convergence of the system into the goal shape. Previous results seem to indicate that uncertainty due to collisions between modules, deadlocks, and local minima increases with the number of modules moving concurrently. Nonetheless, few works were able to truly quantify that effect in relation to their methods, as in (Yim et al., 2001) where a brief discussion on the convergence of the algorithm was provided. In the rest of the literature, the likelihood of convergence of the proposed methods remains unclear. While existing algorithm with parallel motions show a high variance in the amount of modules that are able to move concurrently, depending on

the method that is being used, reconfiguration works based on scaffolding techniques specifically tailored for massively parallel internal motions show great promise in allowing the parallel and collision-free motion of modules, as Lengiewicz and Holobut (2019) and Thalamy et al. (2019) have shown.

The only sequential modular robot self-reconfiguration algorithms covered here were proposed by Fitch et al. in the context of heterogeneous SR. While MeltSortGrow (Fitch et al., 2003) is truly sequential, *TunnelSort* (Fitch et al., 2007) and *ConstrainedTunnelSort* (Fitch et al., 2005) both could easily have modules move in parallel during their tunneling phase.

3.8. On the Complexity of Self-Reconfiguration

It has been noted in Section 2.2.1 that common metrics for self-reconfiguration algorithms are: *reconfiguration time*, usually expressed in number of time steps, and *number of moves*, which counts the total number of individual module motions required to complete the reconfiguration. However, by itself the *number of moves* is not sufficient to get a good sense of the performance of a given algorithm, as it does not convey enough information about the degree of parallelism that it achieves, which is a critical parameter for most reconfigurations on massive MSR. Therefore, reconfiguration time is plausibly the most important metric of self-reconfiguration, at it directly communicates the level of parallelism of the algorithm.

As pointed out in the introduction, researchers are not yet able to find the optimal number of moves or time for a given reconfiguration problem, but as shown in the previous section some algorithms have been able to achieve $O(n^2)$ number of moves both for homogeneous (Kawano, 2015; Vassilvitskii et al., 2002) and heterogeneous (Kawano, 2016, 2017; Fitch et al., 2003) modular robots, and even $O(n)$ moves (Støy, 2006; Lengiewicz and Holobut, 2019), potentially at the cost of a lack of a convergence guarantee as in (Yim et al., 2001). Furthermore, Michail et al. (2017) formally demonstrated that the 2D self-reconfiguration of systems with modules that could only perform rotations was much harder than with modules capable of both rotation and translation. This is evidently also the case for 3D systems.

An additional metric that provides information on the energetic cost of a reconfiguration along with the number of moves is the *number of messages* exchanged during the reconfiguration. It is seemingly of lesser importance than the number of moves as the energetic and time cost of sending a message is nearly negligible compared to the cost of actuation for movements. However intensive communication is often used as a way to simplify the

planning process and prevent physical collision and other undesirable events from occurring, which is likely to still cause a massive energetic overhead due to the sheer volume of transmiting messages. Furthermore, because of the immense size of such distributed systems in conjunction with the limited memory size of modules, excessive communications could quickly overload the message queues of the modules and therefore have a dramatic effect on reconfiguration as a consequence of the subsequent message losses (Naz et al., 2018).

3.9. Simulation Environments

Regarding the simulation environment generally used in the field to experimentally evaluate algorithms, we can distinguish three main trends: (i) Authors failing to specify their means of simulation; (ii) Authors developing simulators from scratch for their particular hardware model; (iii) Authors experimenting through more general-purpose simulators. One exception is Lengiewicz and Holobut (2019), who performed a numerical simulation through *Wolfram Mathematica*. The second group concerns the majority of aforementioned algorithms, more specifically where Yim, Støy, Rus, or Fitch are one of the co-authors, with simulation environment developed in *Java 3D* as the main underlying technology. Though it might be noted that *SRSim* (Fitch et al., 2003), developed by Robert Fitch has been used for a variety of self-reconfiguration and locomotion algorithms using the *SlidingCube* model and in one case (Fitch and McAllister, 2013) a model of the M-Tran hardware. Halfway between hardware specificity and general-purpose is *DPRSim*, used in (Dewey et al., 2008). It includes both a physical and graphical engine suited for millions of modules. Finally, there are a number of general-purpose simulators that have been effectively used in the context of SR even though they have not been used in any of the works represented here. These simulators are *ARGoS* (Pinciroli et al., 2012), which specializes in swarm robotics and supports a large number of hardware platforms, and *VisibleSim* (Dhoutaut et al., 2013; Piranda, 2016), a discrete-time step simulator for modular robotic ensembles that has been used extensively to demonstrate 2D self-reconfiguration and other distributed algorithms on a variety of hardware models.

