Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time

Damien Woods¹, Ho-Lin Chen², Scott Goodfriend³, Nadine L. Dabby¹, Erik Winfree¹, Peng Yin⁴

¹Caltech, ²National Taiwan University, ³UC Berkeley, ⁴Harvard
Motivation

- Nature computes
  - Engineers are building nanoscale molecular (chemical) computers

- We need a computational theory of self assembly and molecular interactions
- Perhaps more importantly (for us), we can find interesting theoretical problems: computation, geometry, asynchronousity, kinetics, thermodynamics

- Rothemund, PWK. Folding DNA to create nanoscale shapes and patterns. Nature 2006
- Soloveichik, Seelig, Winfree. DNA as a Universal Substrate for Chemical Kinetics. PNAS 2010
- Rothemund, Papadakis, Winfree 2004
Algorithmic self-assembly with tiles: computation and geometry

- Seeman built tiles out of DNA in the laboratory
- Winfree showed that DNA tiles can run algorithms (much like cellular automata, Wang tiles)
- A variety of algorithmic tile assembly systems have been built from DNA
• Can we describe this growth process in abstract terms?
  - Conversion of amorphous fuel into a complicated structure (with a function!), movement of cells with respect to each other, robust to temperature and chemical changes in the environment, development works perfectly while cells are being “pushed around”, cellular differentiation is happening in a distributed manner. Fast. Autonomous.

• Universality of biology

• Can we engineer something of this complexity and size, but built on the molecular scale?
MOLECULAR MOTORS


Gu, Chao, Xiao, Seeman. Nature. 2010

An active self-assembly model: nubots

- **2D triangular grid**
- **Monomers** have a **state** and a **position** on the grid
- The set of all possible states is finite ("monomers are simple"). Positions are pairs of integers.
- **Configuration**: finite set of monomers (i.e. monomer states and positions at some time instant)
- Configurations change over time using a finite set of **rules** (next slide)
An active self-assembly model

Some example rules:

1. Make/break a flexible bond
2. Change states
3. Change a rigid bond to a flexible bond or vice-versa, change states
4. Appearance
5. Disappearance
6. Position change in the w direction
7. Movement rule
8. Position change in the -w direction

Damien Woods, ITCS 2013
An active self-assembly model

Movement is **not** permitted to break **bonds**: A **rigid** bond between 2 monomers breaks if their relative position changes. A **flexible** bond between 2 monomers breaks if one of them moves so that they are no longer adjacent.

Movement rule example:

**Position change rule**

- **Current configuration**
- **Successful application of movement rule**
- **Movable set empty**
- **Failure to apply movement rule** (movement would break flexible/rigid bonds)
An active self-assembly model

Rules are applied over time in an *asynchronous* fashion.

System evolves as a continuous time Markov process.

For a given monomer, the time, in seconds, until an applicable rule is applied (or “fires”) is an exponentially distributed random variable with mean 1.

- For a given monomer, if a rule is applicable it fires in expected time 1.
- If there are \( n \) monomers with applicable rules, then the expected time until *some* rule fires is \( 1/n \).
- If there are \( n \) monomers with applicable rules, that can be applied independently, then the expected time for *all* rules to fire is \( O(\log n) \).
Rules: \( r_1 = (1, 1, \text{null}, \overline{w}) \rightarrow (2, 1, \text{rigid}, \overline{w}) \), \( r_2 = (1, 2, \text{rigid}, \overline{y}) \rightarrow (1, 1, \text{null}, \overline{y}) \), \( r_3 = (1, 1, \text{rigid}, \overline{w}) \rightarrow (1, 1, \text{rigid}, \overline{y}) \).

Expected time = 3(track length)
EXAMPLE: ROTATING ARM

Rule: \( r1 = (1, 1, \text{rigid}, \overline{w}) \rightarrow (1, 1, \text{rigid}, \overline{y}) \).

Expected time = \( O(\log (\text{arm length})) \)
EXAMPLE: GROW A LINE

Rule set for length $k+1$ line: $\{ r_i \mid r_i = (i, \text{empty}, \text{null}, \bar{x}) \rightarrow (0, i - 1, \text{rigid}, \bar{x}) \}$, where $k \geq i > 0$.

How much time?
Expected time $= \text{total line length } - 1 = k$

How to do better?
FAST GROWTH OF A LINE

a  Construction overview
   a1  Initial configuration
   a2  Subroutines
      (1)  k > x > 0
      (2)  x > 0

b  Rules for Subroutines (1) & (2)
   a.k  r1  k.x 0  x  r2  b.1
   b.k  r6  x-1  x-1  r7  x-1
   b.x  r11  x-1  x-1  r12  x-1

C  Example execution of Subroutine (2)
   r2  b.1  r3  b.2  r4  b.2  r5  b.2
   r7  b.5  r8  b.6  r9  b.7  r10  b.8

D  Example configuration
FAST GROWTH OF A LINE

Example execution of Subroutine (2)

Example configuration

States decrement upon insertion:
\[ x \rightarrow x-1, \quad x \in \{k, k-1, \ldots, 2, 1\} \]

Highly parallel! Expected time = \( O(\log n) \) to make a line of length \( n = 2^k \)
FAST GROWTH OF A LINE

$t = \text{time}$

$s = \text{rule applications (steps)}$

Growth of a line of length $2^6 = 64$

Growth of a line of length $2^6 = 64$
FAST SYNCHRONIZATION

How to detect when the line is finished growing?

