Time Complexity of Self-Assembly Process with Misbehaving parts

Ivan Lucatero
Allan Hancock College
ilucater@calpoly.edu

Sebastian Perusset
University of California, Santa Barbara
perusset@umail.ucsb.edu

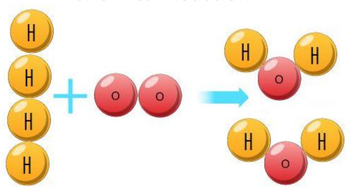
Anahita Mirtabatabaei
University of California, Santa Barbara
mirtabatabaei@engr.ucsb.edu

Introduction

Programmable self-assembly is the process by which autonomous parts are separately programmed to coalesce into a functional system. We study a specific programming employed in the self-assembly of DNA strands. We are interested in computing the time complexity of such process when an unknown number of particles are

Occurrences in Nature

Chemical Reaction



Fire Ants Self-Assemble into Raft to Survive Flood

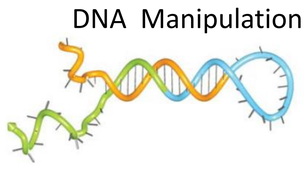
[4]

Future Applications

Swarm Robotics

[1]

DNA Manipulation



[3]

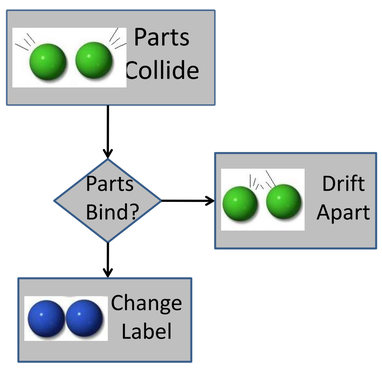
Research Goals

- Simulate robotic and/or biological system with Matlab/Simulink
- Simulation is to include:
  - Time complexity
  - Misbehaving Parts
    - Effects on evolution time
    - Similarity to Initiator

References

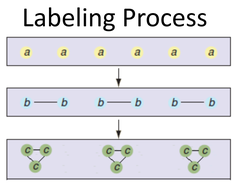
[1] http://ipvs.informatik.unistuttgart.de/BV/symbrion/tiki browse_image.php?imageId=1
[2] Paolo Di Prodi, Lorenzo Cococcia, Matlab Code
[3] http://www.nature.com/nature/journal/v451/n7176/extref/nature06451-s1.pdf
[4] http://commons.wikimedia.org/wiki/File:Fire_ants_cluster_in_water.jpg
[5] http://www.mathworks.com/matlabcentral/fx_files/29262/2/particlebuffer.jpg
[6] Eric Klavins "Programmable Self Assembly" IEEE Control Systems Magazine » August 2007

Approach to Code



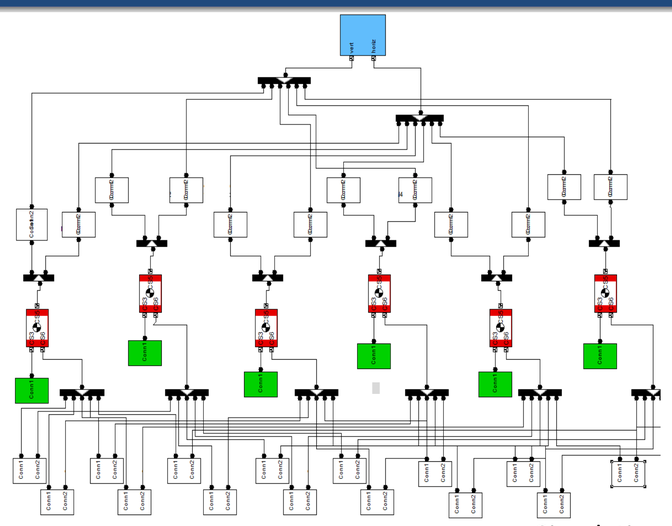
- Conditional programming
- Parts' size identification
- Collision detection

Labeling Process

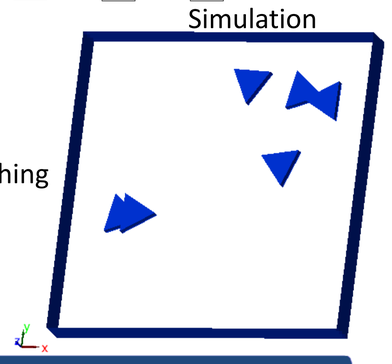


[6]

Simulink Robotic Trial



Simulation



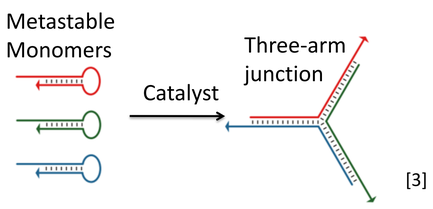
Code above demonstrates the redundancy used by this toolbox. **The simulation** shows the lack of collision detection and of an attaching mechanism.

