# On Self-Optimized Self-Assembling of Heterogeneous Multi-robot Organisms

Serge Kernbach, Benjamin Girault, and Olga Kernbach

**Abstract.** This chapter is devoted to a bio-inspired self-assembling of heterogeneous robot modules into specific topological configurations. The approach involves several algorithmic inspirations from biological regulatory networks for achieving environmental dependability and considers constraint-based optimization techniques for finding optimal connections between heterogeneous modules. Scalability and locality of sensor information are addressed.

## 1 Introduction

Self-assembling is an important process, where a disordered set of existing components aggregates into a well-ordered structure by following simple assembling rules [1]. Such a process takes place on macro- or micro- levels without external guidance by utilizing several self-organizing mechanisms. A great source of inspiration for self-assembling algorithms is a molecular self-assembling, which appears in forms of e.g. crystals, colloids, or self-assembled monolayers [2]. Macroscopic self-assembling is primarily related to a robot research, where robot modules aggregate into complex topological structures to achieve a flexible functionality [3].

A substantial difference between micro- and macro- self-assembling consists in the size and capabilities of components as well as in the appearing problems and challenges. On the micro-level such components are molecules utilizing chemical bounding forces, whereas on the macro-level, components are robot modules, capable of docking with each other [4]. These modules can have very simple form [5], or possess several cognitive features, such as sensing, objects detection/recognition, actuation, communication and others [6]. Microscopic and

Serge Kernbach · Benjamin Girault · Olga Kernbach
Institute of Parallel and Distributed Systems, University of Stuttgart,
Universitätstr. 38, 70569 Stuttgart, Germany
e-mail: {serge.kernbach,giraulbn,
olga.kernbach}@ipvs.uni-stuttgart.de

macroscopic self-assembling targets also different objectives. When self-assembling on the micro-level is normally a large-scale phenomenon appearing as a uniform structural pattern [7], on the macro-level we are interested primarily in creating dedicated low-scale structures with a desired functionality. Typically, these structures are aggregated robots, which demonstrate locomotive functionality in a legged, clambering or rolling form [8].

Despite differences, micro- and macro- self-assembling shares also several common problems. One of them is a recruitment of new elements for assembling, where we can find several bio- and chemo- inspired works [9] leading to an efficient recruitment strategy [10]. However, the major effort on both levels is related to a guided self-assembling [11]: formation into a desired final form and, as a consequence, a need of multiple optimization steps, required to obtain such a form. Artificial programmability and self-optimization can be addressed in several ways: we used here some ideas from gene regulatory networks [12] and its multiple algorithmic inspirations, e.g. [13], [14]. Another interesting scientific challenge is the distribution of self-regulating mechanisms and integration of multiple constraints, appearing during the assembling process.

This chapter is based on the previous works [15], [16] and extends them towards two-steps optimization, which happen during the expression of high-level topological descriptions into a concrete configuration and an assembling of modules into this configuration. Sec. 2 gives a common pictures of self-assembling procedure and discusses main optimization steps. Sec. 3 introduces several constraints, which appear during robot-robot assembling, and Sec. 4 considers constraint-optimization approach. In Sec. 5 we focus on specific tasks such as grouping, or scaling of topologies. Finally, Sec. 6 demonstrates some results and Sec. 7 concludes this work.

## 2 General Self-Assembling Scenario

Problems of robot self-assembling and self-disassembling are well-known in reconfigurable robotics, see e.g. [17] or [18]. Here several high- and low- dimensional approaches [15],[19] are distinguished. To be more strong in definitions, we define self-assembling as a process, where robot modules $R_i$ establish multiple bilateral connections $R_k : R_p$ (denoted as docking between modules $R_k$ and $R_p$, see Fig. 1(a)), which step by step lead finally to an appearance of the topology $\Phi$.

Each topology $\Phi$ has some macroscopic functionality; more exactly, each particular connection $R_k : R_p$ introduces a degree of freedom (DoF) $\varphi_i$. As shown in Fig. 1(b), each connection between modules introduces only one DoF; by a combination of all DoF, an organism is capable of a collective movement. We express interactions between all $\varphi_i$ as a macroscopic functionality $F$ of an artificial organism. Figs. 1(c) and 1(d) provide two examples of such rotating and wheeled macroscopic functionalities. The relationship between $\Phi$ and $F$ is complex and depends not only on the involved DoFs, but also on the environment. We discuss these issues in the second comment below.

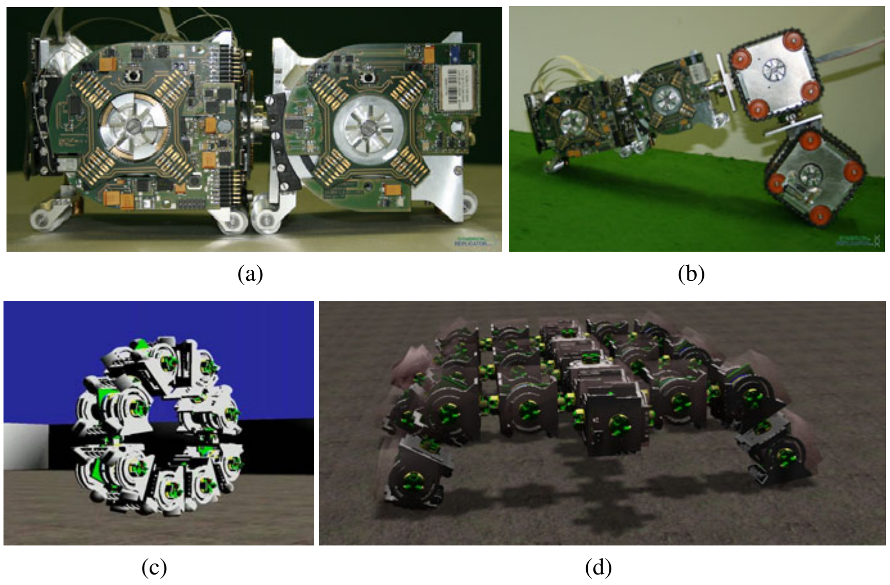


**Fig. 1** **(a)** Examples of a bilateral connection $R_k : R_p$; **(b)** Several 1DoF connections produce a macroscopic locomotion; **(c,d)** Two examples of macroscopic functionalities, achieved by many robot modules.

The process of self-assembling can be split on different steps, which are sketched in Fig. 2. Firstly, different types of robot modules $R_i$ and other objects (such as energy "cubes") are placed on the arena, see Fig. 2(a). In this stage all robots exist in the so-called swarm mode, i.e. as independent robots. On the second step, they select from possible existing topologies a small subset $\Phi^S$, which is optimal for given environment conditions. Such modules, which are needed for these topologies, group in some area of arena, see Fig. 2(b). This grouping approach is described in Sec. 5.1. After this first optimization, robots perform the second optimization, where a set of $\Phi^S$ is reduced to one $\Phi$. Robots take into account the number of modules in the aggregation site, their capabilities and availability for the desired functionality $F$. Finally, the chosen modules queue up in a right spatial order for docking, Fig. 2(c), and dock into an organism $\Phi$, see Fig. 2(d). Here, robots utilize different recruitment approaches, e.g. [10], to add a robot into an existing topology. After docking, all robot modules exist in the so-called organism mode, i.e. they are co-dependent on each other.