3.10. Evaluation Methods

In the context of experimental evaluation of algorithms, the disparity does not stop at the simulation environments. Validation methods and self-

reconfiguration cases vary greatly across the presented algorithms, therefore making any attempt at comparison cumbersome.

There are two sets of SR instances that are commonly used to evaluate algorithms. The first serves more as a proof-of-concept and consists in the reconfiguration of an initial shape into a chair or table shape Støy (2006); Spröwitz et al. (2010); Vassilvitskii et al. (2002); Fitch et al. (2003); Kawano (2015, 2017), while the other is a set of reconfiguration problems introduced by Yim et al. (2001) and used in Støy (2006). It involves three reconfiguration cases from a plane into a disk, solid ball, hollow ball, or cup. The initial plane represents a *maximal constraint-free connected overlap* with the goal configuration, providing a maximal overlap between initial and goal shape without blocking constraints on exterior modules. These reconfigurations can be performed at various numbers of modules and cover all possible classes of goal shapes—i.e., convex, concave, solid, and hollow shapes. Finally, works on heterogeneous reconfiguration by Fitch et al. were evaluated with heterogeneous volumes made of a gradient of module types, that they would reverse under various environmental constraints (Fitch et al., 2003, 2005, 2007). As discussed previously on the topic of complexity (Section 3.8), researchers have been mainly interested in quantifying the number of individual module movements required by the aforementioned reconfigurations, as well as total reconfiguration time; alas, very few works mentioned the number of messages.

In term of the scale of experiments, most works used hundreds to dozens of hundreds modules. Few works have demonstrated simulations involving thousands to million of modules. Nonetheless, Dewey et al. (2008) did so by assembling a trumpet made from 1 million meta-modules and a complex building consisting of 10 millions meta-modules, from an initial cuboid.

3.11. Validation Methods and Analyses

Researchers also resort to formal analysis as a way to validate their algorithms, and for further demonstrating the particular capabilities of their methods—under their various assumptions. Ideally, reconfiguration algorithms should be able to demonstrate both *correctness* and *completeness*. The former implies that it will produce a motion plan that reconfigures any initial configuration into any goal configuration (*partial correctness*), and is guaranteed to terminate (*total completeness*, while the latter means that any well formed reconfiguration problem can be performed. We will also use the term *partial completeness* to discuss algorithm which are provably

Table 1: Summary of complexity analyses and proofs provided in *Top-Down* works. Missing works did not provide any item. *(partial)* means that completeness was proved for a limited class of reconfigurations.

Work	**Proved Correctness**	**Proved Completeness**	**Analysed Complexity**
(Butler and Rus, 2003)	x	(partial)	
(Dewey et al, 2008)	x		
(Fitch et al, 2003)	x	x	x
(Fitch et al, 2005)		(partial)	x
(Fitch et al, 2007)	x		x
(Kawano, 2015)	x	x	x
(Kawano, 2016)	x	x	x
(Kawano, 2017)	x	x	x
(Vassilvitskii et al, 2002)	x	x	x

complete only for a limited class of reconfigurations. Table 1 summarizes the formal analyses provided by *Top-Down*. It should be noted that most algorithms that could demonstrate completeness were able to do so because they mostly rely on sequential motions at some point during reconfiguration, or only proved *partial completeness*. Indeed, it is worth noting once again that parallel motions lead to an increased entropy in the reconfiguration process, hence preventing an easy access to provable properties of the algorithms. This is an additional argument for the more thorough experimental evaluation methods that we discuss in next section, as they may be the only way to accurately assess the performance of highly parallel and distributed reconfiguration algorithms.

4. Discussion on Programmable Matter

In this section we further discuss the state of the art of self-reconfiguration methods in MSR, but from the vantage point of our vision of programmable matter. We analyze how previous research efforts compare against the specific requirements of MSR-based programmable matter, and what perspectives can be envisioned for the future of the field. In our vision, PM is made of potentially millions of very restricted sub-mm micro-electro-mechanical-systems (MEMS) modules, with limited embedded computation used for communication and manipulating actuators, themselves supporting adherence and locomotion (Bourgeois et al., 2016).

4.1. Self-Reconfiguration Criteria

There are a number of essential properties that we argue self-reconfiguration algorithms for PM should have:

- **Lattice-Based**: An organization of modules in a lattice appears to be the most advantageous solution, as its highly regular structure allows for an easier positioning of modules thanks to its discretization of the space, compared to a chain arrangement. Yet, maintaining a regular lattice structure on MSR consisting of up to millions of modules might turn out problematic in practice due to repeated imperceptible misalignments and irregularities in the geometry of the modules, seemingly inconsequential problems which would be dramatically magnified by the size of the system. Alternative architectures such as irregular lattices might therefore need to be devised and studied.

