Self-Reconfiguration Using Directed Growth

K. Støy[1] and R. Nagpal[2]

[1] The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, Denmark, kaspers@mip.sdu.dk
[2] Division of Engineering and Applied Sciences, Harvard University, USA
rad@eecs.harvard.edu

Abstract

Self-reconfigurable robots are built from modules which are autonomously able to change the way they are connected, thus changing the overall shape of the robot. This process is difficult to control, because it involves the distributed coordination of large numbers of identical modules connected in time-varying ways.

We present an approach to self-reconfiguration where the desired configuration is grown from an initial seed module. Seeds produce growth by creating a recruitment gradient, using local communication, which spare modules climb to locate the seed. The growth is guided by a novel representation of the desired configuration, which is automatically generated from a 3D CAD model. This approach has two salient features: (1) the representation is concise, with a size proportional to the global shape rather than the number of modules and (2) there is a clean separation between the goal and the local rules used by the modules which are independent of the goal. We demonstrate three implementations of the local rules for recruitment, and show how one can trade-off the number of moves and messages, against time taken to reconfigure.

1 Introduction

Reconfigurable robots are built from modules and can be reconfigured by changing the way the modules are connected. If a robot is able autonomously to change the way the modules are connected, the robot is a self-reconfigurable robot. Self-reconfigurable robots are versatile because they can adapt their shape to fit the task. They are also robust because if a module fails it can be ejected and replaced by a spare module. Potential applications for such robots include search and rescue missions, planetary exploration, building and maintaining structures in space, and entertainment. Challenges exist both in the development of the hardware for the modules, as well as their control. This paper focuses on the challenge of controlling reconfiguration in a robot with many identical modules.

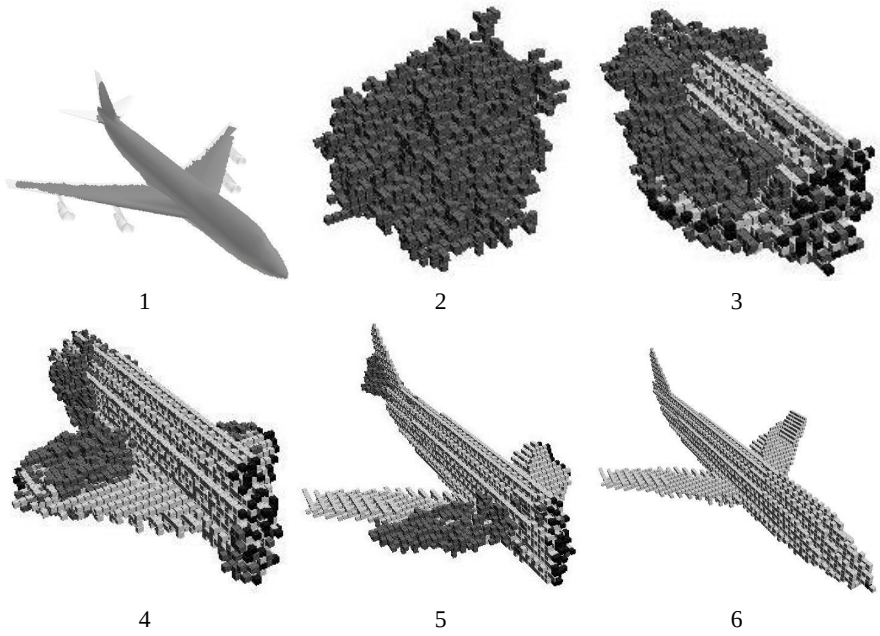


Fig. 1. 1: A brick based representation (grey) is generated based on a CAD model (light grey). **2-6**: This representation is used to control a reconfiguration process.

In this paper we present an approach for reconfiguration that consists of two steps. First, a 3D CAD model representing the desired configuration is transformed into a geometric representation based on overlapping bricks of different sizes. The representation is supplemented with a scaffold structure which removes local minima, hollow, or solid sub-configurations from the configuration. The second step is the actual reconfiguration process. The desired configuration is grown by choosing an arbitrary initial seed module. The seed module uses the bricks representation to determine if a neighbour module is needed at an unfilled neighbour position, and if so creates a recruitment gradient in the system. Spare modules climb this gradient to reach the unfilled position and may become seeds themselves if further construction is needed. Figure 1 shows an example of this self-configuration approach.

