Passive Reconfigurable Robot

Chintan A. Dalal[*]

Mechanical Engineering and Applied Mechanics
University of Pennsylvania
Philadelphia, PA 19104

February 17, 2006

Abstract

Current Implementation of self-reconfigurable robotics rearrange through a planned, deterministic reconfiguration path. This process of self-assembling is accomplished through deliberate active motions. Here we study a form of self-reconfigurable robotics based on passive, stochastic self-organization. This macro-scale, programmable modules exploit Brownian motion in the environment to form parallel local structures. This then aggregates to self-assemble in to a required global geometry. We implement the graph grammar technique. We discuss and extend the theory of graph grammars for generic self-organizing processes. We also simulate and validate this theory, giving an example of forming a shape 'P', from solid-state cellular units. Finally discussing challenges in the initial approach to addressing the passive reconfiguration problem.

1 Introduction

Self reconfigurable robot forms required global geometrical structure, from variable number of homogenous, macro-scale independent modules. They can rearrange to new morphologies autonomously which acts together to accomplish diverse tasks. There have been lots of successful works which reconfigures through a planned deterministic path. The individual units have onboard power and ability to locomote, or are deterministically moved to the appropriate place by other units. These systems have many applications and advantages: economic mass production of units, graceful degradation of function when damaged also called Self-repair, and the ability to transition into topologies suitable to the task at hand.

The deliberate active reconfiguration places severe power and mechanical actuation problems on the design of each module. As we try to engineer processes at smaller scales power storage is difficult and mechanical locomotion and actuation are limited. Also the computation, for uniquely addressing, manipulating and coordinating each one, grows exponentially as we increase the number units which partake in self-assembling.There are many Biological and Physical systems which built in bulk spontaneously out of large number of simple components, without centralized control or sequential execution. Bimolecular self-assembly occurs when two single strands of Deoxyribonucleic Acid (DNA) assembles into the double helix, as shown in Figure 1. This extremely long DNA molecule is actually made up of long strings of chemical building blocks called 'nucleotides'. The human genome is formed from a sequence of roughly three billion of these nucleotides which have self-assembled itself in to strands of DNA. Other

[*]This work is advised by Dr. Mark Yim

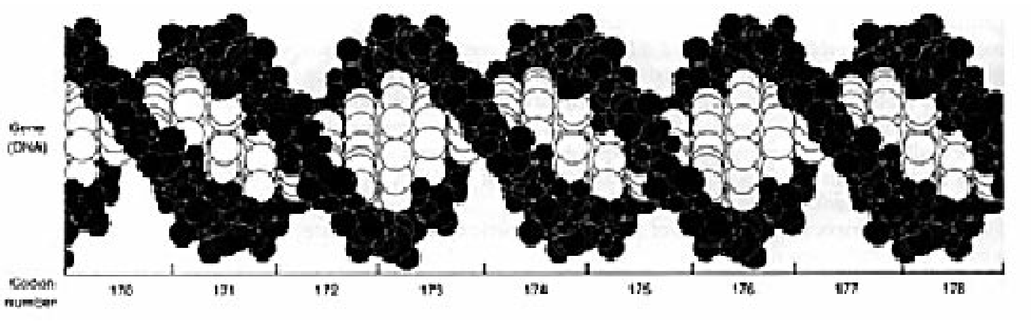


Figure 1: Passive self-reconfiguration of DNA into the double helix

such examples are protein molecules like Hemoglobin, Ribosome and hybrid structured viruses. These cells and organisms take advantage of the fact that its both easier and needs less genetic information to assemble very large complexes, from smaller easier to make subunits, the process is called 'bottom up approach'. One can see such passive stochastic formation in physical system too, like, drop coins in bucket of water and one sees that all the coins self-assembles in to a geometry, due to surface tension.

Here I plan to study a robotic system that very much mimics the self-assemble processes of natural systems. Vast number of small modular units moves around in zero-gravity space or in agitated fluid environment. These units are unpowered and become active only once they bond to the main structure. They have no locomotion ability. The potential location of a unit is determined by active bonding sites and statistical mechanics of Brownian motion. Unit interfaces are identical and their function differentiates depending on their final context. The robots are initialized from arbitrary starting position. When they come in close proximity to each other, they attach or repel based on whether their conformation are complementary. Once attach they share power and information, and then move together in the embedded Brownian environment. Eventually when all the required units attach together they form the desired structure.

To accomplish passive reconfiguration, we try to understand and build a theory and simulation which addresses question like: In what pattern should the modules parallely self-assemble such that the final shape is formed with minimum amount of entropy, how much of concavity can the subassemblies survive before it goes into deadlock, how much of global information should be provided to each modules, can the pattern policies be decided 'Online', how many number of modules are needed to form the shape, how many various shapes can be formed and finally can we built a general mathematical framework which addresses all of the above issues.

In Section 3, we explain theories for the robotics systems environment, its shape and a computational framework. In Section 4, we provide logical steps for making handmade graph grammar, Exemplifying and validating by forming shape 'P' and then giving some simulation difficulties and results. Finally Section 5 discusses some of the challenges and opportunities towards passive reconfiguration.

2 Previous and Related Work

There have been number of design and prototypes developed in this field of self reconfigurable robotics. Yim [3] developed modular robots that can be manually reconfigured to form different structures that have various means of locomotion, and has since developed a number of systems such as PolyBot [4]. Fukuda and Kawauchi [12] developed cellular robots called CEBOT. Rus and Vona [13] developed Crystalline modules that can form robotic systems

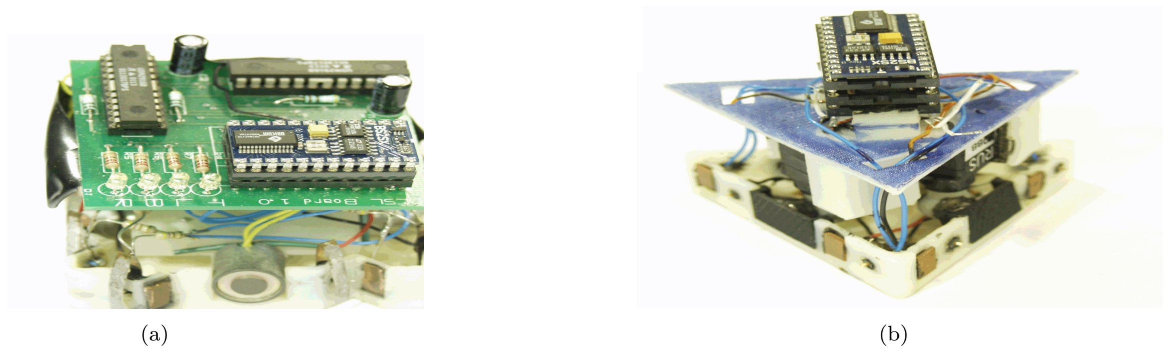


Figure 2: **Two prototype units made by White [1]**: (a) square, usign electromagnets and (b) triangle, using swiveling permanent magnets

by collapsing and expanding the body of each module. Rus *et al* also experimented with prototypes of molecules [14][15],that have a pair of two degree of freedom atoms and can form 3D shapes. Støy et al [16] developed role based control strategy for robot CONRO [17].

There are also various theoretical and algorithmic concepts, which tries to solve certain aspects of the questions addressed in previous Section. Kondacs [9] presented a programming language for assembly of arbitrary two dimensional shapes, that instructs the agents to grow the shape via replication and location based control mechanism. As this micro-scale units are ligthly packed and are based on axioms *Origami* it cannot be used for our purpose. White [1] has proposed this concept of passive reconfigurable robot and there on built a primitive, planar working prototype. They have built square and triangular cellular units, moving in nearly frictionless motion, bouncing off each other and the table boundaries. It assembles in 'L' shapes and a line. Basic components on the units (shown in Figure 2) are Basic Stamp II microchip to write mathematical program in it, H-bridges to control the electromagnets and magnets on the sides of the units. Usually there are two types of magnets used for the purpose, electromagnet and the permanent magnets. The permanent magnet have larger attraction basin do not require elaborate alignment pattern. However design would involve moving parts, this makes it less compatible with the overall process of passive self-reconfiguration at micro-scale level.