- **Distributed**: As SR is a computationally intense process and PM is required to be autonomous, software solutions will need to be distributed in order to be independent from any controlling external entity and mitigate the computational load on individual modules. Furthermore, as previously pointed out, centralized methods do not offer any robustness to failure, a critical aspect of PM as explained below.

- **Homogeneity**: Since programmable matter is generally thought of as massive ensembles of mass-producible and interchangeable modular robotic units, it is evident that reconfiguration frameworks should operate on homogeneous MSR.

- **Motion Parallelism**: With scalability as the main concern of software methods for PM, a high degree of parallelism is required. Algorithms should thus aim to maximize simultaneous module movements. Consequently, the primary performance metric for SR is arguably reconfiguration time, as discussed in the previous discussion on complexity.

- **Reliability**: It has already been noted that there is a compromise to be made between parallelism and ease of convergence into the goal reconfiguration. One can think of extreme examples with on the one hand large colonies of ants attempting to build a bridge with a substantial failure rate but with nearly all the agents involved acting concurrently, and on the other hand, slow and steady sequential Lego-like

construction tasks where convergence is assured at the cost of a reduced building speed. Building objects made of programmable matter would nevertheless require a high degree of confidence in the success of the reconfiguration. A high fidelity and resolution is also important, potentially with a very slight tolerance for misplaced modules in some applications of the technology.

- **Robustness**: Robustness will have to be an essential property of self-reconfiguration algorithms for PM, as faults are almost guaranteed to occur during the reconfiguration of systems comprising millions of individual units.

Besides, the network aspect of the underlying hardware on which is based our vision of PM has to be carefully taken into account, as it has been shown that large lattice-based distributed systems relying exclusively on neighbor-to-neighbor communications are particularly at risk of latency and reliability issues. This is a result of the huge diameter of such systems coupled with a network high average distance, which together pose a serious design challenge to prospective algorithmic solutions (Naz et al., 2018).

4.2. Relevance of Existing Works

When confronting these requirements to the works discussed in the earlier sections, it comes to light that existing SR methods have yet to satisfy them all. What seems to stand out from this analysis is that current algorithms are either relying too heavily on the characteristics of specific hardware systems (in the *Bottom-Up* approach), or are making impractical assumptions on the abilities of the underlying hardware, such as unreasonably powerful sensors or over-simplistic motion primitives (in the *Top-Down* approach).

The relationship between hardware and software poses a number of issues as it is quite challenging to find a common ground between the two, since designing powerful hardware at the micro or nano scale is difficult, while on the other hand it is very difficult (if not impossible) to solve problems with hardware that is very primitive. We therefore find that the different communities discussed in this paper each have a crucial role to play in solving this problem. The theory community is interested in solving problems with the minimum effective hardware, and thus informs other communities of the expected performance of a given task for different hardware capabilities. The robotics community is interested in producing capable hardware at a scale as

small as possible, and informs other communities of what can be expected in term of the capabilities of the models and their mode of motion. Finally, the researchers interested in software methods for these metamorphic systems can compose solutions according to this set of information.

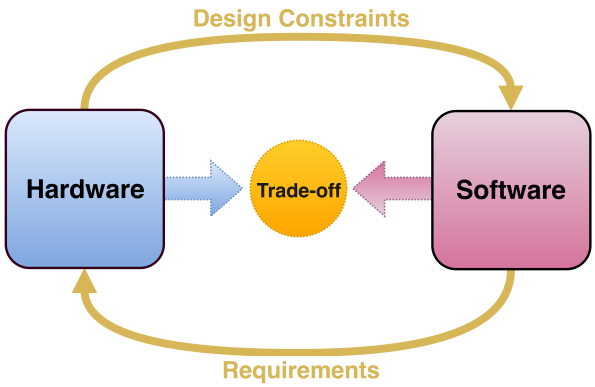

Figure 8: Interrelationship of hardware and software programmable matter components. (Best seen online with colors.)

Thus, we believe that viable solutions to self-reconfiguration in the context of programmable matter will necessary be the result of a compromise between *Bottom-Up* and *Top-Down* approaches (see Figure 8), in that they will need to factor in the design, production, and integration constraints imposed by the hardware, while also meeting the requirements requested by the software. These two mutually dependent classes of prerequisites will therefore need to converge for practical solutions to emerge.

4.3. Perspectives

We hint below at some open problems related to modular robot self-reconfiguration that will need to be tackled in the coming years in order to progress towards practical reconfiguration solutions.

Firstly, we suspect that there may be no one-size-fits-all self-reconfiguration solution, where a single method can offer both completeness and near-optimal performance. Perhaps the best approach to self-reconfiguration could involve having a large set of algorithms specifically tailored for a very particular class of reconfiguration problems, that are autonomously selected depending on the characteristics of the current problem. If that is the case, designing software methods for classifying reconfiguration problems

(based on common features between initial and goal configurations, or lack thereof) as well as for dynamically selecting the best method for solving this problem among a library of methods, will certainly be a challenging issue. This is similar to the concept of *hyper-heuristics*, that defines algorithms for autonomously selecting the most efficient *heuristic* for the current problem, usually using artificial intelligence approaches.