The general scenario, shown in Fig. 2, indicates only the main stages during the real robot self-assembling. We have to make two comments for this scenario.

**1. On-line/Off-line issue.** The first comment is related to the way of how to obtain the final topology $\Phi$. This approach originates from [15] and consists in the following idea. The set of pre-generated building blocks for patterns $\{\Phi\}$ can be maximized so that to cover the most functionally useful behaviors in different predictable environmental situations. For example, this can be performed by off-line

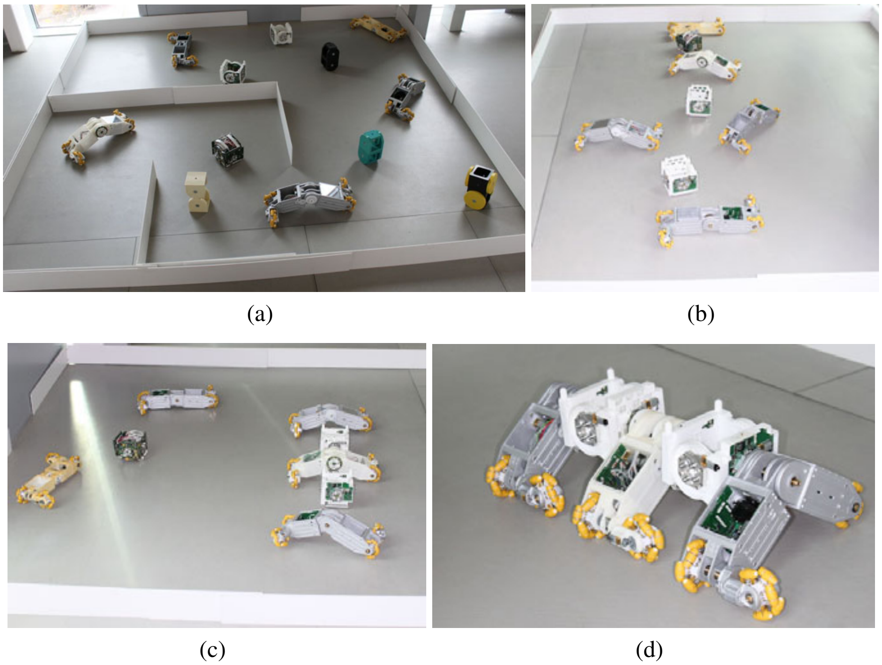


**Fig. 2** Different envisaged steps of the self-assembling scenario. **(a)** Initial stage – all robots are irregularly distributed on the arena; **(b)** Approaching of the selection modules for docking; **(c)** Ordering of modules in 2D disconnected form; **(d)** Docking of modules on 2D grid into an assembled organism and collective actuation into 3D state.

evolutionary simulation, by utilizing bio-inspired mechanisms from insect or animals, and in general is done *in advance*. These patterns can structurally be perturbed by a few modules *on-line* depending on the environment. This perturbation creates some deviation in the expected functionality and behavior, which can be handled by different on-line adaptive mechanisms. This finally reduces the set $\{\Phi\}$ to some $\Phi_1$, $\Phi_2$, ..., $\Phi_m$, which makes sense in a concrete environment. We will denote this smaller subset as $\Phi^S$. Thus, the problem of finding a specific solution is narrowed down to the problem of optimizing a deviation from one of the pre-generated patterns, for which all controlling mechanisms exist. Since a linear optimization is very fast, for example, the linear sum assignment problem is of $O(n^3)$ complexity [20], this approach can be run on-board and on-line.

**2. Topology, Functionality and Environment.** In general, the degrees of freedom $\varphi_i$ between robots $R_k : R_p$ depend on both $R_k$ and $R_p$, i.e., we can encounter the situation when both $R_k$, $R_p$ are relevant, one of them is relevant and none of them are relevant, see more in [16]. Moreover, a common functionality $F$ of an organism is determined by interactions between all $\varphi_i$. It is hardly possible to say in advance which functionality can be useful or not for a particular environment. There are two possibilities to explore a usefulness of functionalities: performing on/off-boar simulation and by trial-and-error approach with different topologies in real environment. Both have advantages and drawbacks and are used for finding $\Phi$. Essential scientific challenge here represents a programmability of self-assembling and its dependability on environment (constraints of robots). We need to find a balance between desirability (design goals) of topologies and capabilities to adapt to fluctuations of environment.

Thus, the self-assembling requires two-steps optimization:

- Environment-dependable generation of topologies from $\{\Phi\}$ to $\Phi^S$. We can denote this process as expression of topology from some general building-blocks-like description (by analogy with gene expression process) into a set of topologies with desired functional properties. Example of such approach is discussed in [15], where we defined a set of operators and basic elements for a generation of topologies.
- On the second step, $\Phi^S$ is optimized taking into account a *current* situation in terms of the number of robots in the assembling area, availability for docking, structure of local environment, assembling dynamics (for example amount of collisions) and other conditions. As a result, robots firstly produce a final topology $\Phi$, and secondly start a docking approach.

These two steps are demonstrated in Fig. 3. The whole approach consists of three functional parts ($F1-F3$) and three descriptive parts ($D1-D3$). Each functional part represents in fact a controller, running on-board of a robot module. It takes as input a corresponding description, executes all necessary activities and generates an output description. After this, the control over the robot(s) is passed to the next controller. Thus, descriptions $D1-D3$ are interfaces between controllers and controllers themselves work to some extent as operators over $D1-D3$. The main ideas of $F1$ and $F3$ controllers are already represented in [15] and [10], further we concentrate on the $F2$ optimization controller and introduce it in more detail.