The basic idea that we employ is that of conformation switching [18]. This symbolic process was first proposed by Saitou [19], who considered assembly of strings in one dimension. Self-assembly as a graph process has been described by Klavins [1], [5], [6], [7]. They have described various examples of formulism and then focused on synthesis of acyclic graphs. The main advantage of this approach is that the mathematics used to model self-organization is equivalent to the code used to program the parts. However, the algorithm can make at the most ternary grammars. Also the rule set from the algorithm is not optimal, in the sense of both total time of shape formation and its size. This may be, due to the fact, that they make binary rules with the *MakeTree* procedure and then use 'scaffolding' technique to close cycles. The entire procedure is explained in detail in [1]. These rules are not commutative, hence the subassemblies cannot be built concurrently. Furthermore, only a small set of geometrical shapes can be formed by the formulism Klavins provide. We have tried to study and extend on Klavins approach. If further work is done around this idea, it seems that, it can solve most of the questions of passive self reconfiguration.

Several groups [20], [21] have explored self-assembly using *passive tiles* floating in liquid. Tile systems [22] can perform arbitrary computation as the graph grammar approach. Another approach is to use geometrical constraints on part-part interaction to model. This type of assembly can be seen in biology, like, protein forming spherical shells called capsides [24] and in T4 bacteriophage [25]. There have also been lots of work on self-assembly in the field of chemistry and nanostructures, by Whitsides [23], [26], [22]. However, they are role based and depend on environment for a particular shape formation. hence difficult to apply on macro scale passively reconfiguring robot.

3 Theory

In this Section we provide some of the theories which we would be using in simulation in Section 4. We would also be implementing this in future, to build a working prototype.

3.1 Brownian Motion

Our robots are situated in a fluid that behaves like Brownian motion. This motion is very irregular, composed of translation and rotation, and the trajectories have no tangent(shown in Figure 3). Every particle move independently. The motion is more active when lesser the viscousity and higher the temperature. One can imagine minute particles suspended in a liquid moving chaotically under the action of collision with surrounding molecules. The mean Kinetic energy of any molecule is equal to the mean kinetic energy of a particle suspended in the ambient space.

$$E = 3KT/2 \qquad where\, K\ =\ Boltzman\, Constant\, and\, T\ =\ Temperature\, of\ environment. \tag{1}$$

The brownian motion of small particle in a stationary fluid in x-direction is governed by the following Langevin equation

$$\frac{\partial u}{\partial t}\ +\ \beta\mu\ =\ n(t) \tag{2}$$

where u velocity of particle, $n(t)$ = white noise excitation due to impact of fluid molecules on the particle and $\beta = 3\pi\mu d/C_c m = 1/\tau$. Brownian motion in force field is given by following equations.

$$\ddot{x}\ +\ \beta\dot{x}\ -\ \frac{F(x)}{m}\ =\ n(t) \tag{3}$$

where $F(x) = -\frac{\partial V(x)}{\partial x}$ and $V(x) = mg(x - x_o)$= gravitational force field.

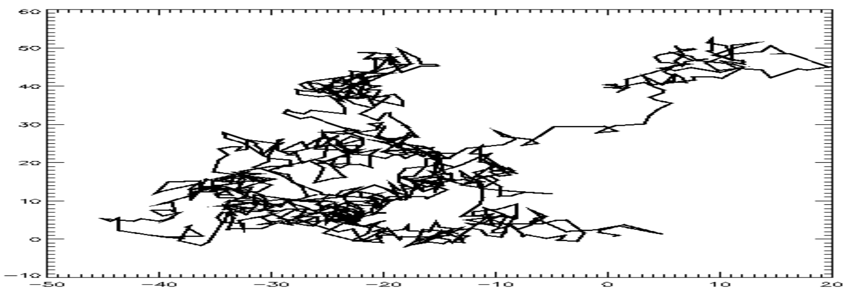


Figure 3: Trajectory of a particle in Brownian Motion

In our simulation, see the Algorithm 1, we consider only translational two dimensional Brownian motion. Due to the incapabilities of MATLAB's *Graphical User Interface* toolbox and complexity in 'bookkeeping' algorithm ,we were not able to simulate random rotational motion in 3D space.In future, we would try to solve this problem by using two different softwares. Mathematical language in MATLAB and graphics in OpenGL. And integrate them to simulate a better approximation of Brownian motion in 3D.

```
Initialize all robots $x$ and $y$ postions inside the boundaries of the space;
foreach robot do
    if Robot is Independent or Leader then
        if Robot inside the boundaries of space or not in close proximity with other robot then
            translate the robots position;
        else
            Inverse its velocity and translate in opposite direction;
        end
    else if robot is a follower then
        assign position relative to the leader;
    end
end
forall robots do
    Draw using MATLAB graphics functions explained in Section 3.2
end
```

Algorithm 1: Algorithm explaining the generation of our embedded space

In the Algorithm 1 the random function generator was used, in MATLAB, for white noise generation. The *variance* selected is equivalent to Fluid agitation needed in the prototype. The *Proximity* for the robot to interact depends on the magnetic feild produced by them. We model here with $\|X_i - X_j\| \leq 2cm$, where X is the robots position.

3.2 Rhombic Dodecahedron(RD)

Here we provide theory on the the shape of the robotic module we would be implementing in future, both in simulation and in prototype. This has also been explored by Yim [10].

The two important property of this shape is that it is 1) *space-filling* and 2) its a *Zonohedron.*

Definition 3.2.1 *A space filling polyhedron is a polyhedron which can be used to generate* tesselation *of space.*

Definition 3.2.2 *Teselation is a collection of disjoint open set, the closure of which covers space.*

The advantage to this is that the approximation to arbitrary actual global shape is better.

Definition 3.2.3 *Zonohedron-consider a star with n-line segments through one point in space such that no three lines are coplanar, then there exists a polyhedron, known as zonohedron whose faces consists of* $n(n-1)$ *rhombi and whose edges are parallel to the n-given lines in sets of* $2(n-1)$.

This says that every face is centrally symmetric and its advantage is that the combinatorics of faces is equivalent to line arrangements in place.

RD is formed by placing six cubes on the faces of seventh. Then joining the centers of six cubes with the vertices of central seventh cube, shown in Figure 4.

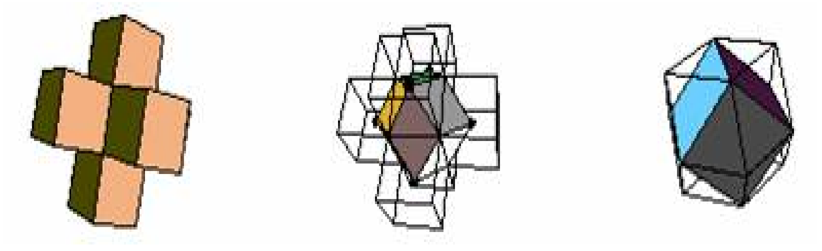


Figure 4: Forming A Rhombic Dodecahedron

14 vertices of the RD are joined by 12 rhombi of the dimension as shown in Figure 5. Here $\alpha = 2/\cot\sqrt{2}$ and $\beta = 2/\tan\sqrt{2}$.

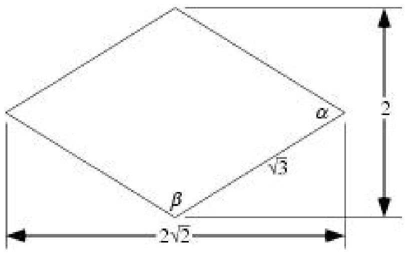


Figure 5: A face of Rhombic Dodecahedron

Some more advantages of RD are:

1. Maneuvering around each other is better
2. Simplify motions of the modules : example for shifting the cubes in structure one need to slide over the edge, but in RD rotating over vertex by 120° each time one can move its position.
3. Batch fabrication at both face level and edge level

However one of the limitation of this shape, is that, it can reconfigure in to only polyhedron geometry.