Also, while Yim et al. (2001) have proposed the *maximal constraint-free connected overlap* between initial and goal configurations, additional overlapping patterns could be designed in order to simplify the subsequent reconfiguration process, or optimize for some predetermined metric—e.g., motion parallelism for minimizing reconfiguration time, number of movements and/or messages for minimizing the energetic impact.

It has also been pointed out that mechanical constraints—gravity and connector stress in particular—will have to be carefully considered in future reconfiguration methods before large metamorphic systems can be physically realized. These additional constraints would potentially limit the range of possible reconfigurations (depending on the underlying hardware), while also greatly reducing the size of the search space due to impractical intermediate configurations and unsafe module motions. It would be interesting to see how distributed mechanical computation methods such as in (Hołobut and Lengiewicz, 2017) could be sensibly integrated into the reconfiguration process, and at what cost.

Then, algorithms with built-in robustness and fault recovery abilities should be further studied, as these are also essential properties required by practical applications of SR. Some approaches might require a pool of additional modules ready to be summoned into the reconfiguration process to replace faulty units (as well as procedures for discarding them), or emergent ways to approximate a desired configuration with a reduced number of resource modules.

Furthermore, though current SR methods already offer attractive performance bounds, solutions often rely on possibly impractical assumptions—e.g., the presence of powerful embedded sensors on modules for collision and deadlock avoidance, scaffold-less tunneling, a disregard for low-level planning, or virtual modules communicating with physical modules in the vicinity of unfilled goal positions. Finding out if the current level of performance can be preserved when some of these assumptions are relaxed is essential.

Finally, we find that present experimental evaluations used in the literature are unsatisfying for clearly reporting on the capabilities of the proposed

methods (e.g., classes of shapes that can or cannot be formed, scalability, convergence reliability), which makes comparison between methods cumbersome and inaccurate. Therefore, we argue that there is a need for a standardized benchmark for self-reconfiguration algorithms with a common purpose, that covers a wide-range of carefully chosen reconfiguration scenarios under varying constraints (and possibly random configurations). We can already identify a number of pitfalls to its design, as self-reconfiguration can be defined in various ways, depending on whether or not the positioning of the goal configuration is constrained, whether an initial overlap between configurations is required, etc. These initial assumptions have a considerable impact on reconfiguration and make for additional parameters to be taken into account.

5. Conclusion

In this paper, we examined the literature on self-reconfiguration methods in three-dimensional lattice-based modular robotic and self-organizing particle systems. From this analysis, we attempt to provide an extensive survey of the current state of the art on these topics, with a thorough study of its strengths and weaknesses from the standpoint of its application to programmable matter. From more than two dozens of those published works, that we classified into three different approaches to the self-reconfiguration problem, we aim to dispense a comprehensive understanding of current challenges and directions to present and future research on this problem. Although several areas of improvement have been proposed, the inclusion of mechanical constraints and an emphasis on the robustness of reconfiguration methods stand out as particularly important for the prospect of realizing practical robot-based programmable matter systems. Furthermore, from the assessment that the hardware and software problems of self-reconfiguration cannot be considered in isolation, we argue that the three different approaches uncovered in Section 2 will have to merge in order to converge into practical solutions. Lastly, we contend that any progress in the field can only be the result of meaningful comparisons between proposed solutions, hence supporting the necessity for a unified evaluation system for existing and future self-reconfiguration algorithms.

Acknowledgment

This work was partially supported by the ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, ISITE-BFC project (ANR-15-IDEX-03), Labex ACTION program (ANR-11-LABX-01-01), and the Mobilitech project.

Ahmadzadeh, H., Masehian, E., 2015. Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization. Artificial Intelligence 223, 27–64. URL: http://linkinghub.elsevier.com/retrieve/pii/S0004370215000260, doi:10.1016/j.artint.2015.02.004.

Ahmadzadeh, H., Masehian, E., Asadpour, M., 2016. Modular Robotic Systems: Characteristics and Applications. Journal of Intelligent & Robotic Systems 81, 317–357. URL: http://link.springer.com/10.1007/s10846-015-0237-8, doi:10.1007/s10846-015-0237-8.

Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R., 2006. Computation in networks of passively mobile finite-state sensors. Distributed Computing 18, 235–253. URL: http://link.springer.com/10.1007/s00446-005-0138-3, doi:10.1007/s00446-005-0138-3.

