The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

Making Shapes from Modules by Magnification

Byoungkwon An*
dran@csail.mit.edu

Daniela Rus*
rus@csail.mit.edu

***Abstract*— We present a distributed algorithm for creating a modular shape by magnification. The input to the algorithm is presented with a small scale version of the desired shape and a magnification factor m . The output of the system is the object that corresponds to the m-fold magnification of the input shape. We describe and analyze a distributed algorithm for this capability and present simulation results. Making shapes by magnification can be viewed as a programming interface for creating objects by programming matter.**

I. Introduction

Programmable Matter is achieved when a collection of small robotic modules that are physically connected have the ability to respond to the request of creating a goal shape autonomously. Many approaches have been proposed to creating shapes from modules. We have been developing a method for creating shapes by subtraction [5]. Starting with a collection of particles we call *a bag of smart pebbles*, we compute which pebbles need to connect together to create the desired object. Figure 2 shows example hardwares of smart pebbles. The resulting object can be pulled out of the bag. When the object is no longer needed the object is returned to the bag and its smart pebble components are recycled and made available for creating a different object.

In our previous work we presented distributed algorithms for creating a desired shape using subtraction as the fundamental operation [5]. In this paper we demonstrate the creation of objects by magnification. The intuition behind this idea is as follows. We present the system with a miniaturized version of the desired shape, for example a dollhouse sized chair. We then "dip" the object in the bag of smart particles along with a magnification parameter. The system outputs a copy of the desired object in the desired size–for example a life-size chair with the same geometry as that of the input object. The "smart" modules capable of computing communicating to neighbors, and making and breaking connections to neighbors. In this paper, the modules capable of computing are up-down counter, loop-break and evaluating a value is equal to 0 or not. The modules collectively determine the shape of the object and compute how to form connections among them to create a magnification of the shape by a given magnification factor. Upon completion, the desired object is extracted by reaching inside the smart bag. This approach to making shapes can be viewed as a new type of user interface for creating object. This user interface allows the "programming" of different shapes without the need for a computer to explicitly code the selection.

More specifically, we are given a set of modular robots with the ability of making and breaking connections. We are also given the description of a geometric shape and a magnification factor. In this paper we present a distributed algorithm that parses the given shape and creates the desired magnification of the shape. The algorithm is implemented in the subtraction approach to modular robots by self-disassembly introduced in our previous work [5], [6]. Creating robotic systems and smart objects by self-disassembly has one main advantage over existing approaches by self-assembly. Self-disassembling systems entail a simple actuation mechanism (disconnection) which is generally easier, faster, and more robust than actively seeking and making connections. Modular robots that can self-disassemble provide a simple and robust approach toward the goal of smart structures and digital clay. A collection of millions of modules, if each were small enough, could form a completely malleable building material that could solidify and then disassemble on command. As in existing selective laser sintering systems, (which fuse particulate matter to create rapid prototypes), a self-disassembling robotic system would only require the user to shake off the unused modules.

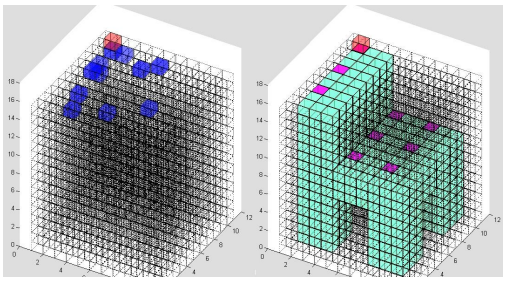

Fig. 1. Screenshot of the simulator. The simulator makes char when the magnification factor is 3. The module on red are the seed module. The modules on blue are running Algorithm 3. The modules on pink is marked itself as internal connecting modules and run Algorithm 4 . The modules on Green is marked itself as internal connecting modules.