We made a GUI library of this shape with the 'fill3' command in MATLAB. This command takes as argument the vertices of the faces and makes patches of this faces with respect to origin. Hence showing a 3D RD shape. Due to such a sloppy process the virtual memory of the computer was completely exhausted with only five modules. Hence we made our simulation in 2D. So that we can first check the mathematical language involved in passive reconfiguration. And then in future make a 3D model with RD shapes, using OpenGL softwares. The function we made in MATLAB is called by *rhombicdodecahedron*, this takes as its arguments the x, y, z postion and the color of the shape to be drawn.

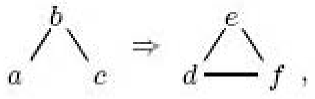
3.3 Graph Grammers

Graph grammars are generalization of the standard 'linear' grammars used in automata theory and linguistics. The basic idea, is that, when robots come in close proximity to each other they communicate with each other by sharing their internal states. If the internal states and local topology matches L, then the parts rewrite their states and local topology with R. These set of rules of the form $L \Rightarrow R$, can be designed so that the parts assemble into any desired structure.

A *simple labeled graph* over an alphabet $\Sigma = \{a, b, c, \ldots\}$ is a triple $G = (V, E, l)$ where V is a set of vertices, E is a set of unordered pairs or edges from V, and $l : V \to \Sigma$ is a labeling function.Vertex x corresponds to the edge of the robot and an edge is denoted by $\{x, y\} \in E$. V_G, E_G and l_G is the vertex set, edge set and the labeling function of the graph G.

Definition 3.3.1 *A rule is a pair of graphs $r = (L, R)$ where $V_L = V_R$. The graphs L and R are called the* left hand side *and the* right hand side *of r respectively.*

if (L, R) is a rule, we usually denote it by $L \Rightarrow R$. We also represent rules graphically as in where the relative



location of the vertics represent their identities. In the above rule, for example, $V = \{1, 2, 3\}$ and vertex 1 is labeled by $l_L(1) = a$ in he left hand size and by $l_R(1) = d$ in the right hand side.

A *monomorphism* between graphs G_1 and G_2 is an injective function $h : V_{G_1} \to V_{G_2}$ that preserves edges: $xy \in E_{G_1} \Leftrightarrow h(x)h(y) \in E_{G_2}$. It is $label-preserving$ if $l_{G_1} = l_{G_2} \circ h$

Definition 3.3.2 *A rule $r = (L, R)$ is applicable to a graph G if there exists a label-preserving monomorphism $h : V_L \to V_G$. An action on a graph G is a pair (r, h) such that r is applicable to G with witness h.*

Definition 3.3.3 *Given a graph $G = (V, E, l)$ and an action (r, h) on G with $r = (L, R)$, the application of (r, h) to G yields a new graph $G' = (V, E', l')$ defined by*

$$\begin{aligned} E' &= (E - \{h(x)h(y) \mid xy \in L\}) \cup \{h(x)h(y) \mid xy \in R\} \\ l' &= \begin{cases} l(x) & \textit{if } x \notin h(V_L) \\ l_R \circ h^{-1}(x) & \textit{otherwise} \end{cases} \end{aligned}$$

We write $G \overset{r,h}{\to} G'$ to denote that G' was obtained from G by the application of (r, h).The vertex set V of the the graph G in the above definition remains the same upon the application of a rule. Only the presence or absence of edges and the labels on the vertices change.

Definition 3.3.4 *A system is a pair (G_0, Φ) where G_0 is the initial graph of the system and Φ is a set of rules (called the ruleset or grammar).*

This set of rules is the algorithm that the robots follows. We assume to initialize with $a \in \Sigma$ i.e. same initial internal state.

Definition 3.3.5 *A trajectory of a system (G_0, Φ) is a (finite or infinite) sequence*

$$G_0 \xrightarrow{r_1,h_1} G_1 \xrightarrow{r_2,h_2} G_2 \xrightarrow{r_3,h_3} \ldots$$

If the sequence is finite, then we require that there is no rule in Φ applicable to the terminal graph. Let the set of trajectories of a system be $\mathcal{T}(G_0, \Phi)$. This set is non deterministic, as several rules in Φ may be simulataneously applicable via several different witnesses.

We denote a trajectory by $\sigma \in \mathcal{T}(G_0, \Phi)$ and use the notation σ_j to mean the j^{th} graph in the trajectory.

Definition 3.3.6 *G is connected if any pair of vertices can be connected by a sequence of edges in G. The connectivity relation partitions any graph into connected components. A graph G is reachable by the system (G_0, Φ) if there exists a trajectory $\sigma \in \mathcal{T}(G_0, \Phi)$ with $G = \sigma_k$ for some k. The set of all such reachable graphs is denoted $\mathcal{R}(G_0, \Phi)$.*

Definition 3.3.7 *A connected graph H is a reachable component of a system (G_0, Φ) if there exists a graph $G \in \mathcal{R}(G_0, \Phi)$ such that H is a component of G. The set of all such reachable components is denoted $\mathcal{C}(G_0, \Phi)$.*

Definition 3.3.8 *A component $H \in \mathcal{C}(G_0, \Phi)$ is stable if, whenever H is a component of $G_k \in \mathcal{R}(G_0, \Phi)$ via a monomorphism f, then H is also a component via f of every graph in $\mathcal{R}(G_k, \Phi)$. The stable components are denoted $\mathcal{S}(G_0, \Phi) \subseteq \mathcal{C}(G_0, \Phi)$. Reachable components that are not stable are called transient.*

The stable components are those that no applicable rule can change.

A graph grammer is essentially a parallel algorihtm. Two actions can be executed in parallel if they operate on disjoint parts of whatever is the current graph. This also called as concurrent semantics, is more formally defined below.

Definition 3.3.9 *Let $A = \{a_1, \dots, a_k\}$ with $a_i = ((L_i, R_i), h_i)$ be a set of actions each applicable to a graph G. The set A is called commutative if for all i and j*

$$h_i(L_i) \cap h_j(L_j) = \phi$$

If A is commutative with respect to G_0 and $G_0 \xrightarrow{r_1, h_1} \dots \xrightarrow{r_k, h_k} G_k$, then we write $G_0 \xrightarrow{A} G_k$.

When A is commutative, we are justifed in writing $G \xrightarrow{A} G_0$ since the graph G_0 is independent of the order in which the actions in A are applied. Using this notion, we form the set of concurrent trajectories of a system.

Definition 3.3.10 *A concurrent trajectory of a system $(G_0 \Phi)$ is a (finite or infnite) sequence*

$$G_0 \xrightarrow{A_1} G_1 \xrightarrow{A_2} G_2 \xrightarrow{A_3} \dots$$

where A_{i+1} is any (not necessarily maximal) commutative set of actions applicable to G_i for each i. If the sequence is finite, then we require that there is no rule in Φ applicable to the terminal graph.

Concurrency allows us to understand the nature of number of steps required to assemble a structure with a given graph grammer. The following defintion tells us how to calculate the worst case and the best case assembly time.

Definition 3.3.11 *For $H \in \mathcal{C}(G_0, \Phi)$ a reachable component, the assembly time $\tau_H(\bar{\sigma})$ of H in the trajectory $\bar{\sigma} \in \mathcal{T}(G_0, \Phi)$ is the smallest integer i such that H is a component of $\bar{\sigma}_i$(or ∞ if there is no such integer). The best case assembly time of H is the minimum of the set*

$$\{\tau_H(\bar{\sigma}) | \bar{\sigma} \in \mathcal{T}(G_0, \Phi)\}$$

The worst case assembly time of H is the maximum of the set

$$\{\tau_H(\bar{\sigma}) | \bar{\sigma} \in \mathcal{T}(G_0, \Phi) \, and \, H \, occurs \, in \, \bar{\sigma}\}$$

If the given reachable component H does not occur in the trajectory, then we declare its worst case assembly time to be ∞.

