Towards Indestructible Molecular Robots

Ilir Çapuni, Anisa Halimi, and Dorjan Hitaj

Epoka University, Department of Computer Engineering,
Rr. Tirana-Rinas, Tirana, Albania
{icapuni,ahalimi,dhitaj12}@epoka.edu.al

Abstract. We study the fault-tolerance of the recently proposed nubot model of molecular-scale self-assembly, which generalizes asynchronous cellular automata to have non-local movement. In particular, we propose a set of rules that ensures that a particular shape saves its information and its shape forever even when independently of each other, certain cells change their state or die with some small probability.

1 Introduction

As computers are growing in importance every day, the never-ending increase in their performance and the corresponding decrease in their price seems to be near the end. For this, alternative models of computation are being looked for, molecular computation being one promising land to look for new models of computation.

The recently-introduced nubot model by Woods et al. in [8] brings movement and computation together in one model.

The model can be understood as asynchronous and non-deterministic cellular automata model where cells — here called *monomers* — have the capability to move relative to each other, and to form bonds between them. It also allows creation and destruction of cells and allows a random uncontrolled motion.

This model is an excellent framework to study the ultimate limitations and capabilities of the growth and rearrangements at a molecular scale. Its complexity is studied in [2].

Given the probabilistic nature of the computation at this scale and the imprecise behavior of molecular elements, certain transitions would inevitably be faulty. Such faults can set a state of a cell to an arbitrary one, create a cell at any site, move a cell to the neighboring site, or even kill a cell completely. Such mishaps would inevitably interfere with computation and movement, leading to undesirable results.

We are interested in a model where faults occur independently at random with some small probability.

Fault-tolerance has been studied in cellular automata. A simple rule for two-dimensional cellular automata that keeps one bit forever even though each cell can fail with some small probability was given in [7]. A 3-dimensional reliably computing cellular automaton was constructed in [6]. All simple one-dimensional cellular automata appear to be "ergodic" (forgetting everything about their initial configuration in time independent of the size). The first, complex, nonergodic cellular automaton was constructed in [3], and improved upon in [5].

The question of fault-tolerant computation with Turing machines (where arbitrarily large bursts may occur with correspondingly small probability) is raised in 1987 by Manuel Blum, and was solved in [1].

1.1 Our Result

In this paper we will focus our attention to the reliable storage problem. The main concern of this problem is to store information such that losing any small part of it is not fatal: it can be restored using the rest of the data.

Our construction encodes the information onto a convex shape, and uses the majority voting to constantly "battle" with the altered bits.

2 Nubot Model

Nubot model consists of two dimensional grid of monomers. Monomer is the basic unit of this model. It is defined as state-labeled disks of unit diameter centered on the grid point.

A set of rules specifies how adjacent monomers will interact with each-other. Monomers have state and are connected to each other through bonds. After applying a set of rules, they can change their state or the type of bond between them. These rules are applied asynchronously.

Monomers can move relative to each other by applying a certain kind of rules. The movement is applied locally, but then it propagates through all the system.

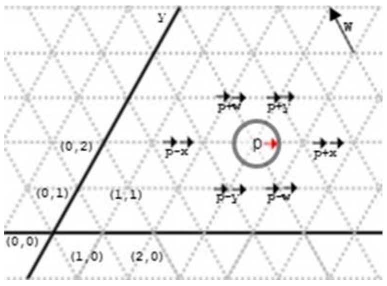

Fig. 1. Coordinate system

The model uses a two-dimensional triangular grid with a coordinate system using $\boldsymbol{x}$ and $\boldsymbol{y}$, but we define a third axis $\boldsymbol{w}$ to make the notation easier as shown in Figure 1. The coordinates of a certain monomer are defined by using only $\boldsymbol{x}$

and $\boldsymbol{y}$. The axial directions are the unit vectors along the grid axes:

$$D = \{\boldsymbol{w}, -\boldsymbol{w}, \boldsymbol{x}, -\boldsymbol{x}, \boldsymbol{y}, -\boldsymbol{y}\}.$$

A pair $\boldsymbol{p} \in \mathbb{Z}^2$ is called a *grid point*, and it has 6 neighbors $\{\boldsymbol{p} + \boldsymbol{u} \mid \boldsymbol{u} \in D\}$. Each grid point –also called a *site* – has at most one monomer.