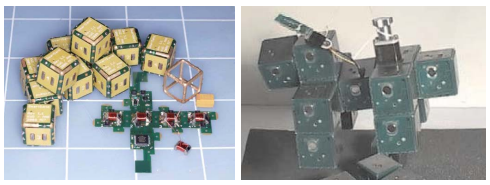

Fig. 2. Two examples of hardware modules that can use the magnification algorithms for making shapes [6], [5]. We are currently implementing the magnification algorithm on a 50 modules of robot pebbles[6]

The process of creating a shape by magnification proceeds as follows. An initial amorphously connected shape is created of the existing modules. Then, a scaled-down version of

*Distributed Robotics Lab, CSAIL, MIT

the desired shape is presented to the system. The system analyzes the desired goal shape and computes incrementally the desired magnification for the shape using parallel evaluation of a system of three rules per module. Using local communication, the group cooperates to distribute this information so that all modules know whether to remain as a part of the system or to extricate themselves. Finally, the unnecessary modules disconnect from the system and drop off to create the desired shape by magnifying the input shape.

A. Related Work

Our work builds on prior and ongoing research in modular and distributed robotics [1], [18], [4], [8], [11], [3], [10], [13], [16], [14], [2] and self-assembling systems [9], [15]. For a good review of this field see [17]. This prior work is concerned with how to build a modular system capable of aggregating different shapes autonomously. The work is focused on the basic module design, modular system architecture, and control and planning algorithms for achieving the desired shape creation or change. Most these systems are composed of identical modules that can connect to each other, communicate, have some actuation capabilities, and in general are able to cooperate to perform a task as a group. Like in these prior systems, we assume that the modules can connect and communicate with each other in order to perform a global task. The only actuation available to the system is in the form of local connections and disconnections. This is the subtraction model that supports making shape by disassembly and was introduced by our prior work [7], [5]. The work in this paper is different in that its focus is the creation of a desired magnification of a given shape in the subtraction model.

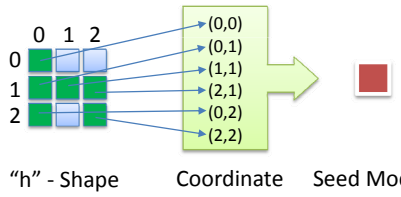

Fig. 3. Example of input "h" shape coordinates into the seed module.

Scaling has been proposed by [12] as a way of making the largest possible instantiation of a given shape geometry according to how many modules there are in a system. The algorithm is centralized and computes the largest object of a desired shape that can be constructed from a set of modules. When some modules are removed, the system adapts and compute smaller sized object. Our work is different from this prior work that uses magnification in several ways. We use subtraction as the basic model for creating shapes. We specify the desired magnification size as part of the goal. We develop and analyze a decentralized approach to this problem and examine the use of buffering as a way of optimizing communication in this system.

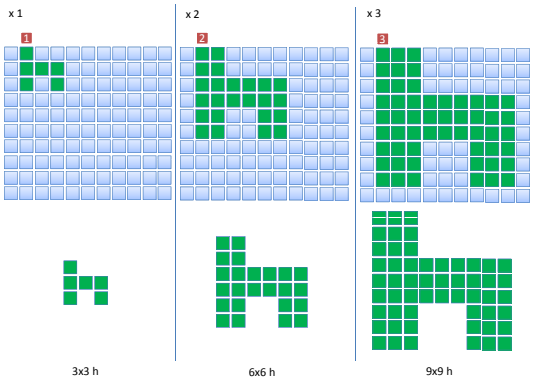

Fig. 4. An example of magnification "h" shape from the smart pebbles in 2D. Magnification factors are 1, 2, and 3 from left to right figures. Top figures are the smart pebbles in the bag. Bottom figures are the connected smart pebbles which is the magnified desired shape. The red module is the seed module. The modules on green is marked itself as internal connecting module. The module on blue is disconnecting module.

II. Problem Formulation: Distributed Magnification

We are given a collection of modules capable of shape-formation by subtraction such as the Smart Pebbles [6]. The modules in the system are not localized. We assume that they have the ability of communicating locally to neighbors and of programming their connections to attach to and detach from their neighbors.

In the future we envision providing the input to the system as a physical miniature shape the that system could envelop, model its geometry, and automatically extract the input information from it. For now we provide the model to the system as shown in Figures 3 and 4.

Shape formation by magnification and subtraction proceeds through three stages: shape parsing, shape distribution, and disassembly.