In Denition 3.3.5 we have difined *interleaved semantics*. In this model actions do not occur simultaneously. However a set of successive actions operate on disjoint parts of a garph to be interleaved in any order.One can compress any interleaved trajectory by applying several consecutive actions simultaneously. Conversely, any concurrent trajectory can be transformed into an interleaved trajectory by choosing some ordering of the elements within each commutative step A_i.

Some more theories and properties of graph grammer is provided in Appendix A. A simple example explaining formation of an acyclic and a cyclic configuration, as shown in Figure 6,is defined by an constructive rule set as below,

$$\Phi_1 = \begin{cases} a \quad a \;\Rightarrow\; b - b, & (r_1) \\ a \quad b \;\Rightarrow\; b - c, & (r_2) \\ b \quad b \;\Rightarrow\; c - c. & (r_3) \end{cases}$$

the first rule in Φ corresponds to $L = (\{1,2\}, \Phi, \lambda x.a)$ and $L = (\{1,2\}, \{12\}, \lambda x.b)$. At first, the only applicable rule is r_1 since initially no vertices are labeled by either b or c. After r_1 is applied, both r_1 and r_2 become applicable. The rule r_3 eventually becomes applicable as soon as two unconnected vertices labeled b arise. From the Figure below we see that the grammer produces the path P_k starting and ending with vertices labeled b and with internal vertices labeled c. The grammer also produces cycles C_k with all the vertices labeled by c.

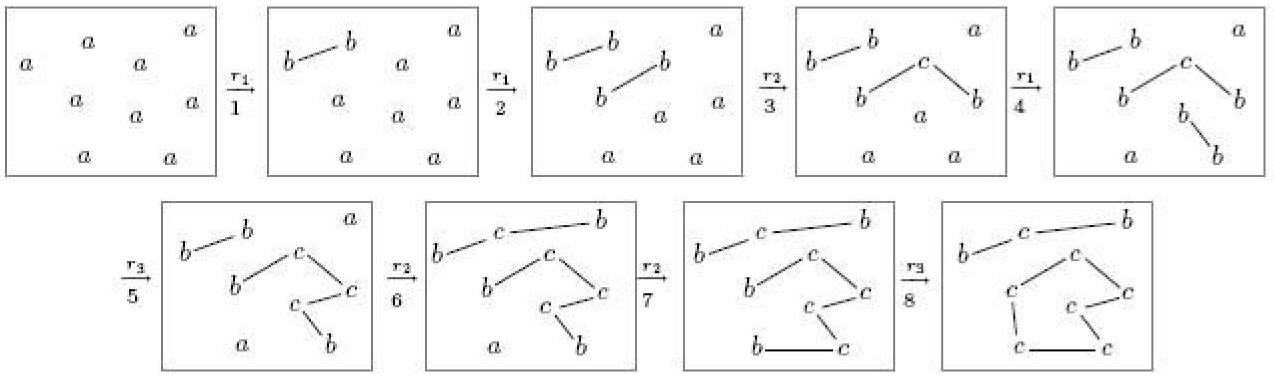


Figure 6: A trajectory of system defined by rule set Φ_1.Two configuration is formed an acyclic and a cyclic path.

4 Implementation

We have implemented the passive reconfigurable system with an example of forming the shape of '**P**'. This particular shape was chosen because of its property of having acyclic as well as cyclic sub-configuration. It also has a cavity in its configuration making it even more interesting and difficult to generate. Coincidently it's also an emblem for University of Pennsylvania.

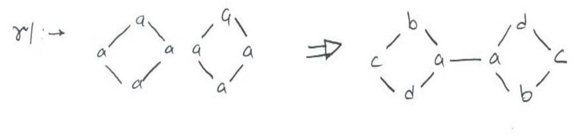
In the process to generate a *handmade* [1] grammar, we seek to, understand its relevance and usefulness in applying to generic passive reconfigurable systems. From all the possible σ of the stable components $\mathcal{S}(G_0, \Phi)$, reaching our final shape 'P' ($\mathcal{R}(G_0, \Phi)$), we decided the rule set Φ as shown below. This is because it gave the best time of formation in simulation and logic. Our grammar behaves like a left-greedy commutative semantics. This behavior is proved, in [1], to have best formation time. Definition in Appendix A.

Our robots are cellular units, with four sides, having the capability to attract or repel according to the grammar. Two robots are required to cooperate via one communication event. We have built a Quaternary rule size, with four edges representing four vertices of a rule. Hence embedding a purely topological object C_4 (a cycle of four vertices) into $\mathbb{R}^2$. We always require that the embedding is counterclockwise.

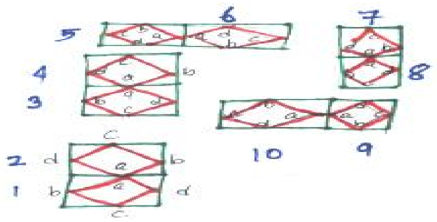
The rules and its explanation are given below:

First we initialize all the rules with label a. i.e $G_0 \triangleq (\mathbb{N}, \phi, \lambda x.a)$

1. Now when two units collide they share their labels and if it matches rule r_1 then a 'dimer' is formed.



(a)



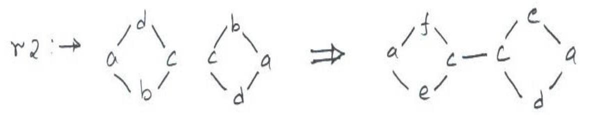
(b)

Figure 7: (a) rule $r_1$and (b) formation of 'dimer'

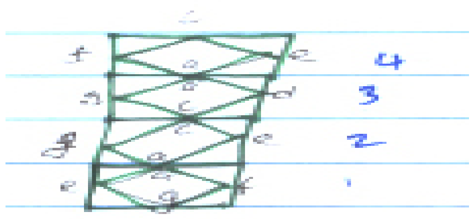
Eventhough the grammer is designed such that the all the rules are commutative i.e. no order is needed to reach the final assemble configuration, we assume that five dimers are formed in one time step $\mathbf{T}_p$ as shown below. This is for the sake of defining a *metric of time* on a non-deterministic algorithm. We have used number in Figure 7(b) only to explain the process. In the actual Simulation there is no indexing used.

2. Now two 'dimers' connect to form a 'Fourmer', using rule r_2. In Figure 8(b) unit 2 is attached to unit 3, forming a line shape subassemble. We change labels so that units attach to this subassemble only on its appropriate positions.Hence here all labels change except a, c and d(on unit 3)

[1]handmade means: Built by logic and trials



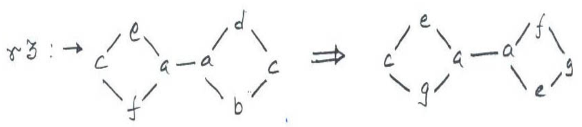
(a)



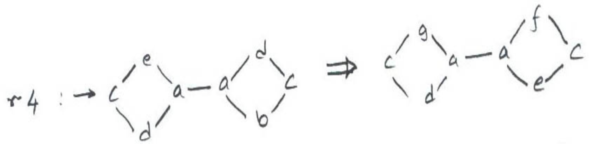
(b)

Figure 8: (a) rule $r_2$and (b) formation of ‘fourmer’

As only two units interacted in rule r_2, we use rule r_3 and r_4 to propogate the labels for new state of fourmer.



(a)



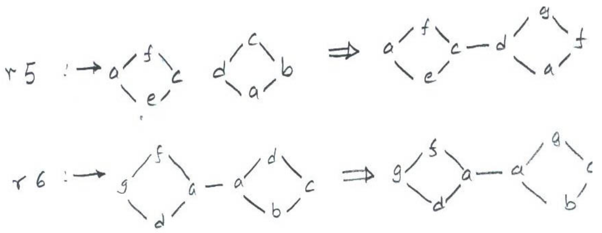
(b)

Figure 9: (a) rule $r_3$and (b) rule r_4 used for propogation

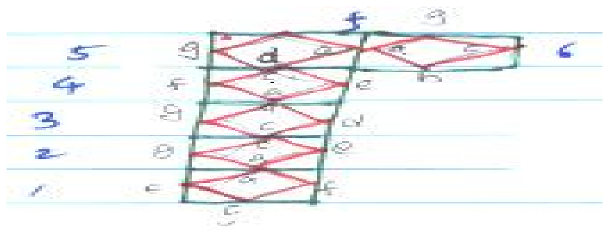
These rules are labelled such that: unit 1 and 2 has no more attachments on it, unit 3 has a possibility of attachment on its RHS, unit 4 has a possibility on the top side.