This approach has several salient features. The representation is automatically generated, is independent of initial configuration, and is concise with a size proportional to the global shape rather than the number of modules. The local rules for the module remain the same, irrespective of the goal shape. This separation, between goal and local rules, allows one to easily optimise or retarget the representation and explore alternate local rules. We demonstrate and compare three different implementations of recruitment that trade-off the extent of the gradient. The general method for recruitment using gradients was first introduced in [11], which used global gradients that cover the entire robot. Two alternate implementations introduced here are:

1) the range of the gradient is increased linearly until the unfilled neighbour position is filled and 2) the range is increased exponentially. We compare the number of moves, messages and time steps taken to complete reconfiguration. The simple global recruitment gradient is more time efficient than the two new approaches, but the linear strategy uses fewer moves.

2 Related Work

The self-reconfiguration problem is: given a start configuration, possibly a random one, how to move the modules in order to arrive at the desired final configuration. It is computational intractable to find the optimal solution (see [3] for a discussion). Therefore, self-reconfiguration planning and control are open to heuristic-based methods.

One type of approach is planning based, where a path is determined for each module in the original configuration. Chirikjian and others have proposed heuristic methods based on finding a suboptimal sequence of moves from initial to final configuration, which is then optimised by using local searches [3, 9]. Rus et al. simplify the planning problem by using an intermediate chain configuration, which is easy to configure into and out of [10]. Several papers have proposed hierarchical planners, where at the high level some of the hardware motion constraints are abstracted away to facilitate efficient planning. Based on the high-level plans, the lower level then produces the detailed sequence of actions [5, 14]. Another approach is to use meta-modules consisting of a small number of modules [5]. By planning at the meta-module level there are no or few motion constraints; on the other hand, meta-modules make the granularity of the robot larger. A related approach is to maintain a uniform scaffolding structure, facilitating planning [13]. Butler implemented the distributed Pacman algorithm on the Crystalline robot, which has very few motion constraints making the planning problem easier [2, 15]. The advantage of the planning approach is that it can accommodate motion constraints and minimise unnecessary moves; the disadvantage is that plans are often comparable in size to the number of modules and depend on knowing the initial configuration.

A different approach is to rely on common local rules as far as possible and then add randomness to deal with the problems that could not be solved using local rules. This was true in early work such as the work on Fracta [6] and also later work [17, 12]. The problem tended to be that even though the robot often ended up in the desired configuration, it did not always converge. This problem was also present in the work of Yim et al [16], however local communication was used to increase the probability of converging to the final shape. One solution to convergence, proposed by Bojinov et al. [1], is not to focus on a specific configuration. Instead, the idea is to build something with the right functionality. Using this approach it is acceptable if a few modules are stuck as long as the structure maintains its functionality. Alternatively, Jones et al. insist on a specific configuration, but achieve convergence by enforcing a specific sequence of construction [4]. In the work presented here, scaffolding is used to guarantee convergence by making sure that the configurations do not contain local minima, hollow, or solid sub-configurations.

Our system can be thought of as combining the two approaches: the global representation is a plan for constructing a shape from simpler shapes (bricks), while the local rules allow modules to recruit nearby modules to form bricks. This approach is similar to approaches for self-assembly used in Amorphous Computing, such as [7, 8]. There a global goal is specified as a construction which is then compiled into biologically-inspired local rules for agents, resulting in self-assembly that is scale-independent, robust and space efficient. The representation we use is inspired by the circle-network proposed by Kondacs for 2D self-assembly, however the agent model and local rules are completely different [8]. Instead we use local rules proposed by Støy [11] to control module movement.

3 Simulated Robot Model

In our simulation, we use modules which are more powerful than any existing hardware platforms but do fall within the definition of a Proteo module put forward by Yim *et al*. [16]. The modules are cubical and when connected they form a lattice structure. They have six hermaphrodite connectors and can connect to six other modules in the directions: east, west, north, south, up, and down. Modules directly connected to a module are referred to as neighbours. A module can sense whether there are modules in neighbouring lattice cells. In this implementation we do not control the actuator of the connection mechanism, but assume that neighbour modules are connected and disconnected appropriately. A module can only communicate with its neighbours. It is able to rotate around neighbours and to slide along the surface of a layer of modules. Finally, we assume that coordinate systems can transformed uniquely from one module to another. This is necessary to propagate the gradients and the coordinates used to guide the growth process.

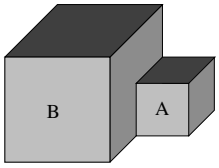
The simulator is programmed in Java3D. The simulation uses discrete time steps. In each time step all the modules are picked in a random sequence and are allowed: 1) to process the messages they have received since last time step, 2) to send messages to neighbours (but not wait for reply), and 3) to move if possible.