A. Shape Distribution

The first phase of shape distribution takes as input the geometry of the sample and the magnification factor (Figure 3). This phase expects a representation of the shape in the form of a list of 3d coordinates expressed in a local coordinate system. The goal is to ensure that each module in the final magnified configuration is informed that it is part of it. We will accomplish this by a sequence of messages flowing outward to mark some key modules on the boundary of the structure. These modules correspond to the mapping of the input shape modules to the goal shape. A second sequence of messages originate at the location of these special modules and locate and mark all the internal modules in the goal shape. These messages are called inclusion messages.

This list can be provided directly to the system as input. Alternatively, this representation can also be extracted from the sample object when the object consists of a set of connected modules. Parsing the geometry of the sample object for a list of local coordinates for each module is an instance of localization and proceeds as follows. The

Algorithm 1 Shape Distribution: Push the Shape Representation Message

```
input1: loc (= list of coordinates)
input2: scale (= magnification Factor)
for f = 1 to The number of loc do
    message.x = loc[f].x × scale
    message.y = loc[f].y × scale
    message.z = loc[f].z × scale
    message.m = scale
    loop
        Send message to the neighbor
    end loop
end for
```

Algorithm 2 Shape Distribution: Processing the Shape Message without the Buffer

```
loop
    wait to receive a message
    if message ≠ (0,0,0) then
        Do Algorithm 3: Shape Distribution
    else
        Do Algorithm 4: Shape Magnification
    end if
end loop
```

Seed Module determines the relational location $(0,0,0)$ (Figures 4 and 5 .) The Seed Module sends a message to the structure as a starting point (Algorithm 1 and Figure 5.) Each message includes the coordinate of the final destination and the magnification factor $((x,y,z),m)$. Figure 2 shows an example of the message. When the message is $((0,6,3),3)$, the final destination is the relational location $(0,6,3)$ and its magnification factor is 3. When a message is received, it is transmitted to the next module as explained in Algorithm 2. The messages arrive their final destination without and any global or local knowledge about the system because the modules receiving the message repeat the Algorithm 2.

Given an encoding of the model shape in the form of a list of 3D coordinates and a magnification factor, the final shape is computed and transmitted to the connected modules forming the "bag of smart sand" as follows (see Algorithm 1.) For each coordinate tuple in the list, the magnification factor is applied to each coordinate. As shownin

Algorithm 3 Shape Distribution: Distributing the Shape Message

```
if message.x ≠ 0 then
    message.x −− ; send message to right until success
else if message.y ≠ 0 then
    message.y −− ; send message to front until success
else if message.z ≠ 0 then
    message.z −− ; send message to bottom until success
end if
```

Algorithm 4 Shape Distribution: Magnifying the Shape Message

```
if message.x = 0, message.y = 0 and message.z = 0 then
    Set the connected module state
    message.m −−
    if message.m ≠ 0 then
        // message is ((0,0,0),m)
        send three messages to right, front, and bottom, respectively, until success
        message.y ++
        // message is ((0,1,0),m)
        send message to right until success
        message.z ++
        // message is ((0,1,1),m)
        send message to right until success
        message.y −−
        // message is ((0,0,1),m)
        send message to right until success
        send message to bottom until success
    end if
end if
```

Algorithm 5 Shape Distribution: Distributing the Shape Message (Negative Coordinate Extension)

```
if message.x < 0 then
    message.x ++; send message to left until success
else if message.y < 0 then
    message.y ++; send message to back until success
else if message.z < 0 then
    message.z ++; send message to top until success
else if message.x ≠ 0 then
    message.x −−; send message to right until success
else if message.y ≠ 0 then
    message.y −−; send message to front until success
else if message.z ≠ 0 then
    message.z −−; send message to bottom until success
end if
```

Figure 5 the coordinate $(0,2,1)$ becomes $(0,4,2)$ when the magnification factor is 2 and the coordinate $(0,2,1)$ becomes $(0,6,3)$ when the magnification factor is 3. A new message is created out of the new coordinates and the magnification factor. Algorithm 1 generates and sends such a message for each module in the original structure. The message sent from the seed module is no difference from the other messages between modules. So, the modules the receiving the messages from seed modules process Algorithms 2 without any difference from the messages from the "normal" modules.