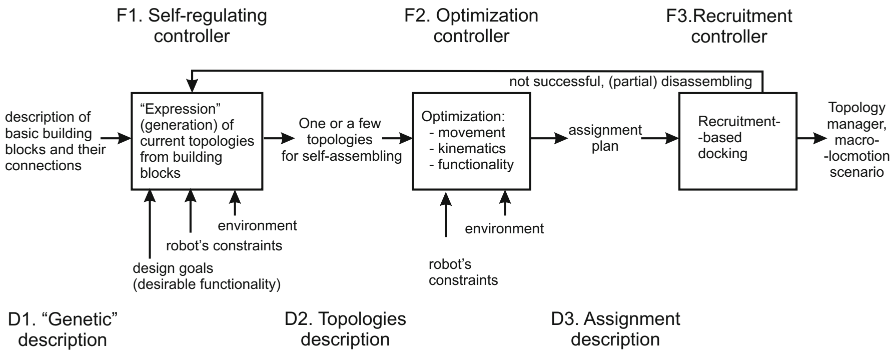

F1. Self-regulating controller
F2. Optimization controller
F3.Recruitment controller
not successful, (partial) disassembling
description of basic building blocks and their connections
"Expression" (generation) of current topologies from building blocks
One or a few topologies for self-assembling
Optimization: - movement - kinematics - functionality
assignment plan
Recruitment- -based docking
Topology manager, macro- -locmotion scenario
environment
robot's constraints
design goals (desirable functionality)
environment
robot's constraints
D1. "Genetic" description
D2. Topologies description
D3. Assignment description

**Fig. 3** General sketch of the self-assembling scenario.

## 3 Optimization Controller: Transition from $\Phi^S$ into $\Phi$ and the Role of Constraints

The self-assembling on the stage of optimization controller aims at several tasks: grouping of robots, identification and collection of constraints, selection of topologies, which satisfy these constraints, and finally a generation of one concrete topology $\Phi$ from a set of possible topologies $\Phi^S$. This is done in several ways. Firstly, $\Phi^S$ is compared with the current situation. For example, when other robots are already building some structure, this structure has a higher priority in the decision process. When there are no such structures, a robot compares current positions of known robots in order to decide which pattern is the most suitable for this configuration or which pattern is the most suitable for the given surface. When a pattern is selected, a robot solves the constrained assignment problem in order to determine a position in this pattern, where it goes to dock.

Secondly, when a robot decides about one of the patterns in $\Phi^S$, it is not constrained by this pattern, a robot can perturb the pattern $\phi_i$ by templates. This can happen only in two following cases. In the first case, the pattern, generated for $n$ robots, already has $n$ robots. The $n+1$ robot, joining to this organism, starts building a new scalability core. For example, when an organism with $n$ robots has four legs, the $n+1$ robot can start building additional legs. In the second case, based on observation in the environment, a robot can estimate a need of specific perturbations, e.g. to make legs longer to overstep some obstacles. However, the difficulty is that other robots may not know about this initiative and the whole pattern become desynchronized. Solution of this problem involves more communication between robots, as suggested in [15].

A topology $\Phi$ of an organism is represented by the connection placeholder $R_k : R_p$ and DoF functionality in this connection. For example in Fig 4, six modules are shown, this topology is described by five connection placeholders. In each $R_k : R_p : x$, the last "x" denotes the DoF: 1 means connection sidewards, 2 means forwards (there are only two ways to connect two modules $R_1$ and $R_2$). The generation of the mapping between robots and connection placeholders represents a classical constrained assignment problem. Choosing a partner $R_i$ for docking, i.e. the generation of $x^i_l \leftrightarrow x^j_k$ is independent of each other. It underlies several requirements: firstly it should satisfy local constraints $\upsilon_i$, secondly, each $x^i_l \leftrightarrow x^j_k$ pair has an associated local cost, and finally, the whole appearing topology has its own global costs.

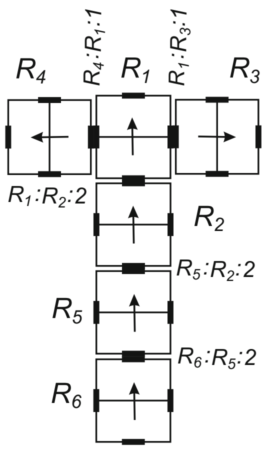


**Fig. 4** Topology of an organism, $R_k : R_p : x$ is a connection placeholder.

**1. Connectivity Constraints.** As mentioned, there are multiple constraints, imposed on connectivity, kinematic properties, heterogeneity and others. Generally, the connectivity means the number of elements, connected to each of modules. For example, the central element of the cross has the connectivity 4 (modules connected from each side). Connectivity constrains the number of connections and can be effectively utilized in a description of topologies. When $c_i$ is the connectivity of the $i$-element, where $i$ goes from 1 to $n$ ($n$ is the number of robots in the topology; in contrast $N$ is a total number of robots), the topology $\Phi$ can be described as $n+1$ set $(c_1, c_2, ..., c_n, c_t)$, $c_t$ is a total number of connections in the topology with $n$ robots. In general case, max. of $c_i$ is equal to the maximal connectivity of the platform. All $c_i$ are re-ordered from $c_{max}$ to $c_{min}$ so that the first element $c$ is always that one, which has a maximal degree of connectivity. The topology $\Phi$ can be described as

$$\Phi = \{c_{max}, c_{max-1} ..., c_{min+1}, c_{min}, c_t\}, c_i \in \{1,2,3,4\}. \tag{1}$$

The description, defined by (1) has different topological properties, see more in [16]. Generally, there are basic topologies, which are unique, provided the topology is coherent (coherent topology = no disconnected nodes). To eliminate disconnected topologies, a coherency constraint has to be integrated into *Constraint Satisfaction Problem* (*CSP*) and the *Constraint Optimization Problem* (*COP*) solver. Basic topologies can be perturbed by one or several modules, this increases $n$ and $c_t$. Such perturbed topologies are not unique. One of possible ways to deal with perturbed topologies is indicated in [15].

**2. Kinematic Constraints.** Topology $\Phi$ defined by (1) creates connections, which are invariant to robot's IDs. To integrate kinematics into topology, $\Phi$ should be supplemented with a functional description: it means to involve the desired degrees of freedom $\varphi_i$ for a particular connection. Since each node has max. four connections (i.e. in general case different $\varphi_i$), the functional topology should include all of them. We use the agreement, that when only one $\varphi$ is specified for a connectivity, it means $\varphi_i = \varphi$. Now we can generalize $\Phi$ from (1):

$$\Phi = ((c_{max} : \{\varphi\}_{max}), (c_{max-1} : \{\varphi\}_{max-1}), ..., (c_{min} : \{\varphi\}_{min}), c_t) \tag{2}$$

Thus, $\varphi$ defines a kinematic constraint imposed on a $R_k : R_p$ link. To be more formal, we introduce several following properties.

**Definition 1 (Functional Constraint).** A constraint $\varphi$ is a *functional constraint* if it verifies:

$$\varphi = ((T_1, D_1), (T_2, D_2)) \tag{3}$$

with $T_i$ the type of the module $i$ and $D_i$ the type of the dock of module $i$ participating in the link, module 1 being the module having the constraint and module 2 being the other module.