Bojinov, H., Casal, A., Hogg, T., 2000. Emergent structures in modular self-reconfigurable robots, IEEE. pp. 1734–1741. URL: http://ieeexplore.ieee.org/document/844846/, doi:10.1109/ROBOT.2000.844846.

Bourgeois, J., Piranda, B., Naz, A., Boillot, N., Mabed, H., Dhoutaut, D., Tucci, T., Lakhlef, H., 2016. Programmable matter as a cyber-physical conjugation, in: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on, IEEE. pp. 002942–002947. URL: http://ieeexplore.ieee.org/document/7844687/, doi:10.1109/SMC.2016.7844687.

Butler, Z., Rus, D., 2003. Distributed Planning and Control for Modular Robots with Unit-Compressible Modules. The International Journal of Robotics Research , 699–715URL: https://doi.org/10.1177/02783649030229002, doi:10.1177/02783649030229002.

Campbell, J., Pillai, P., 2008. Collective Actuation. The International Journal of Robotics Research 27, 299–314. URL: http://journals.sagepub.com/doi/10.1177/0278364907085561, doi:10.1177/0278364907085561.

Cannon, S., Daymude, J.J., Randall, D., Richa, A.W., 2016. A Markov Chain Algorithm for Compression in Self-Organizing Particle Systems, in: Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, ACM Press. pp. 279–288. URL: http://dl.acm.org/citation.cfm?doid=2933057.2933107, doi:10.1145/2933057.2933107.

Daymude, J.J., Hinnenthal, K., Richa, A.W., Scheideler, C., 2019. Computing by Programmable Particles, in: Flocchini, P., Prencipe, G., Santoro, N. (Eds.), Distributed Computing by Mobile Entities. Springer International Publishing, Cham. volume 11340, pp. 615–681. URL: http://link.springer.com/10.1007/978-3-030-11072-7_22, doi:10.1007/978-3-030-11072-7_22.

Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T., 2015a. An Algorithmic Framework for Shape Formation Problems in Self-Organizing Particle Systems, in: Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication, pp. 1–2. URL: http://doi.acm.org/10.1145/2800795.2800829, doi:10.1145/2800795.2800829.

Derakhshandeh, Z., Gmyr, R., Richa, A.W., Scheideler, C., Strothmann, T., 2016. Universal Shape Formation for Programmable Matter, in: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 289–299. URL: http://dl.acm.org/citation.cfm?doid=2935764.2935784, doi:10.1145/2935764.2935784.

Derakhshandeh, Z., Gmyr, R., Strothmann, T., Bazzi, R., Richa, A.W., Scheideler, C., 2015b. Leader Election and Shape Formation with Self-organizing Programmable Matter, in: DNA Computing and Molecular Programming. Springer International Publishing, Cham. volume 9211, pp. 117–132. URL: http://link.springer.com/10.1007/978-3-319-21999-8_8, doi:10.1007/978-3-319-21999-8_8.

Dewey, D.J., Ashley-Rollman, M.P., Rosa, M.D., Goldstein, S.C., Mowry, T.C., Srinivasa, S.S., Pillai, P., Campbell, J., 2008. Generalizing metamodules to simplify planning in modular robotic systems, in: Intelligent

Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, pp. 1338–1345. doi:10.1109/IROS.2008.4651094.

Dhoutaut, D., Piranda, B., Bourgeois, J., 2013. Efficient Simulation of Distributed Sensing and Control Environments, in: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, IEEE. pp. 452–459. URL: http://ieeexplore.ieee.org/document/6682107/, doi:10.1109/GreenCom-iThings-CPSCom.2013.93.

Di Luna, G.A., Flocchini, P., Prencipe, G., Santoro, N., Viglietta, G., 2018a. Line Recovery by Programmable Particles, in: Proceedings of the 19th International Conference on Distributed Computing and Networking, ACM Press. pp. 1–10. URL: http://dl.acm.org/citation.cfm?doid=3154273.3154309, doi:10.1145/3154273.3154309.

Di Luna, G.A., Flocchini, P., Santoro, N., Viglietta, G., Yamauchi, Y., 2018b. Shape Formation by Programmable Particles, in: Aspnes, J., Bessani, A., Felber, P., Leitão, J. (Eds.), 21st International Conference on Principles of Distributed Systems (OPODIS 2017), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany. pp. 31:1–31:16. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8637, doi:10.4230/LIPIcs.OPODIS.2017.31.

Doty, D., 2012. Theory of algorithmic self-assembly. Communications of the ACM 55, 78. URL: http://dl.acm.org/citation.cfm?doid=2380656.2380675, doi:10.1145/2380656.2380675.

Fitch, R., Butler, Z., 2007. Scalable locomotion for large self-reconfiguring robots, in: Robotics and Automation, 2007 IEEE International Conference on, IEEE. pp. 2248–2253. 00000.