Algorithm 2, 3, and 4 shows how a pushed shape message is processed. Algorithm 5 that can replace Algorithm 3 is for negative coordinate extension. The goal is to push this message through the structure until it reaches the module it labels (Algorithm 3 .) Suppose module (i,j,k) has received

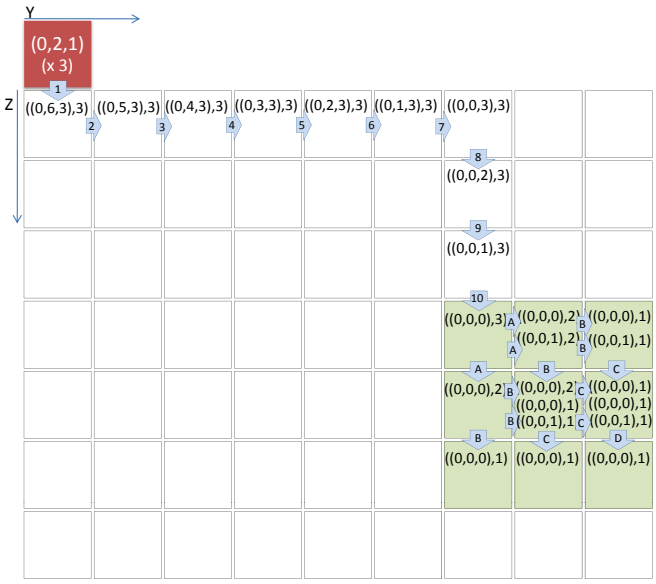

Fig. 5. An example of the message transmission. The red module is the Seed Module. The green modules are connected modules (Algorithm 4.) The tuples $((i,j,k),m)$ in the modules are the received messages. The numbered arrows show the message transmitting. The numbers indicate the sequence of transmission. Arrow 1 shows the first transmission (Algorithm 1.) Arrows 2 - 10 show the case when the transmitting message is not ((0,0,0),m) (Algorithm 3.) Arrows A - D show the case when the transmitting message is ((0,0,0),m) (Algorithm 4 .)

the message $((0,6,3),3)$. The module will forward the message to one of its neighbors that is closer to the destination than itself. If the module is pushed along the x axis its first coordinate is decreased. If the module is pushed along the y axis its y coordinate is decreased. If the module is pushed along the z axis its z coordinate is decreased. For the example in Figure 5 the message $((0,6,3),3)$ traveling along the y axis will be received in sequence, for example $((0,5,3),3)$, next $((0,4,3),3)$, $((0,3,3),3)$, $((0,2,3),3)$, $((0,1,3),3)$, and finally $((0,0,3),3)$. At this point the message will change direction and start traveling along the z axis till it becomes $((0,0,0),3)$.

At this point the given module in the original structure has found its corresponding module in the magnified structure (Algorithm 4.) When all the modules in the input shape have found their match in the goal shape, the goal shape skeleton is computed but the internal modules are not marked. The final stage of shape distribution identifies and marks the internal modules. Intuitively, we would like for each already marked to be in the final shape to send final shape inclusion messages to the internal modules that are closest to it. If the module has coordinates (i,j,k) and the magnification factor is m, this module must send inclusion messages to all the modules in the $m \times m \times m$ block originating at (i,j,k). The following procedure accomplishes this task. Upon receipt of the message $((0,0,0),m)$ the receiving module generates messages it sends to its neighbors in the x, y, and z directions. Each message hop decreases the magnification factor. For example, the module receiving the message $((0,0,0),3)$ sends the message $((0,0,0),2)$ in the x,y and z direction, the messages $((0,1,0),2)$ and $((0,1,1),2)$ in x direction, the message $((0,0,1),2)$ in x and z direction (Algorihtm 4.) The message generation procedure is repeated by all nodes asynchronously and in parallel until the magnification factor becomes 1. Upon receipt of the message $((0,0,0),1)$, the receiving node stops forwarding the inclusion message as Figure 5 and Algorithm 4.

At this stage all the modules in the goal magnified shape are marked. The next phase of the algorithm requires that all modules that did not receive inclusion messages disconnect. The remaining connected structure is the desired magnified object.

B. Correctness Analysis

We now demonstrate that the magnified shape is computed correctly.

Theorem 2.1: Given a goal shape and a magnification factor m, Algorithms 1, 2, 3 and 4 will mark all the modules of the goal shape and only the modules of the goal shape.