From the viewpoint of functional constraints, the topology $\Phi$ can be then defined as:

$$\Phi = ((c_1, (\varphi_{1,1}, \dots, \varphi_{1,c_1})), \dots, (c_n, (\varphi_{n,1}, \dots, \varphi_{n,c_n}))) \tag{4}$$

The variable (see description of the CSP approach in Sec. 4) whose value is true means that there is a link between the nodes, and if its value is false, then there is no such a link. The cost of the link is the cost of docking for two nodes if no dynamical obstacles were present in the environment. The constraints are the tricky part of this optimization step. First, we want an acyclic connected topology, which means a topology without loops. Such a topology is achieved when the two following properties hold:

*Property 1 (Connected Topology Characterization).* A topology of $n$ nodes is connected if and only if every subset of $i \leq \lfloor \frac{n}{2} \rfloor$ nodes has at least one link involving one of its node to one node of the complementary subset of nodes in the group.

*Property 2 (Acyclic Connected Graph Characterization).* A connected topology is acyclic if and only if it has exactly $n-1$ links between two nodes.

Such a topology is indeed a tree. Writing the constraints for the LP solver from those two properties is now easy and straightforward. There will be one constraint per subset of $i \leq \lfloor \frac{n}{2} \rfloor$, and one constraint to check that the topology is acyclic. Thus, the topology generated is ensured to be a single tree.

Finally, we include the functional constraints. To do that we need to split the whole optimization process in two phases: the Constraint Satisfaction and the Constraint Optimization Problems. CSP consists of the running LP solver giving an assignation of the constraints to the nodes. We suppose that this assignation is valid, *i.e.*, the type of node is compatible with the type constraints. Then we add a constraint ensuring that there are enough links satisfying functional constraints. For instance, if the node $i$ needs to dock to at least $N_A$ of type $A$ and $N_B$ of type $B$, then there will be two more constraints verifying that the number of links to the node $i$ of type $A$ (resp. $B$) is greater of equal to $N_A$ (resp. $N_B$).

**3. Local costs.** Robots have different on-board capabilities to measure distances between robots, their orientation, relative rotation and other parameters [21]. Moreover, as shown in Fig. 5(a), robots can measure distance not only to direct neighbors, but also to any visible object/robot. However, the further away the robots are, the less accurate is the measurement. Moreover, not visible robots, see Fig. 5(b), are not included into the local cost matrices.

The measured distance $S_{i,j}$ between $R_i$ and $R_j$ is one of the local costs for docking: the closer $R_i$ and $R_j$ are to each other, the "cheaper" is their docking $x^i_l \leftrightarrow x^j_k$. Other costs of $x^i_l \leftrightarrow x^j_k$ are the distance cost $\alpha S^{i-j}_{l-k}$, rotation cost $\beta S^i_{rot}$, cost of "being hidden" $S^i_{k-hid}$, where $\alpha, \beta$ are coefficients. The cost $S^i_{k-hid}$ is the price for the robot $R^i$ of not-knowing the situation around $R^i$. For example, when the costs of connection between $x^1_1 \leftrightarrow x^2_1$ are $S^{1-2}_{1-1} = \alpha S^{1-2}_{1-2} + \beta S^2_{rot} + S^2_{1-hid}$. More generally, local costs can include also any other factors, which determine a value of a particular connection. All local costs between all $x^i_l \leftrightarrow x^j_k$ are collected in the local costs matrix $\underline{\underline{S}}$.

Since morphogenesis is distributed and egocentric, the generations of $x^i_l \leftrightarrow x^j_k$ and $x^j_k \leftrightarrow x^i_l$ are asymmetric, i.e. from the viewpoint of the module $R_i$ a cost of

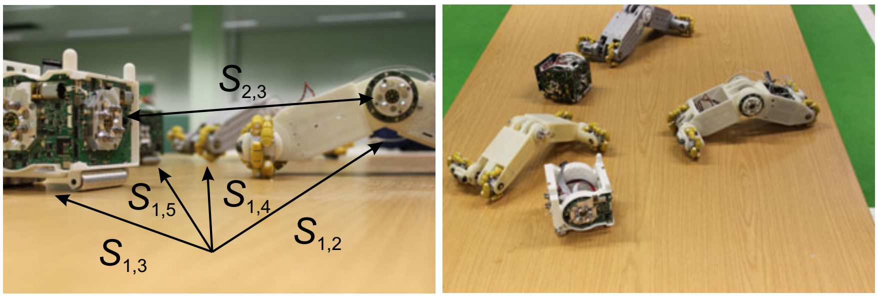


**Fig. 5** **(a)** Local costs, from the viewpoint of a robot; **(b)** Global view, where some robots are not visible.

connection between $R_i$ and $R_j$ may be different than the connection between $R_j$ and $R_i$ from the viewpoint of the module $R_j$. This leads to the effect, that the robot $R_i$ knows precisely only its local costs, all other costs can be estimated only roughly.

**4. Global costs.** Issue of global costs represents a crucial point. In general, it means how good the whole organism fits the environment and can be expressed as velocity of motion, energy consumption, weight and any other global factor for a specific environment. Normally, it requires multiple tests with different configurations and is very expensive or even impossible for on-line estimation in real situations. Therefore the proposal is to use a set of different a-priori-tested topologies, whose global costs for different environments are known. During experiments, depending on availability of tools and sensed environment, robots have to agree in which configuration they should collectively work.

## 4 Constraint-Based Optimization

The problem of constraint-based optimization can be formulated in two different ways, as described in [15] and [16]. The first approach, shown in Fig. 6(a), considers a classical assignment for each $ID_i \rightarrow R_i$, where $ID_i$ and a robot's ID and $R_i$ is a label of a robot in a topology. In the following table we display horizontally costs of all possible permutations between $ID_i \rightarrow R_j$ and vertically the connections $R_i \rightarrow R_j$ from the Fig. 6(a). The cost matrix (example of costs) takes the following form

| | 1 : 2 | 1 : 3 | 1 : 4 | 1 : 5 | 1 : 6 | 2 : 3 | 2 : 4 | 2 : 5 | 2 : 6 | 3 : 4 | 3 : 5 | 3 : 6 | 4 : 5 | 4 : 6 | 5 : 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 : 2 | 35 | 40 | 80 | 36 | 30 | 41 | 42 | 31 | 32 | 20 | 55 | 23 | 60 | 21 | 32 |
| 2 : 3 | 35 | 40 | 80 | 36 | 30 | 41 | 42 | 31 | 32 | 20 | 55 | 23 | 60 | 21 | 32 |
| 2 : 4 | 35 | 40 | 80 | 36 | 30 | 41 | 42 | 31 | 32 | 20 | 55 | 23 | 60 | 21 | 32 |
| 4 : 5 | 35 | 40 | 80 | 36 | 30 | 41 | 42 | 31 | 32 | 20 | 55 | 23 | 60 | 21 | 32 |
| 5 : 6 | 35 | 40 | 80 | 36 | 30 | 41 | 42 | 31 | 32 | 20 | 55 | 23 | 60 | 21 | 32 |

(5)