New labels are embedded in all the units to transfer from one configuration to other.

3. Making a subassemble configuration of shape‘L’, by implementation of rule r_5. And propogation to unit 6, by rule r_6. Again the label on unit 6 is such that robot can only attach on its RHS.



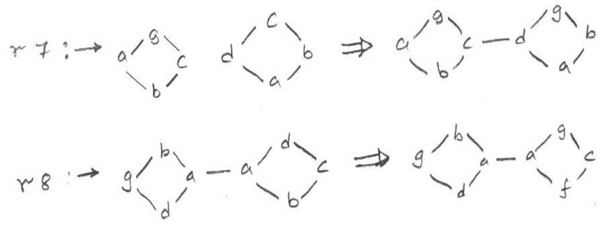
(a)



(b)

Figure 10: (a) rule r_5 and rule r_6 (b) formation of ‘L’

4. Rule r_7 and r_8 are used to connect a dimer to unit 6 on its RHS


(a)

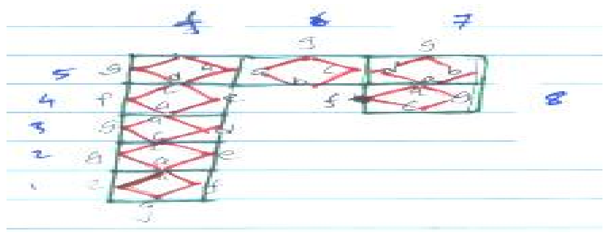
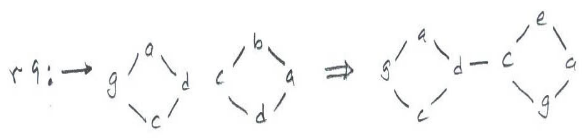

(b)

Figure 11: (a) rule r_7 and rule r_8 (b) formation of a stable subassemble

5. Lastly, Rule r_9 is used to close the subconfigurations with a dimer This completes the shape of 'P'.


(a)

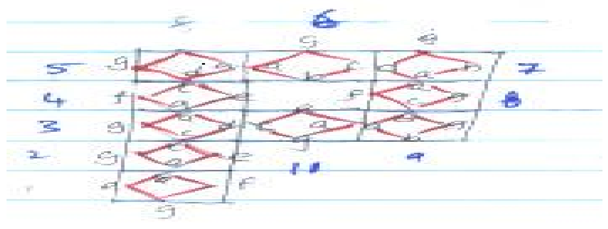

(b)

Figure 12: (a) rule $r_9$and (b) Formation of final assemble 'P'

Hence with rule set Φ of size 9 and labels $\{a, b, \ldots, g\}$ size 6, we were able to construct our final configuration from 10 cellular robots. This shows the effectivenes of graph grammers from memory and faster assembling(due to concurrency) point of views.

However, there is an issue when two 'Fourmer' is formed. This essentially puts the subconfiguration in *deadlock*. This can be solved by using more units than the total needed or by construting a rule set which takes care of deadlock situation.

Following Algorithm was used for simulation of the passive reconfigurable system, in MATLAB, using the above set of grammars.

1. Constructive ($E_L \subset E_R$) grammar rules are stored as strings in two matrix (for LHS and RHS). Size of GrammerMatrix=(number of rules, size of embedded cyclic rules) here (9, 4).
2. Boundaries of wall are defined in which robot moves in Brownian motion. **Our Matlab function is wall2D.m**.

3. *Initialized.*

```
foreach robots do
    initialize its position according to Algorithm 1.;
    initialize its labels according to $G_0 \triangleq (\mathbb{N}, \phi, \lambda x.a)$. Store the current labels in Label matrix with size
    (no.of robot, Cycle size) here (10,4).;
    initialize colors for each of the robot.;
    initialize a matrix Attach with size (no.of units, no.of units, no.ofunits+1).;
    And Attach(ij1)=$\begin{cases} i & \text{if i=j} \\ 0 & \text{Otherwise} \end{cases}$;
    /* we use this matrix for drawing purposes. It keeps track of which robot is leader,
       which one is follower and which sides is attached.Note: The indexing is not used
       for the purpose of reconfiguration. It is still commutative and arbitrary,
       depending on the rules.                                                      */
    The matrix usage is as follows:
      (a) Each column with an index number represents that particular robot.;
      (b) If index is present in its corresponding row than it's a leader, for other indexes
          in that particular row.;
      (c) Always shift the attaching robot, with a higher index to a lower one.;
      (d) The indexes in the face of a matrix represents the side index attached to that particular robot i.e.
          Attachh(ijk) says that j is follower of i if $j \neq i$ and attached to k with an index of side which
          is present in that particular cell.;
      (e) We use the jumpToColumn counter to jump to particular robot to draw as a follower. ;
end
initialize a matrix called Orient. This keeps track of each edge and its orientation. It helps in attaching
two edges which are close to each other and also associates labels with each edge.;
We also define two matrix called relativeDistance and order. The former has the vertex values for each
cell ( this helps while attaching and finding proximity of edges) and the later has combinations for labeling
anticlockwise labeling order.;
```

4. **While** all the units are not attached
   (this happens when $Attach(1, j, 1) \neq 0 \ \forall j$).

5. *Initialize* the graphic windows for drawing the robots.

6. Now,

```
foreach robot do
    $Initialze$SubAssemble $=\phi$, and $AlreadyDrawnUnits = \phi$.;
    Find the followers for robot $i$ and put it in SubAssemble .;
    if $SubAssemble \neq \phi$ then
        Move the robot leader in that group with the Brownian Motion of Algorithm 1.;
        /* Usually the first index is the leader in our case                          */
        Now put that index in AlreadyDrawnUnits.;
        if Size(SubAssemble) >1 then
            /* if there are followers of robot $i$                                        */
            jumpToColum=$SubAssemble_1$.;
            while size(AlreadyDrawn) $\neq$ size(SubAssemble) do
                /* draw all the follower units of that subassemble                        */
                Initialize AttachEdges=robot i's immediate attached robot.;
                /* find Attach(i,jumpToColumn,k)$\neq$ 0 $\forall k$, this is the attached robots index */
                forall immediatly attached robot j do
                    if jumpToColumn=1 then
                        Move the follower robot to a position relative to the leader robot.;
                        /* we use the following function to find the relative postion
                           relativeDistance(:,find(attach(group(1),jumpToColumn,groupSide(s)+1)
                           */
                        /* ==orient(jumpToColumn,:)))  i.e.  the index of the orient matrix,
                           where the attached side is present, is the index of the
                           relativeDistance matrix defined earlier.  And this gives the
                           relative postion                                               */
                        put the index of robot in the alreadyDrawn matrix.;
                        increase the alreadyDrawnCounter.;
                    else if jumpToColumn >1 & alreadyDrawn $\neq$ $AttachEdges_j$ then
                        Move the follower robot to a position relative to the leader robot, using
                        relativeDistance;
                        put the index of robot in the alreadyDrawn matrix.;
                        increase the alreadyDrawnCounter.;
                    end
                end
                if size(SubAssemble)>2 then
                    if $Attach(i, justDrawn, k) \neq 0$ then
                        /* if the follower, to leader, has its own follower.  Then jump to that
                           column and draw its attached robots                                */
                        jumpToColumn counter set to the just drawn robot;
                    else
                        jumpToDrawn counter set to the leader robot to draw its other follower;
                    end
                else if size(SubAssemble)==2 (when a dimer) then
                    jumpToColumn counter set to the just drawn robot;
                end
            end
        end
    end
end
```

7. When all the robots are drawn, we save the figure window as a jpeg file. Then we will use to make a simulation movie. This process of simulation was opted because of the sloppy graphics tools in MATLAB.