Two adjacent monomers are connected to each-other by a *rigid* or *flexible bond*, or no bond exist between them (the *null bond*). In the figures in this paper, a flexible bond is depicted as a small empty circle and a rigid bond is depicted as a solid disk. Rigid bonds are more stable and can not be broken easily.

A *configuration* is a finite set of monomers at distinct locations and the bonds between them, and is assumed to define the state of the entire grid at some instance of time.

2.1 Rules

Two adjacent monomers can interact through an interaction rule,

$$r = (s1, s2, b, \boldsymbol{u}) \to (s1', s2', b', \boldsymbol{u'}),$$

where $s1, s2 \in \Sigma \cup \{\text{empty}\}$ are monomer states, empty denotes the lack of a monomer; $b \in \{\text{flexible}, \text{rigid}, \text{null}\}$ is the bond type between two monomers, $\boldsymbol{u}$ is the relative position of the $s2$ monomer to the $s1$ monomer. The same thing applies for the right part of the arrow. If $s1$ or $s2$ is empty, the bond b between them is null, also if either or both $s1'$, $s2'$ is empty, then b' is null.

The interaction rule represents the contents of the monomers before and after the rule r is applied. By applying certain rules, we can add monomers, one or both monomers can be removed, adjacent monomers can change state and bonds, or one monomer can move relative to another monomer.

A rule where $\boldsymbol{u} \neq \boldsymbol{u'}$ and none of $s1, s2, s1'$, and $s2'$ is empty is a *movement* rule. To apply such a rule, one monomer is chosen nondeterministically to be stationary and is called the *base*, and the other one will be called the *arm*. This work will not consider this kind of rules.

If monomer X is the arm, then from its current position $\boldsymbol{p}(X)$ it will move to its new position $\boldsymbol{p}(X) - \boldsymbol{u} + \boldsymbol{u'}$, and change its state to $s2'$. However, the arm may be attached to a group of other monomers, and for this we need to define the *movable set* of monomers A and B contained in configuration C,

$$\mathcal{M}(C, A, B, \boldsymbol{v})$$

to be the minimal set that can be moved in direction $\boldsymbol{v}$ without disrupting existing bonds or causing collisions with other monomers. The movement is performed if the movable set is not empty.

In Figure 2 we give some examples that illustrate state and bond changes. To change the states of the monomers as shown in the figure we apply $r1 = (2, 4, \text{null}, \boldsymbol{x}) \to (1, 5, \text{null}, \boldsymbol{x})$. To make a flexible bond, $r2 = (0, 0, \text{null}, \boldsymbol{x}) \to (0, 0, \text{flexible}, \boldsymbol{x})$ is applied. $r3 = (1, 1, \text{rigid}, \boldsymbol{x}) \to (1, 1, \text{null}, \boldsymbol{x})$ is used to break

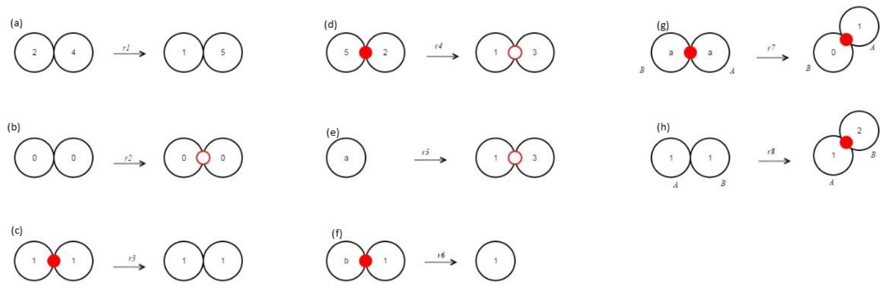

Fig. 2. Examples of monomer interaction rules. (a) Change states. (b) Make a flexible bond. (c) Break a rigid bond. (d) Change a rigid bond to a flexible bond and change the states. (e) Appearance of a monomer. (f) Disappearance of a monomer. (g) Movement in the $\boldsymbol{w}$ direction. (h) Movement in the $-\boldsymbol{w}$ direction.