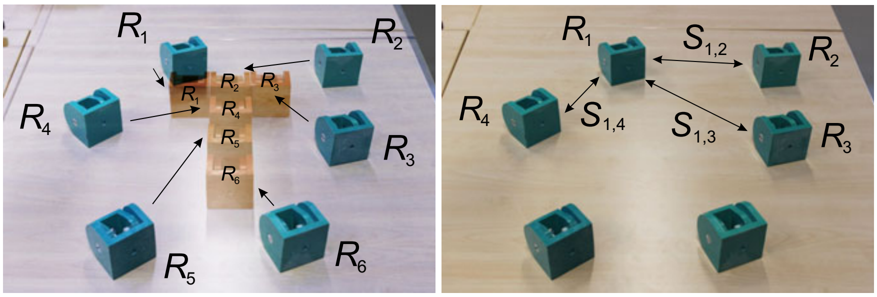


**Fig. 6** Example of the assembling problem. **(a)** Approach based on the assignment for each $ID_i \rightarrow R_i$; **(b)** Approach based in the linear program for the objective function $\Theta$ with $s_{i,j}$, shown are connections only to $R_1$.

where the assignment should satisfy

$$C_{ID_2 \rightarrow R_2} = \sum_{i}^{4} \sum_{j}^{4} S_{i,j} = 3. \tag{6}$$

The constraint (6) is a degree of connectivity for the main core element $R_2$. The problem, shown in Fig. 6(a), can be formulated as e.g. quadratic assignment problem (QAP), see e.g.[22] or as constraint-satisfaction problem (CSP) and constraint-optimization problem (COP), see e.g. [23]. Since QAP is a NP hard problem, we will solve this in the CSP+COP way. Solving the assignment problem, taking into account (6) for all elements, we receive the following assignment matrix

$$\begin{array}{c|ccccccccccccccc}
 & 1:2 & 1:3 & 1:4 & 1:5 & 1:6 & 2:3 & 2:4 & 2:5 & 2:6 & 3:4 & 3:5 & 3:6 & 4:5 & 4:6 & 5:6 \\
\hline
1:2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2:3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2:4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4:5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
5:6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array} \tag{7}$$

This approach has two essential drawbacks: need of robot's ID and large matrices (5),(7). Thus, this approach was improved in [16] and has the following form. First of all, we need to define the objective function $\Theta$, which is specified by $s_i$, see Fig. 6(b) (here only $R_1$ is shown). When $n$ robots are involved into some topology $\boldsymbol{\Phi}$, the variables $\underline{x}$ represent all possible bilateral connections between there robots. The vector of variables has $m$ components:

$$m = \frac{n!}{(n-2)!2!}. \tag{8}$$

There are several different $\Theta$, in the experiments we used

$$s_i = f_i(R_k : R_p) = D(R_k : R_p) + F(R_k : R_p), k, p = 1, ..., n; k \neq p, i = 1, ..., m \quad (9)$$

where $D(R_k : R_p)$ is a distance between neighbor $R_k$ and $R_p$, $F(R_k : R_p)$ satisfaction of functional constraints (0 when satisfied or $> maxD(R_k : R_p)$ not satisfied); all of them are estimated only to locally visible robots $R_k$ and $R_p$. The optimization is formulated as a linear program (LP), which optimize the linear objective function $\Theta = \underline{s}^T \underline{x}$, where $\underline{s}$ is the vector of costs and $\underline{x} = (x_1, x_2, ... x_m)^T$ is a vector of variables, which are bounded by 0 and 1. LP is constrained as follows

$$\underline{\underline{A}}\underline{x} = \underline{b}, \quad x_i \in \{0, ..., 1\}, \qquad (10)$$

where $\underline{\underline{A}}$ is a matrix and $\underline{b}$ is a vector of numerical coefficients, which form $m$ linear equations (in general case inequalities). In this form it is known as integer program. Finally, by solving (10), all variables $x_i$ take "0" or "1" so that to optimize $\underline{s}^T \underline{x}$. Now $\underline{\underline{A}}$ and $\underline{b}$ in (10) have to be defined; they reflect the connectivity constraints of the corresponding topology. For the topology shown in Figs. 4 and 6, this leads to the following linear problem

$$\underline{\underline{A}} = \begin{pmatrix} 1\;1\;1\;1\;1\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0 \\ 1\;0\;0\;0\;0\;1\;1\;1\;1\;0\;0\;0\;0\;0\;0 \\ 0\;1\;0\;0\;0\;1\;0\;0\;0\;1\;1\;1\;0\;0\;0 \\ 0\;0\;1\;0\;0\;0\;1\;0\;0\;1\;0\;0\;1\;1\;0 \\ 0\;0\;0\;1\;0\;0\;0\;1\;0\;0\;1\;0\;1\;0\;1 \\ 0\;0\;0\;0\;1\;0\;0\;0\;1\;0\;0\;1\;0\;1\;1 \\ 1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1\;1 \\ 0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0 \\ \ldots \\ 0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0\;0 \end{pmatrix}, \underline{b} = \begin{pmatrix} c_{max} \\ c_{max-1} \\ c_{max-2} \\ c_{min+2} \\ c_{min+1} \\ c_{min} \\ c_t \\ 0 \\ \ldots \\ 0 \end{pmatrix}, \quad or \quad \underline{b} = \begin{pmatrix} 3 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 5 \\ 0 \\ \ldots \\ 0 \end{pmatrix}. \quad (11)$$

As mentioned, all $c_i$ are disconnected from robots, i.e. we have to map the set of $c_i$ to all possible combinations between these robots

$$(c_{max}, c_{max-1}, ..., c_{min}) \rightarrow Permutation(R_1, R_2, ..., R_n). \qquad (12)$$

Since the number of permutations is equal to $n!$, computational power of the most of microprocessors allows computation for $n$ below 10 closely to real time. This is more than enough for a large diversity of cores, complex topologies are created through scalability. Since variable $x_i$ points to connections between robots, defined by (12), the vector $\underline{b}$ is equal to the set of $c_i$ in the order from $c_{max}$ to $c_{min}$ and the matrix $\underline{\underline{A}}$ creates corresponding placeholders. There are several comments to this approach.

1. All topologies have a list of associated constraints $\Upsilon$ such as a total $N$ of robots or requirement on heterogeneous modules. Moreover, each topology has a list of global costs: consumed energy, velocity of motion on a normal surface, geometry of concave and convex obstacle treatable with this topology or a possible geometry of docking elements.