Fitch, R., Butler, Z., Rus, D., 2003. Reconfiguration planning for heterogeneous self-reconfiguring robots, in: Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on, pp. 2460–2467. doi:10.1109/IROS.2003.1249239.

Fitch, R., Butler, Z., Rus, D., 2005. Reconfiguration Planning Among Obstacles for Heterogeneous Self-Reconfiguring Robots, in: Robotics and Au-

tomation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, pp. 117–124. doi:10.1109/ROBOT.2005.1570106.

Fitch, R., Butler, Z., Rus, D., 2007. In-Place Distributed Heterogeneous Reconfiguration Planning, in: Distributed Autonomous Robotic Systems 6, pp. 159–168. doi:10.1007/978-4-431-35873-2_16.

Fitch, R., McAllister, R., 2013. Hierarchical Planning for Self-reconfiguring Robots Using Module Kinematics, in: Distributed Autonomous Robotic Systems 10, pp. 477–490. doi:10.1007/978-3-642-32723-0_34.

Fitch, R.C., 2004. Heterogeneous self-reconfiguring robotics. Ph.D. thesis. Darthmouth College.

Fukuda, T., Kawauchi, Y., 1990. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator, IEEE Comput. Soc. Press. pp. 662–667. URL: http://ieeexplore.ieee.org/document/126059/, doi:10.1109/ROBOT.1990.126059.

Gmyr, R., Kostitsyna, I., Kuhn, F., Scheideler, C., Strothmann, T., 2017. Forming Tile Shapes with a Single Robot. Technical Report.

Gorbenko, A.A., Popov, V.Y., 2012. Programming for modular reconfigurable robots. Programming and Computer Software 38, 13–23. URL: https://doi.org/10.1134/S0361768812010033, doi:10.1134/S0361768812010033.

Holobut, P., Kursa, M., Lengiewicz, J., 2014. A class of microstructures for scalable collective actuation of Programmable Matter, IEEE. pp. 3919–3925. URL: http://ieeexplore.ieee.org/document/6943113/, doi:10.1109/IROS.2014.6943113.

Hou, F., Shen, W.M., 2010. On the complexity of optimal reconfiguration planning for modular reconfigurable robots, in: Robotics and Automation (ICRA), 2010 IEEE International Conference on. doi:10.1109/ROBOT.2010.5509642.

Hołobut, P., Lengiewicz, J., 2017. Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning, in: Robotics and Automation (ICRA), 2017 IEEE International Conference on, pp. 2103–2109. doi:10.1109/ICRA.2017.7989242.

Kawano, H., 2015. Complete reconfiguration algorithm for sliding cube-shaped modular robots with only sliding motion primitive, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pp. 3276–3283. doi:10.1109/IROS.2015.7353832.

Kawano, H., 2016. Full-resolution reconfiguration planning for heterogeneous cube-shaped modular robots with only sliding motion primitive, in: Robotics and Automation (ICRA), 2016 IEEE International Conference on, pp. 5222–5229. doi:10.1109/ICRA.2016.7487730.

Kawano, H., 2017. Tunneling-based self-reconfiguration of heterogeneous sliding cube-shaped modular robots in environments with obstacles, in: Robotics and Automation (ICRA), 2017 IEEE International Conference on, pp. 825–832. doi:10.1109/ICRA.2017.7989100.

Knaian, A.N., Cheung, K.C., Lobovsky, M.B., Oines, A.J., Schmidt-Neilsen, P., Gershenfeld, N.A., 2012. The Milli-Motein: A self-folding chain of programmable matter with a one centimeter module pitch, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, Vilamoura-Algarve, Portugal. pp. 1447–1453. URL: http://ieeexplore.ieee.org/document/6385904/, doi:10.1109/IROS.2012.6385904.

Kotay, K.D., Rus, D.L., 2000. Algorithms for self-reconfiguring molecule motion planning, in: Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on, pp. 2184–2193. doi:10.1109/IROS.2000.895294.

Lengiewicz, J., Holobut, P., 2019. Efficient collective shape shifting and locomotion of massively-modular robotic structures. Auton. Robots 43, 97–122. URL: https://doi.org/10.1007/s10514-018-9709-6, doi:10.1007/s10514-018-9709-6.

Michail, O., Skretas, G., Spirakis, P.G., 2017. On the Transformation Capability of Feasible Mechanisms for Programmable Matter. arXiv:1703.04381 [cs] URL: http://arxiv.org/abs/1703.04381. arXiv: 1703.04381.

Naz, A., Piranda, B., Tucci, T., Copen Goldstein, S., Bourgeois, J., 2018. Network Characterization of Lattice-Based Modular Robots with Neighbor-to-Neighbor Communications, in: Distributed Autonomous

Robotic Systems, Springer International Publishing, Cham. pp. 415–429. URL: http://link.springer.com/10.1007/978-3-319-73008-0_29.