Matlab Editor Bimolecular System

- Less redundancy with use of loops
- Data is readily accessible
- Easy to vary number of parts
- Very illustrative parts

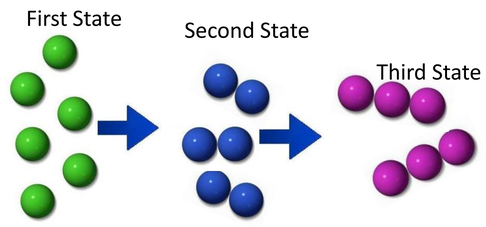
```
for i=1:N
    if (y(i)<=R(i) || y(i)>=H-R(i))
        theta(i)=2*pi-theta(i);
        mindex = find(sstate(i,:) == 1);
        if isempty(mindex) == 0
            theta(mindex)=2*pi*ones(1,
            length(mindex))-theta(mindex);
        end
        %check if the particle is bouncing
         vertical boundaries
        elseif (x(i)<=R(i) || x(i)>=W-R(i))
                theta(i)=pi-theta(i);
                mindex = find(sstate(i,:) == 1);
                if isempty(mindex) == 0
                    theta(mindex)=pi*ones(1,length
                    (mindex))-theta(mindex);
                end
        else
            x0(i)=x(i);
            y0(i)=y(i);
        end
end
```

Focused on DNA Self-Assembly

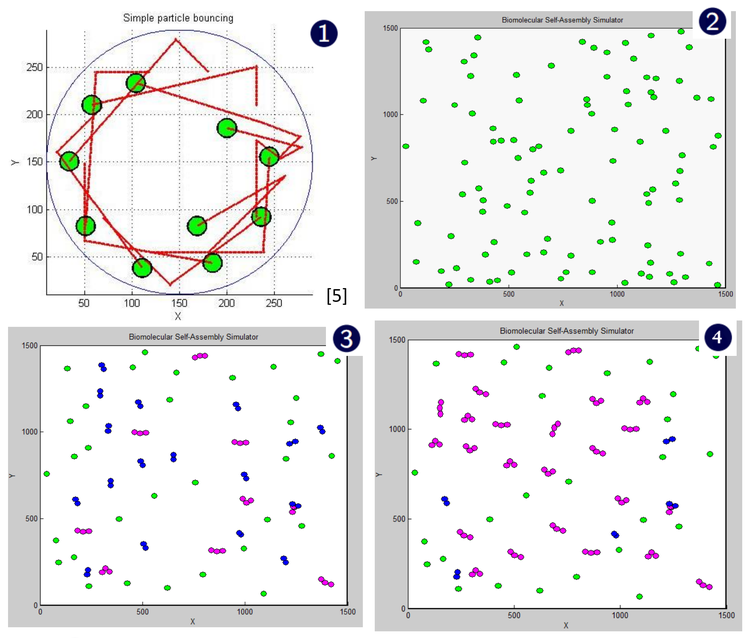


[3]

Analogous Molecular Implementation



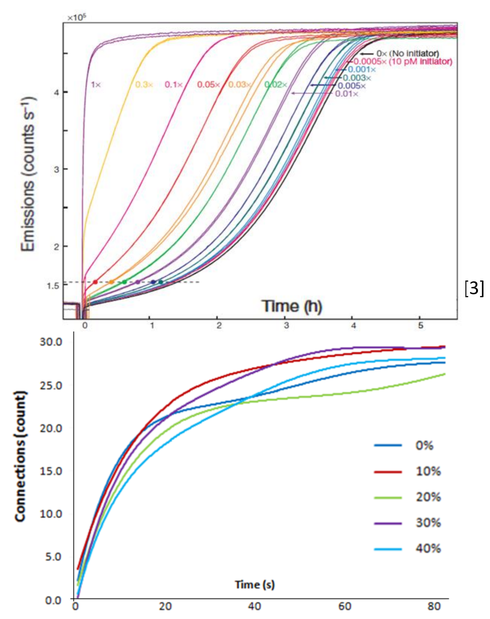
Evolution of Simulation



[5]

1. "Piccolo Particle Simulator" starting point of code [2]
2. First stage of the biomolecular simulator, before any collisions have occurred
3. Second stage, where many parts have changed labels
4. Third stage, where a lot of collisions have occurred

Comparing Results



[3]

Figure on top shows the data gathered by the real experiment. An emission is given off every time a junction is made. The disparity in their trials comes from the variance of the catalyst.

Figure on bottom shows the data from our simulation. We introduce different densities of misbehaving parts to assimilate the catalyst.

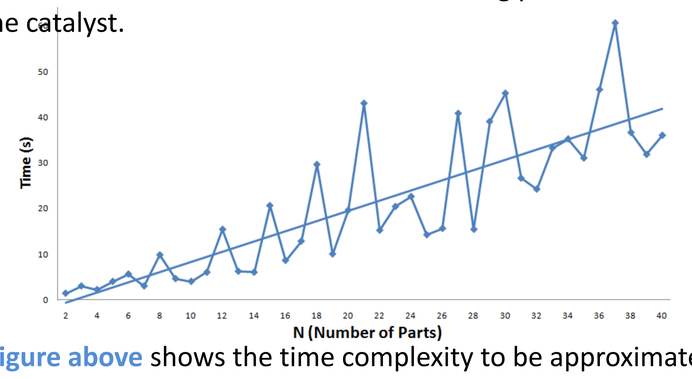
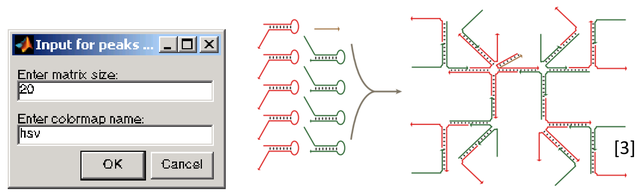


Figure above shows the time complexity to be approximately a linear function of the number of parts, with peaks every third interval.

Future plans

- Make code more user friendly
- More complex shapes
- Include more detailed collisions



[3]