2. Before start self-assembling, robots check whether $N$ of available robots match the set of possible topologies. For instance, when there are topologies requiring $\{15,17,22,25\}$ robots and there is only 20 available robots, they can self-assemble only into first two topologies.
3. Several topologies require tools or specialized robots. Robots should check availability of these specialized robots and correspondingly limit the set of possible topologies.
4. Self-assembling starts from the core element with the highest degree of connectivity. When such a core element is already built, robots consider the next element with the lower degree of connectivity and so on, until the whole structure is assembled.
5. When the selected topology is already partially assembled, and more free robots arrived to the assembling place, robots can decide instead of disassembling and new assembling to create a new core in the already assembled structure. Prerequisite is that the topology can be scaled up in the number of cores.

## 5 Grouping and Scaling Approaches

The CSP/COP-based optimization approach, described the previous section, is a distributed and decentralized process that each module in the assembling area performs. It cannot be described as a single task to achieve, the whole approach depends on interactions between modules and particular optimization processes running onboard. In this section we focus on two other tasks, which precede or follow the CSP/COP optimization: grouping and scalability approaches for different cases of $N > n$.

### *5.1 Grouping Approach*

Grouping is required for the case when the number of available robots $N$ is larger than the number of required modules $n$ in a topology. Thus, for $N > n$ we need to choose the modules that will participate in the aggregation. The first idea is to choose the $n$ nearest modules. This strategy works for topologies without any constraints on the type of modules, however can fail in other cases. Consequently, we need firstly to consider the type of modules in the neighborhood, and secondly, if a neighbor belongs to the already finished topology, it has to be removed from the list of available robots. Moreover, robots can have internal constraints that make some actions more difficult than others. For example, non-holonomic robots will not be able to move laterally, straight forward movement is cheaper than moving in any other direction. Hence, distances between $A$ and $B$, from the point of view of $A$ will be the complexity of $A$ to reach $B$.

To perform grouping, a simple function selecting iteratively each nearest neighbor can be considered. However, this can lead to a non-optimal solution. To perform the grouping optimization, an application of LP solver is more suited. Each neighbor has assigned a Boolean value by the LP solver: `true` if and only if the module is selected to be part of the group. The objective function is the sums of distances (as described in the previous sections) between the calling module and the selected modules of the neighborhood. The involved constraints are the number of modules selected (equal to the number of modules in the topology) and the type constraints (there is at least as much robot of a given type as required by the topology type constraints).

## 5.2 *Scaling Approach*

There are two general cases, where we need to consider a scalability of self-assembling. Firstly, we frequently encounter the situation, when the number of available robots $N$ is larger than the number of robots in the topology $n$. Here there are several strategies, which are considered below. Secondly, the topology with $n$ robots can join to another topology of $m$ robots; in this case the scalability represents the generating problem for $n+m$ topology. Such problem can be solved by morphing algorithms when basic hardware modules are able for this functionality. In our case, we solve the problem of $n+m$ topologies by a disassembling of one topology and an assembling of free robots into another one.

As mentioned, in the first case, there are several possibilities to scale $N$ in the relation to $n$:

**(1)** for $N = xn, x = 1,2,3,...$, the topology with $n$ robots can be replicated $x$ times. Each of these new topologies is an independent structure. This is the simplest form of scalability, which can be denoted as the behavioral scalability.

**(2)** $x$ topologies from the previous case can join into one common structure. This is typically segmented body construction, where $n$ robots within one segment are repeated $x$ times. This is the structural scalability.

**(3)** the robots from $N \ mod \ n > 0$ cannot create a new topology. These robots are still useful for the already existing topology, as e.g. energy reserve, so these robots can perturb the topology $\Phi$, this is the perturbational scalability.

**(4)** finally, $N \ mod \ n > 0$ robots are not aggregating with any other structures, they build a "reserve" for e.g. self-repairing.

The experiments performed in [16] and here indicated that a combination of the (1) and (4) strategies is the most useful approach for creating multiple artificial organisms. All results described in Sec. 6 utilize this strategy. For the cases (2) and (3) we need to recalculate kinematic constraints. For this we need to perform hybrid topological-kinematic techniques, which are described e.g. in [24].

## 6 Implementation and Results

We performed several series of experiments with real robots and in simulation. The implementation on the real platform was intended to test computational properties as well as to estimate the level of distortion in creating the objective function $\Theta$. Since currently there are not enough robots for testing scalability, several experiments are performed in simulation, which is done in AnyLogic. For implementation of

LP solver for CSP, we used lp_solve 5.5 routine (see lpsolve.sourceforge.net) of Mixed Integer Linear Programming solver (C++ version is used for real robots, Java version is used for simulation). Robots use Blackfin double core as the main CPU (in each module) with 64 Mb SDRAM on board. The robot arena was approximately 50x larger than the size of the robot. For measuring the time of experiments, we use the notion of "iteration of the autonomy cycle". This is an internal value of robots and allows estimating the running time more precisely to the steps of the CSP/COP approach (and not to the motion of the robot). Tests are performed with two topologies: "T"-like form shown in Fig. 4 and a snake. These topologies are used also for scalability tests. Additionally to experiments presented in [15], [16], we explored here different strategies for a grouping approach and its impact on a performance, and measured a performance of self-assembling at different scaling parameters.

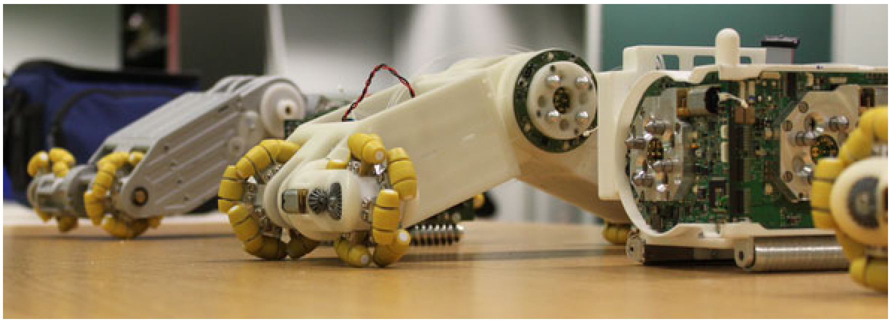


**Fig. 7** Ordering phase of the self-assembling approach. Five robots approached each other and moved into right spatial positions for creating a topology shown in Fig. 2(d). During this approach, a massive collision avoidance behavior can be observed.

**Grouping approach.** As mentioned in Sec. 5.1, grouping is a necessary step before self-assembling. It allows selecting the appropriate $n$ robots from the swarm of $N$ robots for building a topology. Implementing a grouping strategy, we encounter several difficulties. First of all, robots are not always visible to each other, see Fig. 7, i.e., estimation of the most closest robots to a group is in many situations not possible.

In the implemented strategy, we add a robot into a group when it is visible at least to one robot which already included into a group. Assigning to a group is performed by the "first visible, first served" principle. When a robot receives an invitation to join to a group, and can confirm or reject it. When no other invitation is accepted, a robot confirms its intention to join to the group and moves towards a visible inviting robot. When a robot approached the group closely enough, it starts CSP/COP procedure and waits a resolution of the assignment problem. After this, it moves to the right spatial position (the ordering phase).