a rigid bond. To change the bond type from rigid to flexible and the states of the monomers in the same time we use $r4 = (5, 2, \text{rigid}, \boldsymbol{x}) \rightarrow (1, 3, \text{flexible}, \boldsymbol{x})$. Appearance of a new monomer is done by applying $r5 = (a, \text{empty}, \text{null}, \boldsymbol{x}) \rightarrow (x, 1, \text{flexible}, \boldsymbol{x})$ and disappearance using $r6 = (b, 1, \text{rigid}, \boldsymbol{x}) \rightarrow (1, \text{empty}, \text{null}, \boldsymbol{x})$. There are two possible choices of movement for the monomer depending on arm and base selection. To move the monomer in a certain direction $r7 = (a, a, \text{rigid}, \boldsymbol{x}) \rightarrow (0, 1, \text{rigid}, \boldsymbol{y})$ is applied.

If $s_i \in \{s1, s2\}$ is empty and s_i' is not, then a new monomer has appeared. If one or both monomers from non-empty monomers become empty, the rule induces the monomer disappearance.

2.2 Toom's Rule

Toom's rule is an example of a two-dimensional "stable" cellular automaton. Each small square in the grid has a value of 1 or 0.

At each instance of time, a cell checks its current state, the neighboring square to the North and the neighboring square to the East. If the majority of these states is 1, then the state of the current square becomes 1; otherwise it becomes 0. The rule has been proved to be stable (see [4]).

3 Our Construction

In this section, we first present the way how do we encode the bit that we want to save onto a shape, and then we devise the rules that can withstand random noise defined above. Recall, we assume that independently at random, a monomer can die, change its state, or a new monomer can be created at an empty site.

To save one bit forever we encode it to a geometric shape in which each monomer is set to the value that we want to save (see Figure 3).

Starting from such a configuration, the Toom rule by its design, will "dissolve" the monomers that are in a minority: If the number of the monomers with

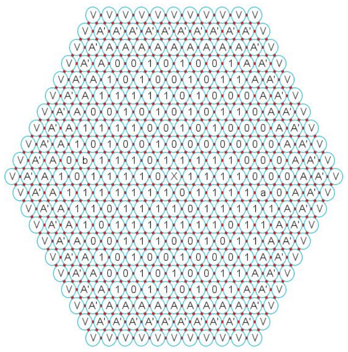

Fig. 3. A shape wrapped with the layers of monomers that consist the shell

state 1 in the initial configuration is greater than the number of the monomers with state 0, bit 1 is saved; otherwise bit 0 is saved. The shape of the storage will have a specific kind of a “monomer fabric”: We use rigid bonds between the monomers.

To apply the plain version of the Toom’s rule, we need the following nubot rules:

$$
\begin{aligned}
r1 &= (1, 0, 1, E) \rightarrow (a, 0, 1, E)\\
r2 &= (a, 0, 1, NE) \rightarrow (0, 0, 1, NE)\\
r3 &= (a, 1, 1, NE) \rightarrow (1, 1, 1, NE).
\end{aligned}
$$

The state a is an intermediate state that helps us to determine properly the state of the monomer in the next step. When rule $r1$ is applied, the monomer checks its own state and the neighbor in the east. Then by using $r2$ and $r3$ we take into consideration the two possible states of the neighbor monomers in the northeast, which results in the majority of the states of itself, east and northeast neighbor. The above rules are given for state 1. Rules pertaining to the state 0 are written analogously.

Since our shape holding the information is of finite size, we need to take care of the edges. We solve this problem by applying Toom’s rule in E and NE or W and SW or SE and NW or any combination between the first ones with the second ones since the rule that is going to be applied, is chosen undeterministically. By applying the rules in these directions even the monomers on the corners can change their state according to the Toom’s rule.