Proof: We observe that the magnification happens in two main phases for each message. In the first phase each module in the original structure is mapped to a location in the magnified structure. This establishes the scaffold of the magnified object. Next, assign the local coordinates $(0,0,0)$ to each module in the scaffold. By Algorithm 4, each module will send inclusion messages to all the modules in the $m \times m \times m$ block originating at $(0,0,0)$. The union of these blocks is precisely the desired magnified object.

■

Algorithm 6 Shape Distribution: Processing the Shape message with the Buffers

```
1:  < Processing Buffers>
2:  loop
3:     Wait until receiving a message
4:     if bufferQueue are full then
5:        reply false
6:     else
7:        receive a message to bufferQueue
8:        reply true
9:     end if
10: end loop
11: < Buffering >
12: loop
13:    wait until bufferQueue has messages
14:    message = bufferQueue.pop()
15:    Do Algorithm 2
16: end loop
```

C. Optimization by Buffering

We observe that if the modules do not have a way to simultaneously store and process messages, the performance of the system is affected by the order in which messages arrive at each module. When multiple messages arrive at the same time it is beneficial to use buffering to parallelize message processing. To use buffering, Algorithm 6 replaces

Algorithm 2. Algorithm 6 shows an extension of the system that uses buffering. This algorithm does not change the flow and correctness of the Magnification algorithm.

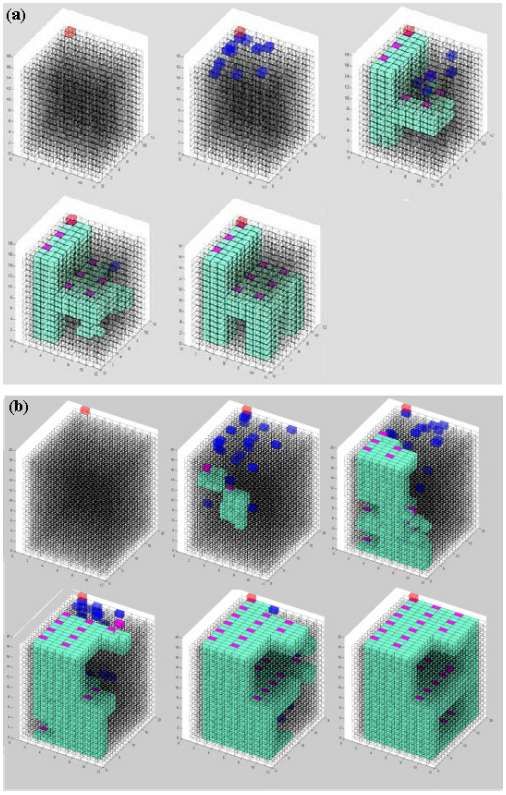

Fig. 6. Two examples of screenshots from the simulator. The first example shows the creation of a chair. The second example shows the creation of a shelf. In each image the red module is the seed module. The modules in blue are running Algorithm 3 . The modules in pink are the final destinations of the messages transmitted from the Seed Module. The modules in pink and green are internal modules and run Algorithm 4.

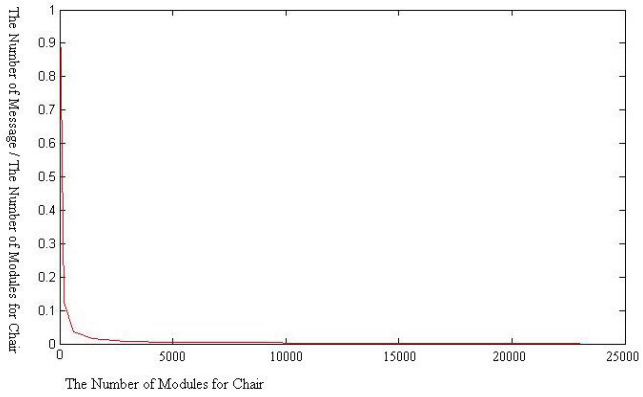

Fig. 7. The Number of Modules for The Shape vs. The Number of Messages / The Number of Modules for Chair Shape