4 From CAD Model to Representation

It is difficult and time consuming to hand-code local rules which result in a desired configuration being assembled. Therefore, we need an automatic way of transforming a human-understandable description of a desired configuration into a representation we can use for control.

In our system, the desired configuration is specified using a connected three-dimensional volume in the VRML 1997 or Wavefront .obj file format, which are industry standards produced by most CAD programs. In earlier work we transformed the model into a cellular automaton, which represents relationships between neighbour modules in the desired configuration [11]. This representation has the disadvantage that it scales linearly in the number modules and has to be completely recompiled if the number of modules is changed.

Here we introduce a representation whose size instead scales with the complexity of the three-dimensional model and does not require recompilation if the number of modules changes. We approximate the input shape using a set of overlapping bricks of different sizes. This choice is fairly arbitrary and other basic geometric shapes, such as spheres or cones, could be used as well. The set of bricks is generated by starting at a user specified point inside the CAD model. The algorithm then fits as large a brick as possible which contains this point and does not intersect the CAD model. This is done recursively for all points just outside this brick, but inside the CAD model. This process continues until the volume has been filled with overlapping bricks. Figure 2 shows a simple example of a shape and its brick representation. The fewer bricks needed, the more concise the representation.



A: (0,0,1) → (3,1,2)
B: (0,0,0) → (2,2,2)

Fig. 2. This figure shows how a volume can be approximated with two overlapping bricks and how we represent this.

In order to control the resolution of the approximation a parameter r is supplied. The points and the corners of the bricks are then constrained to be positioned at coordinates equalling an integer times r. Table 1 shows the number of bricks needed to approximate a model of a Boing 747 at different resolutions. The size of representation scales with the complexity of the input shape and the resolution of the approximation. Furthermore, the bricks based representation can at run-time be scaled to match a specific number of modules.

Resolution	low	medium	high
Modules	32	4512	34493
Bricks	3	168	1152

Table 1. This table shows the number of modules and bricks needed to approximate a CAD model of a Boing 747 at three different resolutions.

5 From Representation to Self-Reconfiguration Algorithm

Starting from a random configuration the robot needs to reconfigure into the desired configuration as described by the representation. The self-reconfiguration algorithm consists of three components: a coordinate propagation mechanism, a mechanism to create gradients in the system, and a mechanism the modules use to move without disconnecting from the structure. We will look at these in turn.

5.1 Coordinate Propagation

All the modules are initially connected in a random configuration, have a copy of the representation of the desired configuration, a scale parameter, and start in the wandering state. An arbitrary module is given a random coordinate contained in the representation. The idea is to grow the configuration from this seed module. The seed can detect whether a module is needed in a neighbour position based on its coordinate and the representation. If this is the case, the seed attracts a wandering module to the unfilled position. When a module has reached an unfilled position and is given its coordinate it also may act as a seed if further construction is needed at this position. A module stops acting as a seed when all neighbour modules, specified by the representation and the seed's coordinate, are in place.

In order to simplify the reconfiguration problem a scaffold structure is enforced on the desired configuration; neighbour modules are only needed at positions which are contained in the brick representation and belong to the scaffold. The introduction of the scaffold sub-structure into the desired configuration simplifies the reconfiguration problem, because during reconfiguration it can be assumed that the configuration does not contain local minima, hollow, or solid sub-configurations. This simplification means that the system is convergent by design as described in [11].

5.2 Creating a Recruitment Gradient Using Local Communication

In this section we will describe how seed modules attract wandering modules by creating a gradient in the system. A seed module acts as a source and sends out an integer, representing the concentration of an artificial chemical, to all its neighbours. A non-source module calculates the concentration of the artificial chemical at its position by taking the maximum received value and subtracting one. This value is then propagated to all neighbours and so on. When the concentration reaches zero the gradient is not propagated further. This means that the source can decide the range of the gradient. We will explore different strategies for deciding this range in the experimentation section. Also, since messages take one time step to travel between neighbours, it can take many time steps for gradients to be propagated in the system.

If wandering modules have to rely on the basic integer based gradient to locate the source, they would have to move around randomly for a while to detect the direction of the gradient. Instead we introduce a vector gradient which makes direction

information available locally, thereby eliminating unnecessary moves. The basic gradient implementation is extended with a vector indicating the local direction of the gradient. This vector is updated by taking the vector from the neighbour with the highest concentration, adding a unit vector in the direction of this neighbour and renormalising the result. The paths to the unfilled positions always go through or on the surface of the structure. The structure does not contain local minima, because of the scaffold structure. Therefore, the paths to unfilled positions never contain local minima.