Patitz, M.J., 2014. An introduction to tile-based self-assembly and a survey of recent results. Natural Computing 13, 195–224. URL: http://link.springer.com/10.1007/s11047-013-9379-4, doi:10.1007/s11047-013-9379-4.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M., 2012. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6, 271–295. URL: http://link.springer.com/10.1007/s11721-012-0072-5, doi:10.1007/s11721-012-0072-5.

Piranda, B., 2016. VisibleSim: Your simulator for Programmable Matter, in: Algorithmic Foundations of Programmable Matter (Dagstuhl Seminar 16271). Dagstuhl.

Piranda, B., Bourgeois, J., 2018. Designing a quasi-spherical module for a huge modular robot to create programmable matter. Autonomous Robots URL: http://link.springer.com/10.1007/s10514-018-9710-0, doi:10.1007/s10514-018-9710-0.

Piranda, B., Laurent, G.J., Bourgeois, J., Clévy, C., Möbes, S., Fort-Piat, N.L., 2013. A new concept of planar self-reconfigurable modular robot for conveying microparts. Mechatronics 23, 906–915. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957415813001633, doi:10.1016/j.mechatronics.2013.08.009.

Rubenstein, M., Ahler, C., Nagpal, R., 2012. Kilobot: A low cost scalable robot system for collective behaviors, in: 2012 IEEE International Conference on Robotics and Automation, IEEE, St Paul, MN, USA. pp. 3293–3298. URL: http://ieeexplore.ieee.org/document/6224638/, doi:10.1109/ICRA.2012.6224638.

Rus, D., Vona, M., 2001. Crystalline Robots: Self-Reconfiguration with Compressible Unit Modules. Autonomous Robots 10, 107–124. URL: https://doi.org/10.1023/A:1026504804984, doi:10.1023/A:1026504804984.

Spröwitz, A., Laprade, P., Bonardi, S., Mayer, M., Moeckel, R., Mudry, P.A., Ijspeert, A.J., 2010. Roombots—Towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules, in: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. doi:10.1109/IROS.2010.5649504.

Stoy, K., Nagpal, R., 2004. Self-repair through scale independent self-reconfiguration, in: Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on, IEEE. pp. 2062–2067. URL: http://ieeexplore.ieee.org/abstract/document/1389701/.

Støy, K., 2006. Using cellular automata and gradients to control self-reconfiguration. Robotics and Autonomous Systems 54, 135 – 141. URL: http://www.sciencedirect.com/science/article/pii/S0921889005001521, doi:https://doi.org/10.1016/j.robot.2005.09.017.

Støy, K., Nagpal, R., 2007. Self-Reconfiguration Using Directed Growth, in: Distributed Autonomous Robotic Systems 6, pp. 3–12. URL: https://doi.org/10.1007/978-4-431-35873-2_1, doi:10.1007/978-4-431-35873-2_1.

Thalamy, P., Piranda, B., Bourgeois, J., 2019. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure, in: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, Montreal QC, Canada. pp. 140–148.

Tucci, T., Piranda, B., Bourgeois, J., 2017. Efficient Scene Encoding for Programmable Matter Self-reconfiguration Algorithms, in: Proceedings of the Symposium on Applied Computing, pp. 256–261. URL: http://doi.acm.org/10.1145/3019612.3019706, doi:10.1145/3019612.3019706.

Tucci, T., Piranda, B., Bourgeois, J., 2018. A Distributed Self-Assembly Planning Algorithm for Modular Robots, in: International Conference on Autonomous Agents and Multiagent Systems) (AAMAS), Association for Computing Machinery (ACM), Stockholm, Sweden.

Vassilvitskii, S., Yim, M., Suh, J., 2002. A complete, local and parallel reconfiguration algorithm for cube style modular robots, in: Robotics and

Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on, pp. 117–122 vol.1. doi:10.1109/ROBOT.2002.1013348.

Woods, D., Chen, H.L., Goodfriend, S., Dabby, N., Winfree, E., Yin, P., 2013. Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time. arXiv:1301.2626 [cs] URL: http://arxiv.org/abs/1301.2626. arXiv: 1301.2626.

Yim, M., Zhang, Y., Lamping, J., Mao, E., 2001. Distributed Control for 3d Metamorphosis. Autonomous Robots 10, 41–56. URL: https://doi.org/10.1023/A:1026544419097, doi:10.1023/A:1026544419097.

Yoshida, E., Murata, S., Kurokawa, H., Tomita, K., Kokaji, S., 1998. A distributed method for reconfiguration of a three-dimensional homogeneous structure. Advanced Robotics 13. doi:10.1163/156855399X00234.