1. Rigid bond
2. Flexible bond
3. Insertion
4. Insertion complete
5. Shift monomer
6. Synchronized
7. Shift does not apply
8. Shift can now apply

Expected time $O(\log n)$ for a line of length $n$.

Damien Woods, ITCS 2013
FAST GROWTH OF A COUNTER

Counts to $n$ in $O(\log^2 n)$ expected time
Builds an $n \times \log n$ rectangle
FAST GROWTH OF A SQUARE

$n \times n$ square in $O(\log n)$ time!
This example: 16 x 16 square
WHAT CAN ACTIVE SELF-ASSEMBLY DO?

We’ve exhibited a selection of specific shapes and behaviors

We’ve seen certain shapes can be built exponentially fast

What is the system capable of in general?

Can it make more general/complicated shapes quickly?

How complicated can a such shape be?
RESULT 1: EFFICIENTLY COMPUTABLE SHAPES

Result 1: An arbitrary connected computable 2D shape of size \( \leq n \times n \) can be constructed:

- in time \( O(\log^2 n + t(|n|)) \) and using \( O(s + \log n) \) states,
- where \( t(|n|) \) is the time for a program size \( s \) Turing machine to compute, given the pixel position index of as a length \( |n| = O(\log n) \), whether the pixel is present in the shape.

(1.1) Binary index of pixel \( n^2 \)

(2a) Turing machine head (carries out a computation to decide if monomer (pixel) should be in final shape)

(4) unshaded monomers are deleted

(1.1) Binary index of 1st pixel

(1) Binary counter assembly

(2b) Turing machine colors the monomers: shaded (grey) monomers will be in final shape, unshaded monomers will not

(3) \( n \) monomers folded into a square

(4) Carving

Damien Woods, ITCS 2013
COMPUTABLE SHAPES

Simulator by Scott Goodfriend
EFFICIENTLY COMPUTABLE PATTERNS: CAN WE DO BETTER FOR SIMPLER PATTERNS?

• In Result 1, the Turing machine computation time $t(|n|)$ is a bottleneck to fast (polylog($n$)) computation. There is nothing we can do about this.
  - So let’s restrict attention to shapes with polylog($n$) time computable pixels.

• In Result 1, we had “non-monotone” growth (e.g., long length folded into a square, & lots of extra space used by Turing machines):
  - Can we restrict growth to be contained in an $n \times n$ “womb” without affecting computational power?

• In Result 1, we used synchronization over long distances:
  - Can we compute without it?

Can we build large structures?

Quickly? In-place?
Without synchronization?
While being pushed around?

Yes!


Damien Woods, ITCS 2013
RESULT 2: EFFICIENTLY COMPUTABLE PATTERNS

**Result 2:** An arbitrary computable 2D pattern of size $\leq n \times n$, whose pixels are computable in (polynomial) time $O(\log^l n)$ and (linear) space $O(\log n)$, can be constructed:

- in time $O(\log^{l+1} n)$,
- using $O(s + \log n)$ states where $s$ is the program size of a Turing machine to compute, given the position index of a pixel, whether the pixel should be present in the shape,
- this can be done in-place (monotone growth) in a region of size $n \times n$,
- and without using long-range synchronization.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$i$</th>
<th>$j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$$
\text{RESULT 2: EFFICIENTLY COMPUTABLE PATTERNS}
$$
RESULT 2: EFFICIENTLY COMPUTABLE PATTERNS
RESULT 2: EFFICIENTLY COMPUTABLE PATTERNS
SUMMARY

• Using active self assembly we can:
  - Model simple walkers and other motors
  - Quickly grow exponentially large computable shapes
  - Quickly grow exponentially large patterns in place (without synchronization, and while being “pushed around”)

• The addition of the movement rule to a cellular automaton is sufficient for exponential growth

• Future work:
  - model other growth processes: dynamic processes, rearrangement, persistence length, tensile strength, movement, signaling with/without perfect synchronization
  - General time analysis tools?
  - Computational complexity: Are polylog time nubots = NC?

• Future work: implement the model in the lab!

Full paper:
D. Woods, H.L. Chen, S. Goodfriend, N. Dabby, E. Winfree, P. Yin.
Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time.
THANKS!

Caltech Center for Biological Circuit Design

NSF
CCF-1219274
CCF-1162589

NSF
Molecular Programming Project (0832824)

Full paper:
D. Woods, H.L. Chen, S. Goodfriend, N. Dabby, E. Winfree, P. Yin.
Active Self-Assembly of Algorithmic Shapes and Patterns in Polylogarithmic Time.
Damien Woods, ITCS 2013
FIN