8. Now we check the proximity of the robots and attach or detach according to the rules.

```
For robot i
For robot j
InteractingRobot=sort([i j])
ProximityOfRobot=2
Let [M1,N1] = (row of robot i, column of robot i) and similarly [M2,N2]
if the Euclidean distance between robots is $\leq 2$ & robots not already attached then
    find the global position of the edges of i and j.
    /* we achieve this by defining, RelativeSidesDistance, as the relative distance each
       edge with its unit origin. Then put the values in the matrix SideDistance
       depending on each edge's orientation from the Orient matrix.                 */
    find two edges with least Euclidean distance. The edges should be vacant.
    /* Note: if there is ambiguity of two edges having the same least distance, then we
       select one.                                                                    */
    if the edge distance$\leq 1$ then
        Make a string, representing labels on the robot edges.
        /* the string is made with an anticlockwise order, starting from the interacting
           edges. Also the grammer in the stored matrix can have grammar of order robot 1
           then 2 or reverse. Hence we would compare the compares from two matrix of
           strings, one forward and other reverse.                                    */
        ForAll size(LHS GrammarMatrix)
        if CurrentstringMatrix == GrammarMatrix LHS then
            change the labels of the sides with that of the GrammarMatrix RHS. Maintaining the order.
            PropogationRobot=find(Attach(M1,N1,k)$\neq 0$ $\forall k$).
            if PropogationRobot$\neq \phi$ then
                /* Now, as there are propogation rules also included in the set $\phi$, which we
                   have to check and apply before going to next time step. We follow the
                   following routine                                                  */
                forall PropogationRobot do
                    Make a string, representing labels on the robot edges.
                    forall LHS GrammarMatrix do
                        if PropogationCurrentstringMatrix == GrammarMatrix LHS then
                            change the labels of the sides with that of the GrammarMatrix RHS. Maintaining
                            the order.
                        end
                    end
                end
            end
```

9. we follow the same procedure as above for propagating through $Robot_j$.

10. we change the matrix Attach and Orient according to the action above. we give the function changeForm and changeOrient as its argument the robots and its sides involved in the interaction. This matrix, as explained in Algorithm 2 and Algorithm 3, changes the matrix values so that next time step Brownian Motion can take place.

11. As the robots are in unison now. The resultant velocity is the average of the two subassemblies.

12. **Else If** No rules match
    Repel the two interacting robots.
    This is achieved by changing the velocity according to the conservation of momemtum.
    end the *grammer checking loop*

13. **Else If** the edge distance$\not\leq 1$
    repel the two robots which are close to each other, but not interacting.
    end the *proximity of side checking loop*

14. **end both the For loop of step 8**

15. **end the While loop** of step 4

```
Let [M1,N1]=find($Robot_i == attach(:,:,1)$);
[M2,N2]=find($Robot_j == attach(:,:,1)$);
if M2<M1 then
    /* We shift the attached robot to an index with lesser in number.    */
    /* This helps in drawing purposes                                     */
    swap the robots with its corresponding sides;
end
Let N3=all the followers of $Robot_j$;
forall the followers do
    /* shift the robot $Robot_j$ with its followers to the row of $Robot_i$   */
    Attach(M1,N3,1)=Attach(M2,N3,1);
    Attach(M2,N3,1)=0;
end
/* place the index of interacting sides by,                               */
attach(M1,N1,sideOfRobot(2)+1)=sideofRobot1;
attach(M1,N2,sideOfRobot(1)+1)=sideofRobot2;
forall followers of $robot_j$ do
    /* shifting the faces values of the $robot_j$ and its followers          */
    Let O3= the faces(in Matrix Attach) of $robot_j$ and its follower, with an index$\neq$0;
    if O3$\neq$0 then
        forall O3 do
            Attach(M1,N3,O3+1)=attach(M3,N3,O3+1);
            Attach(M2,N3,O3+1)=0;
        end
    end
end
```

Algorithm 2: Algorithm for updating the Attach matrix

```
Let [M1,N1]=find($Robot_i$ == $attach(:,:,1)$);
[M2,N2]=find($Robot_j$ == $attach(:,:,1)$);
N3=find(attach(M1,:,1)==0);
/* N3 are the followers for robot M1                                                    */
N4=find(attach(M2,:,1)==0);
if the $Robot_i$ and $Robot_j$ are dimmers and $robot_j$ has lesser index than $robot_i$ then
    swap the positions of $Robot_i$ and $Robot_j$ ;
end
if $Robot_j$ has more followers than $Robot_i$ then
    /* $Robot_i$'s subassembly gets attached to $Robots_j$'s subassembly (has higher mass) and
       reorients itself accordingly                                                     */
    swap the positions of $Robot_i$ and $Robot_j$ ;
end
/* we make a matrix of clockwise indexing for all the combinations of different sides at
   different positions                                                                  */
Let ClockwiseOrder= [1 2 3 4;2 3 4 1;3 4 1 2;4 1 2 3];
/* we find the current position of interacting side of $Robot_i$ by                     */
CurrentPositionOfSideOfRobot1=find(SideOfRobot1==Orient($Robot_i$,:));
/* Thus the current position of interacting side of $Robot_j$ is                        */
CurrentPositionOfSideOfRobot2=CurrentPositionOfSideOfRobot1+2;
/* we use the following routine to change the orientation of $Robot_j$                  */
PositionOf SidesToChange= find(SideOfRobot2==orderClock(:,1));
SidesPresentAtTheChangedPosition= find(CurrentPositionOfSideOfRobot2==orderClock(:,1)).;
forall four sides of $Robot_j$ do
    Orient($Robot_j$,orderClock(SidesPresentAtTheChangedPosition,counter))=orderClock(PositionOf
    SidesToChange,counter);
end
/* Now we propogate the change in orientation to the followers of $Robot_j$             */
if size(N4)==2 then
    /* the maximum subassembly to reorient is dimmer and N4(2) is the follower of N4(1) */
    CurrentPositionOfSideOfLeader=find(SideOfLeader==Orient($Leader$,:));
    /* Thus the current position of interacting side of $Robot_j$ is                    */
    CurrentPositionOfSideOfFollower=CurrentPositionOfSideOfLeader+2;
    /* we use the following routine to change the orientation of $Follower$             */
    PositionOf SidesToChange= find(SideOfFollower==orderClock(:,1));
    SidesPresentAtTheChangedPosition= find(CurrentPositionOfSideOfFollower==orderClock(:,1));
    forall four sides of $Follower$ do
        Orient($Follower$,orderClock(SidesPresentAtTheChangedPosition,counter))=orderClock(PositionOf
        SidesToChange,counter);
    end
end
```

Algorithm 3: Algorithm for updating the Orient matrix

The Simulation movie, provided in the *enclosed CD*, was made in 'Adobe Premiere Pro'. The stored jpeg images of every instances, made in the MATLAB programme as expalined above, was made in a mpeg format with a rate of 30 frames per second. Hence it took 2.02 minutes in simulation for self assembling in to 'P'. Different Images from the simulation, at critical stages of sub assembling, is shown in Figure 4. The Validation heuristics, for the graph grammer rules, to reach the final configuration $\mathcal{R}(G_0, \Phi)$ is given in Appendix B. The best case assemble time for a particular trajectory σ to reach 'P', according to Definition 3.3.11, is 4 $\mathbf{T_p}$. This happens when concurrently 5 dimers are formed, then two fourmer, then an eightmer and eventually tenmer is reached. The heuristics on $\mathbf{T_p}$ and examples of different σ are explained in Appendix B. The worst case assemble time is 9 $\mathbf{T_p}$. This happens when σ has serial components $\mathcal{H}(G_0, \Phi)$ of subassembling. In our σ the assemble time is 5 $\mathbf{T_p}$. This trajectory of ours is called left-greedy concurrent trajectory. And it gives the best assemble time, for reaching 'P', in our Simulation.