5.3 Staying Connected

Wandering modules climb the vector gradient to reach unfilled positions. Unfortunately, the wandering modules cannot move independently of each other, because they depend on each other for connection to the robot. The problem is then to keep the system connected while allowing wandering modules to move. In our solution finalised modules in the configuration emits a *connection gradient* and wandering modules only move if they do not perturb the gradient. Detailed rules for movement and proofs were presented in [11].

6 Experiments

In this section we investigate and compare three different strategies for implementing the recruitment gradient. In global recruitment a module needing a neighbour creates a gradient throughout the entire configuration. This, in effect, means that wandering modules always go toward the closest unfilled position even though other wandering modules may already be on their way there. This may result in three problems: 1) poor performance, because many modules are attracted to the same unfilled position and therefore many move in vain; 2) interference between modules because of overcrowding - a well known problem in literature, see for instance [16]; 3) the amount of communication needed to maintain global gradients increases with the size of the configuration and limits the system's scalability.

We address these problems by investigating two alternative recruitment strategies. In the first strategy a source linearly increases its concentration and therefore the range of the recruitment gradient. In the second strategy the source increase its concentration exponentially. The motivation behind these two recruitment strategies is that they recruit as locally as possible and therefore address the three problems mentioned above. We compare the strategies based on three criteria: time taken to reconfigure, number of module moves, and number of messages.

The task is to self-reconfigure from a random connected configuration of 618 modules to one which resembles a Boing 747 aeroplane. The representation of the configuration is built by the generator based on a CAD model. The representation is then downloaded into the modules of the simulation and the self-reconfiguration process is started.

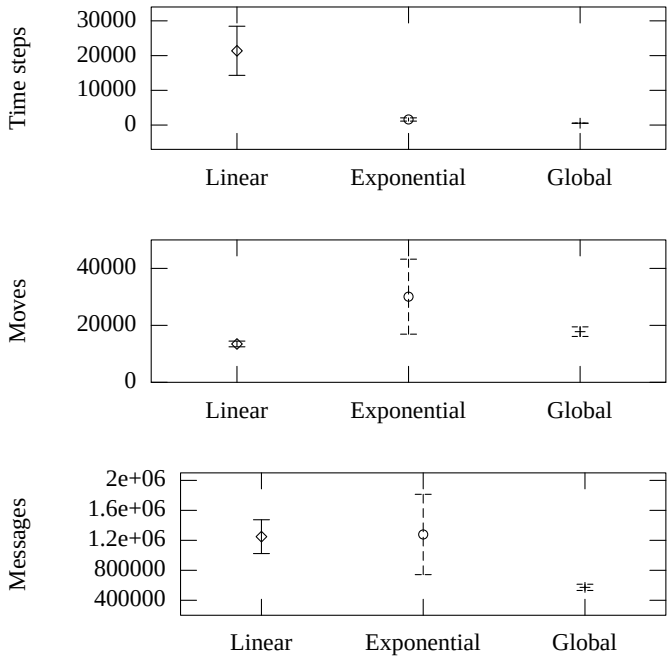


Fig. 3. This figure shows how the total number of time steps (top), moves (middle), and messages (bottom) depend on the recruitment strategy. Mean and standard deviation of 20 trials are shown.

In Figure 3, we can see that the global recruitment strategy outperforms the linear and exponential strategies in terms of the number of time steps and messages needed to reach the desired configuration. It can also be seen that the linear strategy outperforms the other two in terms of moves needed to complete a configuration. Therefore, the choice of strategy depends on which constraint is the most important.

Communication, in our system, is time consuming and as can also be seen in Figure 3 the global strategy uses significantly fewer messages than the other two strategies. This could lead one to conclude that the poor time efficiency of the linear strategy relies on the fact that it uses more messages. However, this is not the entire explanation. Figure 4 shows in more detail how the three strategies use their moves during construction. The global strategy recruits aggressively, making many modules move in parallel. Aggressive recruitment seems to improve time efficiency without increasing the number of moves significantly, because it takes advantage of a *heuristic*: if one module is needed more will be needed later. Where the global strategy is a parallel process, the linear essentially is a sequential process: instead of recruiting n modules in parallel, n modules are recruited one at a time. This makes the linear approach inefficient in terms of time even without factoring in the cost of communication.