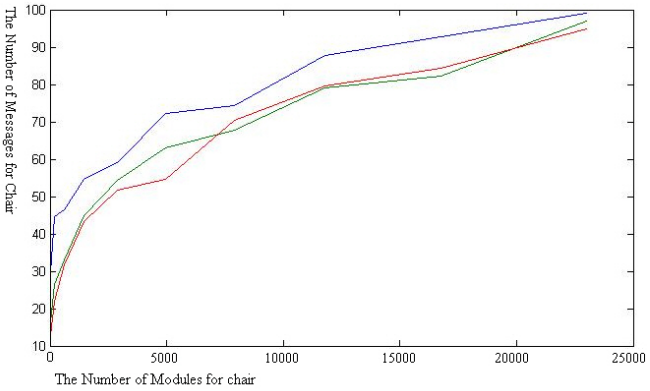

Fig. 8. The Number of Messages for The Shape vs. The Number of Modules for The Shape. Blue is No buffer, green is 1 buffer, and red is 2 buffer

III. Implementation in Simulation

We have implemented Algorithms 1, 2, 4, 5, and 6 in simulation in MATLAB. We have used this system to experiment with several different object geometries and magnification factors. We have done simulation experiments on several complex objects (Figure 6). We repeated each object magnification simulation three times and iterated for all magnification scales between 1 to 10. We have done the simulations without buffering, and with 1 and 2 buffers.

To evaluate building a shape with this system, we use three metrics: (1) the number of messages, (2) the size of each module's buffer, and (3) the computational power of each module's processor. The number of messages is a function of the size of required memory, the input, and time. The buffer size is dependent on the size of circuits in the module and to the time required to build the desired object. The algorithm has a minimalist flavor. It consists of three main computation functions: communication of a message containing four numbers, counting, and comparing to zero.

Number of messages. For a scale 1 size chair composed of 23 module, the number of the messages is 23. Our algorithm also uses 23 messages for building the scale 10 size chair which needs 23000 modules. We conducted simulation experiments for creating chairs with all scaling factors between 1 and 10. Table I shows the Message Efficiency for building the chair with scaling factors 1 through 10. We define Message Efficiency as the ratio of the number of messages to the number of modules while building the desired shape. Figure 7 plots the message efficiency.

Buffering. To see the effects of buffering, consider using Algorithm 3, 4 and 5 to make a chair. When (i, j, k) is (0, 0, 0), the processor will be locked until all of the magnification messages are sent, because all of the messages are unique and independent of the other messages. Table II and Figure 8 show how buffering affects running time. The simulations were evaluated using the ratio between the number of buffers and building time, when chairs with varying scaling factors are built. Each scaling factor was simulated 10 times and the reported numbers are averages. For small scaling factors, the buffers make a difference by speeding the computation. For large scaling factors, the number of buffers does not affect the running time of the algorithm.

TABLE I

RESULT OF SIMULATION - THE NUMBER OF MESSAGE / THE NUMBER OF MODULES FOR THE SHAPE

scale 1 (23)	scale 2 (184)	scale 3 (621)	scale 4 (1472)	scale 5 (2875)
1.0000	0.1250	0.0370	0.0156	0.0080
scale 6 (4968)	scale 7 (7889)	scale 8 (11776)	scale 9 (16767)	scale 10 (23000)
0.0046	0.0029	0.0020	0.0014	0.0010

*(): The number of modules using for the chair

TABLE II

RESULT OF SIMULATION - TIME OF BUILDING THE SHAPE

	scale 1 (23)*	scale 2 (184)	scale 3 (621)	scale 4 (1472)	scale 5 (2875)
0 buffer	30.0000	44.6667	46.3333	54.6667	59.0000
1 buffer	16.3333	26.3333	33.0000	45.0000	54.3333
2 buffers	12.6667	22.0000	31.6667	43.3333	51.6667
	scale 6 (4968)	scale 7 (7889)	scale 8 (11776)	scale 9 (16767)	scale 10 (23000)
0 buffer	72.3333	74.3333	87.6667	92.6667	99.0000
1 buffer	63.0000	67.6667	79.0000	82.3333	97.0000
2 buffers	54.6667	70.3333	79.6667	84.3333	95.0000

*(): The number of modules using for the chair

IV. CONCLUSIONS AND FUTURE WORKS

We have explored the use of magnification as a way of programming shapes in modular systems. In this paper, we have presented algorithms for creating complex shapes by magnification in a modular robot system. We describe the algorithms, analyze their correctness and discuss efficiency in the context of simulation data. The algorithm is decentralized and minimal in the number of messages required to create a shape. Two important goals for designing the magnification algorithm has been to minimize information flow in systems with limited resources, and to minimize the storage and communication required to build a large shape that consists of many modules. The magnification algorithm requires only three function as discussed in implementation section. This approach to communication minimization can be applied to more general complex networks, for example as shown in Figure 9.