However, there are various parameters in simulation which influence the assemble time. The speed of the Brownian Motion (parameterized as variance of random funtion) and the number of extra Units moving in the environment, are directly proportional to time. While, the boundary of the space is inversely proportional to time. We would like to learn more of these parameters and its statistical nature, in future.

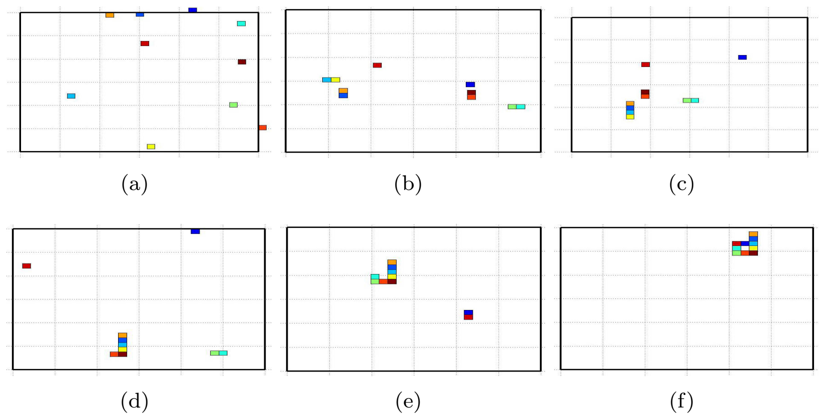


Figure 13: Various formation of subassemblies in Simulation (a) initialized arbitrary position (b)dimer formation (c)fourmer formation (d) sixmer subassembly (e) eightmer and dimer subassemblies and (f) formation of 'P'

5 Discussion

We explained, in Section 3, some of the theories needed to better understand the processes of self reconfiguration. We described the basics of graph grammar in Section 3.3, thereon, implementing C_4 grammar in $\mathbb{R}^2$ space. This has opened a possibility of successfully building larger cyclic grammars in higher spaces i.e. making grammar rule set Φ for Rhombic Dodecahedron modular robots in 3D. However, in 3D one has to also account for orientation of formation while generating Φ. In Appendix B, we suggest some thoughts and theories. This might also answer some of the questions proposed in Section 1, or if, integrated with the graph grammar technique it can better imitate the self-assemble process.

In our attempt, to make a stochastic robotic system which self-assemblies into the required global geometry, we have built a theoretical heuristics which was successfully implemented in simulation. We believe that if further work is done along this direction, one can optimally mimic the self assembling process at micro-scale.

There are many inherent problems in the passive reconfigurable robotic system.

Optimization of rules Given a set of σ, reaching the final configuration $\mathcal{R}(G_0, \Phi)$, which one is optimal. Optimal can be measured in terms of smallest size of Φ, least time to self assemble or least set of labels required.

Stochastic Model Here we have explored only controllable rule set Φ_C. But there may be cases when subassembles break due to collision, in that case, one has to also generate uncontrollable rule set Φ_U. In 3D space one has to consider the orientation of forming, hence there would be another orientation rule set Φ_O. Also there are many dynamics issues in Brownian motion which has to be explored to understand the robotic system, like, the impact of bonds, the attraction basin of each unit, the retention of bonds, friction, etc.

Geometry We still need to investigate on the geometrical structures that can be reached using the graph grammar rules. Also, find ways of incorporating topological problems of deadlock and concavity, in Φ. Further, we plan to extend the graph grammar technique with embedding C_{12} (for Rhombic Dodecahedron) in $\mathbb{R}^3$ (for 3D space).

In Future, we plan to pursue (a) a graph grammar generating algorithm, which gives the rule set Φ when provided with the parameters of geometrical structure,(b)define explicit notion of time, for deterministic self reconfiguration (c) formalize the theory of self organization at all levels and (d) based on this theory build a working prototype.

6 Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor, Dr. Mark Yim, Associate Professor of Mechanical Engineering at the University of Pennsylvania. His imaginative and encouraging discussions guided me throughout the course of work. His wide knowledge in the field of Robotics and patience in explaining me its intricacies were a great value in understanding and completing this research work.

References

[1] Eric Klavins, Robert Ghrist, and David Lipsky. A grammatical approach to self-organizing robotic systems. Submitted to the *IEEE transaction on Automatic Controls*, 2005.

[2] P.J. White, Kopanski, and H. Lipson. Stochastic self-reconfigurable cellular robotics. In *Proceedings of the International conference on Robotics and Automation*,Neworleans, LA, 2004.

[3] Yim, M., D. G. Duff and K. D. Roufas. Polybot: a modular reconfigurable robot. *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pp. 514-520, 2000.

[4] Yim. M., A reconfigurable modular robot with many modes of locomotion. In *JSME Int. Conf. on Advanced Mechatronics*, Tokyo, Japan, 1993.

[5] J. Bishop, S. Burden, E. Klavins, R. Kreisberg, W. Malone, N. Napp, and T. Nguyen. Self-organizing programmable parts. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ Robotics and Automation Society, 2005.

[6] Eric Klavins. Directed self-assembly using graph grammers. In *Foundation of Nanoscience: Self Assembled Architectures and Devices*, Snowbird, UT, 2004.

[7] Eric Klavins. Universal self-replication using graph grammers. In *The 2004 International Conference on MEMs, NANO and Smart Systems*, Banff, Canada, 2004.

[8] Henriques, P.R.Kosar, T.Mernik, M.Pereira, M.J.V.Zumer, V.Zumer. Grammatical Approach to Problem Solving. *Information Technology Interfaces, 2003. Proceedings of the 25th International Conference*, 2003.

[9] A. Kondacs. Biologically Inspired Self- Assembly of Two-Dimensional Shapes Using Globalto- Local Compilation. In *18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[10] M.Yim, J.Lamping, E.Mao, J.G.Chase. Rhombic Dodecahedron Shape for Self-Assembling Robot. In *Xerox PARC, SPL TechReport P9710777*, 1997.

[11] Y. Zhang, K. Roufas, and M. Yim. Software architecture for modular self-reconfiguable robots. In*IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hawaii, October 2001.

[12] Fukuda, T. and Y. Kawauchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. *Proceedings of the 1990 IEEE Conference on Robotics and Automation*, pp. 662-667, 1990.

[13] Rus, D. L. and M. Vona. A physical implementation of the selfreconfiguring crystalline robot.*Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pp. 1726-1733, 1998.

[14] Kotay, K. D. and D. L. Rus. Motion synthesis for the self-reconfiguring molecule.*Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 843-851, 1998.

[15] Kotay, K. D., D. L. Rus, M. Vona, C. McGray. The self-reconfiguring robotics molecule.*Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pp. 424-431, 1998.

[16] K. Sty, W.-M. Shen, and P. Will. On the use of sensors in selfreconfigurable robots. *in Proceedings of the Seventh International Conference on the Simulation of Adaptive Behavior*. B. Hallam, D. Floreano, J. Hallam, G. Hayes, and J.-A. Meyer, Eds., 2002, pp. 4857.

[17] A. Castano, A. Behan, and P. Will. The CONRO modules for selfreconfigurable robots. *IEEE Transactions on Mechatronics*, vol. 7, no. 4, pp. 403409, 2002.

[18] K. Saitou. Conformational switching in self-assembling mechanical systems. *IEEE Transactions on Robotics and Automation*, 15(3):510-520, 1995.

[19] K. Saitou and M. Jakiela. Automated optimal design of mechanical conformational switches. *Artificial Life*, 2(2):129- 156, 1995.

[20] E. Winfree. Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. *Journal of Biomolecular Structure and Dynamics*, 11(2):263-270, May 2000.

[21] H. Wang. Notes on a class of tiling problems. *Fundamenta Mathematicae*, pages 295-305, 1975.

[22] N. Bowden, A. Terfort, J. Carbeck, and G. M. Whitesides. Self-assembly of mesoscale objects into ordered two dimensionalarrays. *Science*, 276(11):233-235, April 1997.

[23] A Terfort and G.M Whitesides. Self-Assembly of an Operating Electrical Circuit Based on Shape Complementarity and the Hydrophobic Effect. *Adv. Mater.* 1998,10, 470-473.