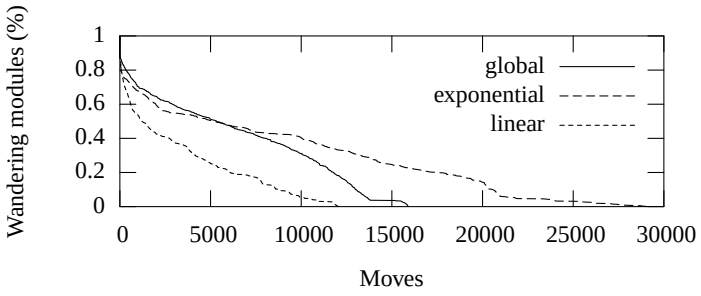


Fig. 4. This figure shows how moves for each strategy are distributed over a reconfiguration process.

The exponential strategy uses many more moves compared to the other two strategies. This is the case because sources tend to compete for the same modules. This causes the wandering modules to be trapped in the middle causing many unnecessary moves. Therefore, the global strategy is always preferable compared to the exponential.

7 Conclusion

We have explored an approach to the control of self-reconfiguration which consists of two steps. In the first step a generator takes as input a 3D CAD model of a desired configuration and outputs a set of overlapping bricks which represent this configuration. In the second step this representation is combined with a control algorithm to produce the final self-reconfiguration algorithm. This algorithm controls the self-reconfiguration process through a growth process: seed modules create recruitment gradients in the configuration that wandering modules climb to locate the seed.

In this paper we demonstrate that a representation based on geometric shapes is efficient in terms of space and is independent of the number of modules. We also show that a global recruitment strategy is more efficient in terms of time and messages while a linear strategy is more efficient in the number of moves. This highlights a key feature of our approach, which is that one can separately optimise (or even change) the global representation and the local rules for module movement. Overall, the proposed system represents a step toward systematic and efficient control of self-reconfigurable robots.

8 Acknowledgements

This research is in part funded by the EU contract IST-2001-33060, the Danish Technical Research Council contract 26-01-0088, and NSF grant EIA-0130391.

References

1. H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. In *Proc., IEEE Int. Conf. on Robotics & Automation (ICRA'00)*, volume 2, pages 1734 –1741, San Francisco, California, USA, 2000.
2. Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. In *Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, volume 2, pages 790–796, Maui, Hawaii, USA, 2001.
3. G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Robotics Systems*, 13:317–338, 1996.
4. C. Jones and Maja J. Matarić. From local to global behavior in intelligent self-assembly. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 721–726, Taipei, Taiwan, 2003.
5. K. Kotay and D. Rus. Algorithms for self-reconfiguring molecule motion planning. In *Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'00*, volume 3, pages 2184–2193, Maui, Hawaii, USA, 2000.
6. S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc., IEEE Int. Conf. on Robotics & Automation (ICRA'94)*, pages 441–448, San Diego, California, USA, 1994.
7. R. Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *Proc., 1st Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 418–425, Bologna, Italy, 2002.
8. R. Nagpal, A. Kondacs, and C. Chang. Programming methodology for biologically-inspired self-assembling systems. In *Proc., AAAI Spring Symposium on Computational Synthesis: From Basic Building Blocks to High Level Functionality*, 2003.
9. A. Pamecha, I. Ebert-Uphoff, and G.S. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.
10. D. Rus and M. Vona. Self-reconfiguration planning with compressible unit modules. In *Proc., IEEE Int. Conf. on Robotics and Automation (ICRA'99)*, volume 4, pages 2513–2530, Detroit, Michigan, USA, 1999.
11. K. Støy. Controlling self-reconfiguration using cellular automata and gradients. In *Proc., 8th int. conf. on intelligent autonomous systems (IAS-8) (to appear)*, Amsterdam, The Netherlands, 2004.
12. K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. A self-assembly and self-repair method for a distributed mechanical system. *IEEE Transactions on Robotics and Automation*, 15(6):1035–1045, Dec 1999.
13. C. Ünsal and P.K. Khosla. A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules. In *Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, volume 1, pages 598–605, Maui, Hawaii, USA, 2001.
14. C. Ünsal, H. Kiliccote, and P.K. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10(1):23–40, 2001.
15. S. Vassilvitskii, M. Yim, and J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proc., IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, volume 1, pages 117–122, Washington, DC, USA, 2002.
16. M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3d metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.
17. E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji. A distributed reconfiguration method for 3-d homogeneous structure. In *Proc., IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'98)*, volume 2, pages 852–859, Victoria, B.C., Canada, 1998.