We are currently working on two extensions of the magnification algorithm. The magnification algorithm can be applied to the formation of complex shapes with variable scale features–for example some parts of the object can be magnified more than others. The magnification algorithm can be modified to operate in reverse direction to compress shape for a modular system which has limited power or memory space. We are also working on designing a new type of computer, that runs the magnification algorithm with distributed architecture.

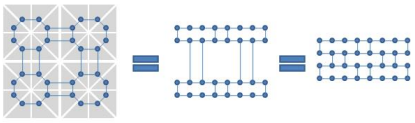

Fig. 9. Simplified from triangle tile to graph.

V. ACKNOWLEDGMENTS

Support for this research has been provided in part by the DARPA Programmable Matter program. We are grateful for this support. We thank Sangbae Kim, Kyle Gilpin, and Rob Wood for insightful discussions and feedback on this research.

REFERENCES

[1] Byoungkwon An. Em-cube: cube-shaped, self-reconfigurable robots sliding onstructure surfaces. In *IEEE International Conference on Robotics and Automation(ICRA)*, pages 3149–3155. IEEE, 2008.

[2] Preethi Srinivas Bhat, James Kuffner, Seth Goldstein, and Siddhartha Srinivasa. Hierarchical motion planning for self-reconfigurable modular robots. In *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[3] A. Castano and P. Will. Mechanical design of a module for reconfigurable robots. In *Proc. of International Conference on Intelligent Robots and Systems (IROS)*, pages 2203–2209, 2000.

[4] Andres Castano, Alberto Behar, and Peter Will. The conro modules for reconfigurable robots. *IEEE Transactions on Mechatronics*, 7(4):403–409, December 2002.

[5] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *Int. J. Rob. Res.*, 27(3-4):345–372, 2008.

[6] Kyle Gilpin and Daniela Rus. Self-disassembling robots pebbles: New results and ideas for self-assembly of 3d structures. In *Proceedings of ICRA Workshop on Modular Robots: The State of the Art*. IEEE, 2010.

[7] Kyle Gilpin, Iuliu Vasilescu, and Daniela Rus. Miche: Modular shape formation by self-dissasembly. In *IEEE International Conference on Robotics and Automation(ICRA)*, pages 2241–2247. IEEE, 2007.

[8] Akiya Kamimura, Haruhisa Kurokawa, Eiichi Yoshida, Satoshi Murata, Kohji Tomita, and Shigeru Kokaji. Automatic locomotion design and experiments for a modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 10(3):314–325, June 2005.

[9] R. Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *Proc. of International Conference on Autonomous Agents and Multiagent Systems*, 2002.

[10] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Trans. on Robotics and Automation*, 13(4):531–45, 1997.

[11] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *International Journal of Robotics Research*, 22(9):699–715, 2003.

[12] Kasper Stoy and Radhika Nagpal. Self-repair and scale-independent self-reconfiguration. In *IEEE Conference on Robotic Systems (IROS)*. IEEE, 2004.

[13] C. Ünsal and P. Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 1742–7, 2000.

[14] Paul White, Victor Zykov, Josh Bongard, and Hod Lipson. Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[15] G. Whitesides and B. Grzybowski. Self-assembly at all scales. *Science*, 295:2418–21, March 2002.

[16] M. Yim. Digital clay. WebSite.

[17] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):43–52, 2007.

[18] Mark Yim, Ying Zhang, Kimon Roufas, David Duff, and Craig Eldershaw. Connecting and disconnecting for self–reconfiguration with polybot. In *IEEE/ASME Transaction on Mechatronics, special issue on Information Technology in Mechatronics*, 2003.