Zhu, Y., Bie, D., Wang, X., Zhang, Y., Jin, H., Zhao, J., 2017. A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata. Journal of Parallel and Distributed Computing 102, 80 – 90. URL: http://www.sciencedirect.com/science/article/pii/S0743731516301824, doi:https://doi.org/10.1016/j.jpdc.2016.11.016.

Ünsal, C., Khosla, P.K., 2001. A multi-layered planner for self-reconfiguration of a uniform group of I-Cube modules, in: Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on, pp. 598–605. doi:10.1109/IROS.2001.973421.

Ünsal, C., Kiliççöte, H., Khosla, P.K., 2001. A Modular Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning. Autonomous Robots 10, 23–40. URL: https://doi.org/10.1023/A:1026592302259, doi:10.1023/A:1026592302259.

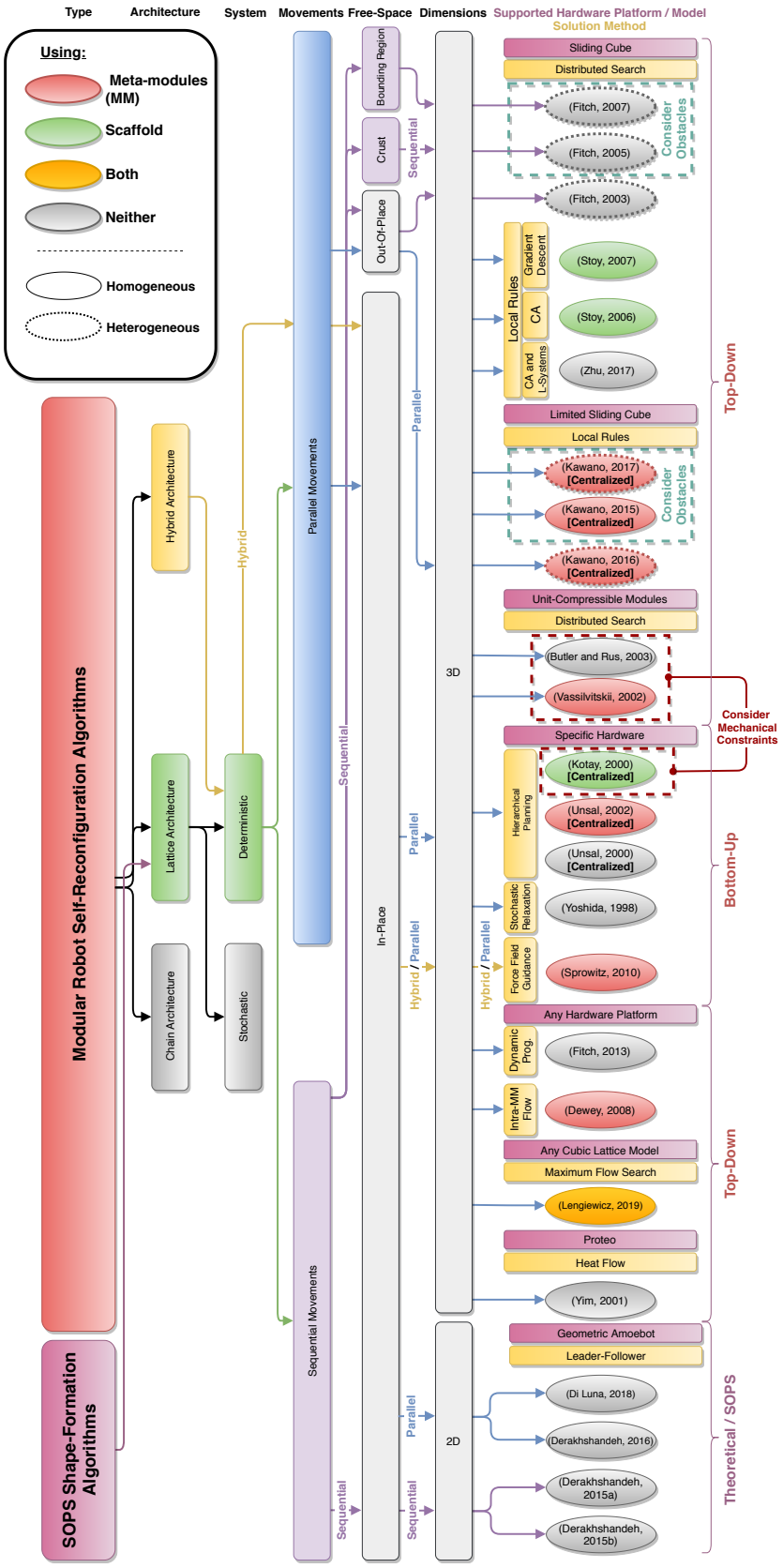

Figure 7: Overview of modular robotic and particle system self-reconfiguration methods. (Best seen online with colors.)