To evaluate performance of this strategy, we used two other approaches: a candidate for joining a group was selected randomly based on WiFi connection (i.e. no position information), and the ideal case when only the closest robot was invited to join a group (by using global information). In Fig. 8 we plot the performance of the self-assembling with the neighbor-based grouping strategy and with a random grouping. In Figs. 8(a), 8(b) the performance is estimated as the sum of distances $s_i$ between all robots of a group, whereas Figs. 8(c), 8(d) demonstrate the performance as the sum of distances $s_i$ between all robots. Fig. 8(d) shows the ideal case with the closest neighbors. We can see that selection of robots for grouping has an impact on the assembling strategy. In many performed experiments it reduces the approaching time on 30%-50% and makes the ordering phase more early. In ideal case the ordering phase starts almost immediately after the start of experiments.

The second series of experiment was performed to investigate the scalability performance of the self-assembling strategy. The idea of this experiment originates

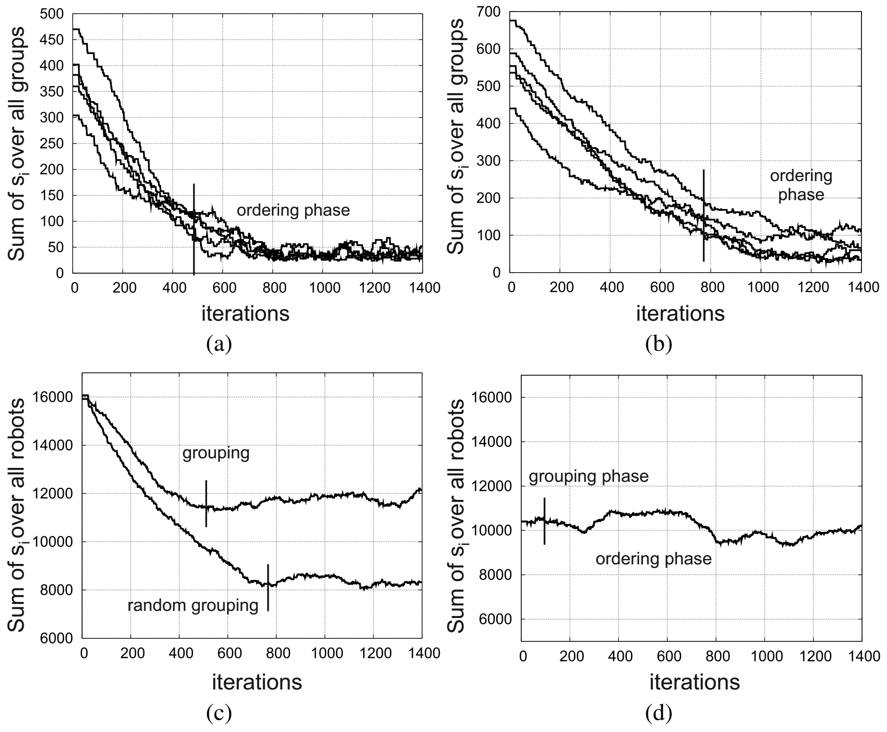
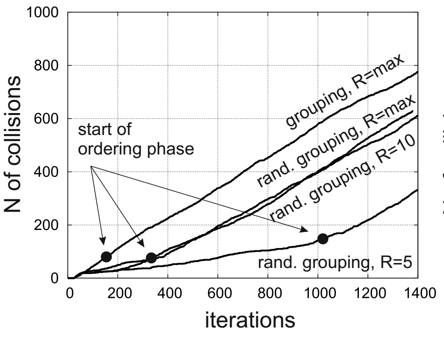
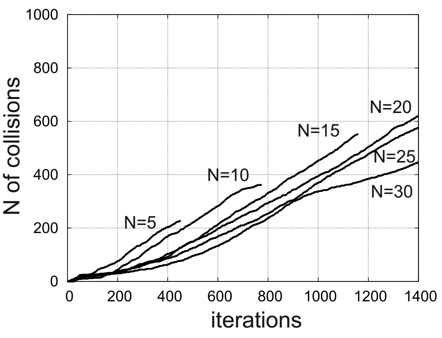


**Fig. 8** Different grouping strategies, $n = 5$, $N = 25$. **(a)** Grouping of locally visible robots to the each other, shown is the sum of all distances between the robots in each group; **(b)** Random grouping, all robots are equally distributed on the arena, shown is the sum of all distances between the robots in each group; **(c)** Comparison between both strategies, shown is the sum of $s_i$ over all robots; **(d)** Ideal grouping strategy, shown is the sum of $s_i$ over all robots.

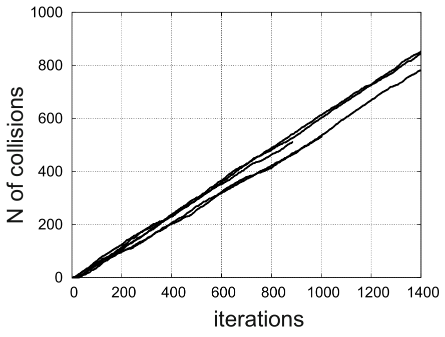
from [16], where we proposed to increase a connectivity of robots. It is achieved by movement of all robots to a specific point on the arena (it was a right, upper corner, which can be approached by using a compass information). When all robots are closely enough to each other, they can perform a faster grouping, creation of cost matrices and ordering. The encountered problem was that robots massively increased a number of collisions and this slowed down the performance. Moreover, the more robots are participated in the experiment the higher was the number of collision, and the slower was the assembling. In Fig. 9 we investigated the number of collisions for different grouping strategies and for different $N$ of robots. We encountered that the collision behavior has two well observable phases: the low-slope phase during the grouping and the high-slope phase during the ordering. The slope of the ordering phase is almost the same at all strategies, and is independent of the number of modules. This can be explained by the Fig. 7, where we can see that the ordering causes strongly local collision avoidance behavior and so is independent of



(a)



(b)



(c)

**Fig. 9** Scalability performance as a number of collisions over the running time of an experiment, $R$ is the sensing radius, $R = max.$: a robot can observe the whole arena, $R = 5, 10$: a robot can observe a range of 5 and 10 body lengths. **(a)** Scalability in relation to different grouping strategies and different visibility radii; **(b)** The strategy with random grouping and with the minimal sensing radius, shown are $N$ of collisions for different $N$ of robots; **(d)** The strategy with neighbor grouping, shown are $N$ of collisions for different $N$ of robots.

the total number of modules. However, different grouping strategies have different collision performance. As it follows from Fig. 9(c), the neighbor-based grouping has the best scalability properties (it is almost independent of $N$), whereas as the random-based strategies, especially with limited sensing radius, has a worse performance when increasing the number of robots. This behavior we observed also during earlier experiments. Thus, the best scalability strategy is to increase the level of locality not only for all robots during the aggregation, but also to increase the level of locality for the grouping phase.