[24] B. Berger, P.W. Shor, L. Tucker-Kellogg, and J. King. Local rule-based theory of virus shell assembly. *Proceedings of the National Academy of Science*, USA, 91(6):7732-7736, August 1994.

[25] R. L. Thompson and N. S. Goel. Movable finite automata (MFA) models for biological systems I: Bacteriophage assembly and operation. *Journal of Theoretical Biology*, 131:152-385, 1988.

[26] A. Kumar, H.A. Biebuyck, G.M. Whitesides. Patterning self-assembled monolayers: Applications in materials science.*Langmuir*, 10(5):1498-1511, May 1994.

A Further on Graph Grammar

Some basics on graph grammar, which will help us to understand its implementation to passive reconfiguration process, is given below.

Paths and Cycles: The components of subassembles may consists of paths and cycles given by $\mathcal{C}(G_0, \Phi) = \{P_1, P_2, \dots\} \cup \{C_3, C_4, C_5, \dots\}$. The stable components of (G_0, Φ) are cycles, $\mathcal{S}(G_0, \Phi) = \{C_3, C_4, C_5, \dots\}$.

Properties:Given a system (G_0, Φ), we bound the size of the reachable and stable sets using a basic topological tool of covering space theory. Some of the axioms and theorems of graph grammars are:

Definition A.0.12 *Given a graph G, an n-fold cover of G is a graph $\tilde{G}$ such that, equivalently:*
1. There exists a label-preserving n-to-1 homomorphism $p : V(\tilde{G}) \to G$ that preserves degree (i.e., the image of an degree k vertex is a degree k vertex).
2. There exists a label-preserving n-to-1 continuous map $p : \tilde{G} \to G$ which is a local homeomorphism.

Theorem A.1 *For (G_0, Φ) an acyclic rule set, $\mathcal{C}(G_0, \Phi)$ is closed under covers. In particular, $\mathcal{C}(G_0, \Phi)$ contains infinitely many isomorphism types of graphs if it contains any graph with a cycle.*

Theorem A.2 *Assume that Φ is an acyclic rule set, and that the stable set contains a component $H \in S(G_0, \Phi)$ but not any of its covers. Then for each edge $e \in E(H)$, there exists a rule in Φ whose left hand side contains a copy of every edge of some cycle in H passing through e.*

Left Greedy Concurrent Trajectories: A partial order $(P, \preceq)$ is a set P with an order relation $\preceq$ that is reflexive, antisymmetric and transitive.

Definition A.0.13 *Let $\sigma \in \mathcal{T}(G_0, \Phi)$ denote a trajectory with partial order $(P, \preceq)$. Define,*

$$\begin{aligned} A_1 &= \{a_i | i \in min(P, \preceq)\} \\ A_2 &= \{a_i | i \in min(P - A_1, \preceq)\} \\ &\vdots \\ A_{j+1} &= \{a_i | i \in min(P - \bigcup_{i=1}^{j} A_i, \preceq)\} \\ &\vdots \end{aligned}$$

Then the left-greedy concurrent trajectory arising from σ is defined to be $\bar{\sigma}$ where $\bar{\sigma}_0 = \sigma_0$ and

$$\bar{\sigma}_0 \xrightarrow{A_1} \bar{\sigma}_1 \xrightarrow{A_2} \dots \xrightarrow{A_j} \bar{\sigma}_j$$

We denote the set of all left-greedy trajectories of a system by $\bar{\mathcal{T}}(G_0, \Phi)$.

These trajectories perform as many commutative steps as possible as early as possible, they are both ' canonical' and 'optimal'.

Lemma A.1 *An arbitrary concurrent trajectory*

$$\bar{\sigma}_0 \xrightarrow{A_1} \bar{\sigma}_1 \xrightarrow{A_2} \bar{\sigma}_2 \xrightarrow{A_3} \dots$$

is left-greedy if and only if for every i and every $a \in A_{i+1}$, the collection $A_i \cup a$ is not commutative.

The left-greedy form of a trajectory has the minimal number of steps (commutative sets of actions) among all trajectories in its similarity class.

More details on the graph grammar theory is provided in Klavin [1]

B Other relevant Theories

Theories from many different fields can be extended to solve the passive self reconfiguration system. One can use potential fields or surface and navigation function around the robots to generate a planned multi-robot motion to form the final configuration. However it may fail when using for large micro scales. Implementation of Reinforcement Learning may be helpful to either generate an optimal grammar or to built a distributed algorithm, which learns the optimal policy for taking specific actions in a particular situation. However, the current techniques in this field are far from successfully mimicking the robotic system. We can also look into a totally different perspective of, initially disassembling to find the optimal sequence of assembling. The basic questions for this could be: where should the cuts be made such that (a) construction of each takes equally least amount of time and (b) there is least concavity, how many of such cuts should be made, and what should be the parameterization of these cuts. Some of these have been explored in the theories of Fair Partitioning and Efficiency.

Validation Heuristics: Following is the heuristics for checking of stable and unique subassemblies $\mathcal{H}(G_0, \Phi)$,. It also assures if the trajectory σ reaches the final required configuration $\mathcal{R}(G_0, \Phi)$.

1. All the Left Hand Side rule unique and finite. All the Right Hand Side rule unique and finite. LHS rule represents the state of two robots at the present time. While the RHS rule represents the state which will be achieved once interacted. Require that no rule in Φ is applicable to the terminal graph.

2. All the rules are reachable and stable. That is, every rule when acted upon gives us a new reachable and stable state. This takes the process nearer to the terminal graph.

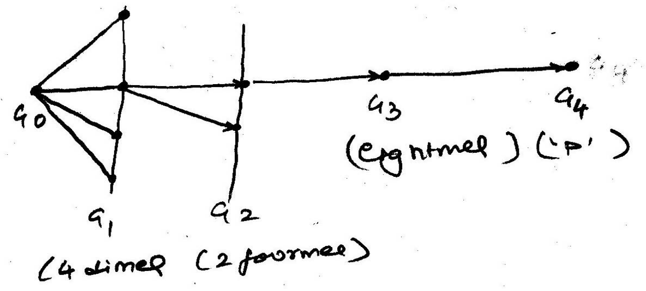
3. These states when achieved concurrently or interleavely gives us at the terminal, reachable and unique, configuration.

We used the above heuristics to check the correctness of our Φ, in Section 4, for the formation of 'P'.

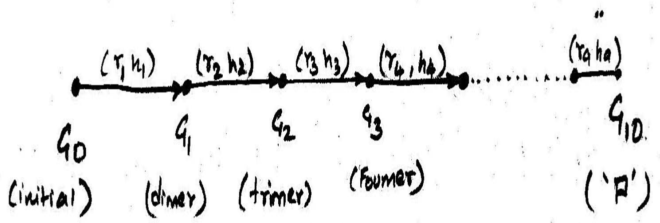
Determining T_P: As the graph grammar rules are non-deterministic, we try to implicitly understand the notion of time by the heuristics of T_P.

We assume that the state is parallely formed by the participating robots. Then the number of states in the trajectory minus the initial state will give us the units for T_P

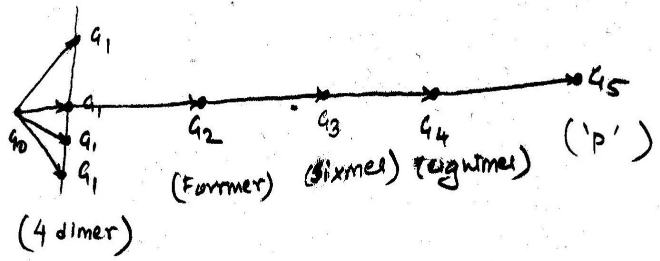
For the case of formation of 'P', we exemplify the above heuristics by considering the worst case of serial trajectory, the best case of concurrent trajectory and our case of left greedy trajectory.



Here it has 4 T_P



Here it has 9 T_P



Here it has 5 T_P

This heuristics gives us an idea of relative time, for the formation of final assembles. Hence a way of finding the optimal trajectory for passive reconfiguration.