## 7 Conclusion

This chapter is devoted to a self-assembling strategy, which utilizes generating and optimizing approaches, known in biological regulatory networks. We primarily focused here on the optimization controller, which narrows down the set of generated topologies to a particular description of connections between modules and performs local optimization defined by the objective function. This function takes into account connectivity and functional constraints, local and several global costs. Due to flexibility of costs and constraints, this approach is very useful for modules with different geometry and functionality, i.e. for heterogeneous reconfigurable robots.

For experimental part we mainly worked on the grouping strategies and scalability performance for different such strategies and different $N$. These experiments are additional to the already published results. We estimated that neighbor-based grouping, even not optimal in a global sense, can provide shorter aggregation time due to spatial optimization process. Non-optimality of spatial grouping is substantially limited by the capabilities to detect a neighbor robot. To improve the performance, we suggested increasing the level of locality by pre-aggregating all robots into a specific area of the robot arena. In the previous experiments, this strategy created multiple bottlenecks due to massive collision avoidance behavior among robots. In the improved approach, when a local grouping is selected, the pre-aggregated self-assembling is well scalable to different $N$, tested for $n = 5$ and $N$ between 5 and 30.

For further works, we would like to verify these tests for the topologies shown in Fig. 2 with three different types of robots. It is also of interest to investigate the influence of environmental conditions for very simple robots (like chemical molecules) and to estimate whether it is possible to transfer results from macroscopic to a microscopic self-assembly.

## References

1. Ariga, K., Hill, J.P., Lee, M.V., Vinu, A., Charvet, R., Acharya, S.: Challenges and breakthroughs in recent research on self-assembly. Science and Technology of Advanced Materials 9(1), 014109 (2008)
2. Whitesides, G.M., Kriebel, J.K., Love, J.C.: Molecular engineering of surfaces using self-assembled monolayers. Science Progress 88, 17–48(32) (2005)

3. Levi, P., Kernbach, S. (eds.): Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution. Springer, Heidelberg (2010)
4. Chiang, C.-J., Chirikjian, G.: Modular robot motion planning using similarity metrics. Auton. Robots 10(1), 91–106 (2001)
5. Miyashita, S., Hadorn, M., Hotz, P.E.: Water floating self-assembling agents. In: Nguyen, N.T., Grzech, A., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2007. LNCS (LNAI), vol. 4496, pp. 665–674. Springer, Heidelberg (2007)
6. Castano, A., Shen, W.-M., Will, P.: Conro: Towards deployable robots with inter-robots metamorphic capabilities. Journal of Autonomous Robots 8, 309–324 (2000)
7. Samori, P., Francke, V., Müllen, K., Rabe, J.P.: Self-assembly of a conjugated polymer: From molecular rods to a nanoribbon architecture with molecular dimensions. Chemistry - A European Journal 5, 2312–2317 (1999)
8. Kernbach, S.: Towards application of collective robotics in industrial environment. In: Rigatos, G. (ed.) Industrial Systems: Modelling, Automation and Adaptive Behaviour, pp. 18–49. IGI Global (2010)
9. Yu, C.-H., Haller, K., Ingber, D., Nagpal, R.: Morpho: A self-deformable modular robot inspired by cellular structure. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008, pp. 3571–3578 (2008)
10. Liu, W., Winfield, A.F.T.: Autonomous morphogenesis in self-assembling robots using IR-based sensing and local communications. In: Dorigo, M., Birattari, M., Di Caro, G.A., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L.M., Groß, R., Şahin, E., Sayama, H., Stützle, T. (eds.) ANTS 2010. LNCS, vol. 6234, pp. 107–118. Springer, Heidelberg (2010)
11. Huie, J.C.: Guided molecular self-assembly: a review of recent efforts. Smart Materials and Structures 12(2), 264 (2003), `http://stacks.iop.org/0964-1726/12/i=2/a=315`
12. Davidson, E.H.: The Regulatory Genome: Gene Regulatory Networks In Development And Evolution. Academic Press, London (2006)
13. Jacob, C., Burleigh, I.: Genetic programming inside a cell. In: Genetic Programming Theory and Practice III, vol. 9, pp. 191–206. Springer, Heidelberg (2005)
14. Banzhaf, W.: On evolutionary design, embodiment, and artificial regulatory networks. In: Iida, F., Pfeifer, R., Steels, L., Kuniyoshi, Y. (eds.) Embodied Artificial Intelligence. LNCS (LNAI), vol. 3139, pp. 284–292. Springer, Heidelberg (2004)
15. Kernbach, S.: From robot swarm to artificial organisms: Self-organization of structures, adaptivity and self-development. In: Levi, P., Kernbach, S. (eds.) Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution, pp. 5–25. Springer, Heidelberg (2010)
16. Kernbach, S.: Heterogeneous self-assembling based on constraint satisfaction problem. In: Martinoli, A., Mondada, F. (eds.) 10th International Symposium on Distributed Autonomous Robotics Systems. Springer Tracts in Advanced Robotics. Springer, Heidelberg (2011)
17. Salemi, B., Shen, W.-M.: Distributed behavior collaboration for self-reconfigurable robots. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA-2004), New Orleans, USA, pp. 4178–4183 (2004)
18. Lau, H.Y.K., Ko, A.W.Y., Lau, T.L.: The design of a representation and analysis method for modular self-reconfigurable robots. Robot. Comput.-Integr. Manuf. 24(2), 258–269 (2008)
19. Castano, A., Will, P.: Representing and discovering the configuration of CONRO robots. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA-2001), vol. 4, pp. 3503–3509. IEEE, Los Alamitos (2001)

20. Burkard, R., Dell'Amico, M., Martello, S. (eds.): Assignment Problems. Society for Industrial and Applied Mathematics (2009)
21. Kernbach, S., Meister, E., Scholz, O., Humza, R., Liedke, J., Ricotti, L., Jemai, J., Havlik, J., Liu, W.: Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution. In: Tyrrell, A. (ed.) Proc. of the IEEE Congress on Evolutionary Computation (IEEE CEC-2009). IEEE Press, Trondheim (2009)
22. Loiola, E., de Abreu, N.M., Boaventura-Netto, P., Hahn, P., Querido, T.: A survey for the quadratic assignment problem. European Journal of Operational Research 176(2), 657–690 (2007)
23. Kornienko, S., Kornienko, O., Priese, J.: Application of multi-agent planning to the assignment problem. Computers in Industry 54(3), 273–290 (2004)
24. Kernbach, S., Meister, E., Schlachter, F., Kernbach, O.: Adaptation and self-adaptation of developmental multi-robot systems. International Journal On Advances in Intelligent Systems 3(1,2), 121–140 (2010)