Another deterministic solution to this problem would be to update the cells in each direction in a controlled fashion as follows. We first apply the Toom’s rule by using NE, E neighbors. Once all the cells have performed these updates, we will apply the rules from the opposite direction, that is, we apply the Toom’s rule using W, SW. Once this is completed, in the same fashion as above, we apply the same rules using E, SE and its opposite direction W, NW. Finally, once this completes, we will apply the Toom's rule using NW, NE and its opposite direction SE, SW.

In the nubot model the rules are not applied at each monomer simultaneously at a certain instance of time. In the nubot model, rules are applied asynchronously and as many times as possible. When none of the rules can be applied the evolution stops. At that point all the monomers in the configuration have reached state 0 or 1.

3.1 Shell

To preserve the shape we need to create a "skin": We add a three layer shell around the shape made of monomers with specific state. So essentially, an initial configuration resembles the one shown in Figure 3.

3.2 Filling the Holes

In this section we will show how we achieve to maintain the shape after applying random noise on the shape and its shell. Recall, the noise causes that some monomers change their state to an arbitrary one or to vanish completely.

If a monomer is deleted by the noise, then a new monomer with state X is added instead of it. State X represents a kind of an "undifferentiated state" of a monomer. Later, application of other rules will set its value according to the values of its neighbors.

To add a monomer with state X in the interior part of the shape we use $r7 = (1, \text{empty}, 0, E) \rightarrow (1, X, 1, E)$. We apply rules similar to $r7$ on any of the possible six directions in order to fill all the gaps that can be created in the shape. Then we change the bonds to rigid ones throughout the shape with the rules like $r8 = (X, 1, 0, SW) \rightarrow (X, 1, 0, SW)$.

The monomer with state X determines its new state according to the state of its neighbors. If it sees 0 in the East, it behaves like it was a monomer with state 1 and sets its state to a; otherwise it sets it to b. We do this using $r9 = (X, 0, 1, E) \rightarrow (a, 0, 1, E)$ and $r10 = (X, 1, 1, E) \rightarrow (b, 1, 1, E)$.

The states a and b are intermediate states in the process of "differentiation" of a newly created monomer with the state X to a monomer with a state 0 or 1.

If one of the monomers of the shell is deleted, we add a new monomer using $r11 = (A, \text{empty}, 0, NE) \rightarrow (A, D, 1, NE)$. If the state of a monomer is D, and it has a neighboring monomer with a state in $\{1, 0, a, b\}$, then, its state changes to X by $r12 = (D, 0, 1, W) \rightarrow (X, 0, 1, W)$. Further, if a monomer with a state in $\{a, b, 1, 0\}$ has a neighbor with state A', its state switches to D, that is $r13 = (a, A', 1, NE) \rightarrow (D, A', 1, NE)$. If a monomer whose state is V neighbors a monomer with the state in $\{0, 1, a, b\}$, then it changes its state to X by rule $r14 = (V, b, 1, SW) \rightarrow (X, b, 1, SW)$. When it sees X, it becomes D by the application of $r15 = (V, X, 1, E) \rightarrow (D, X, 1, E)$

To determine properly the state of the new monomers, Toom's rule is applied by the monomers comprising the shell, using $r16 = (V, A, 1, E) \rightarrow (V', A, 1, E)$,

$r17 = (V', A, 1, SE) \rightarrow (A', A, 1, SE)$, $r18 = (D, V, 1, W) \rightarrow (D', V, 1, W)$, and $r19 = (D', V, 1, NW) \rightarrow (A', V, 1, NW)$.

Finally, if the noise causes a monomer to be created on an empty site with the empty neighborhood, that cell will die, after it has confirmed that it is surrounded by empty sites. Stability of the Toom's rule is established in [4].

3.3 Test Results

In Figure 4 we start from a damaged triangle and the simulation reaches a non-damaged version of it. Similarly, in Figure 5 we repeat the same for the case of a square.

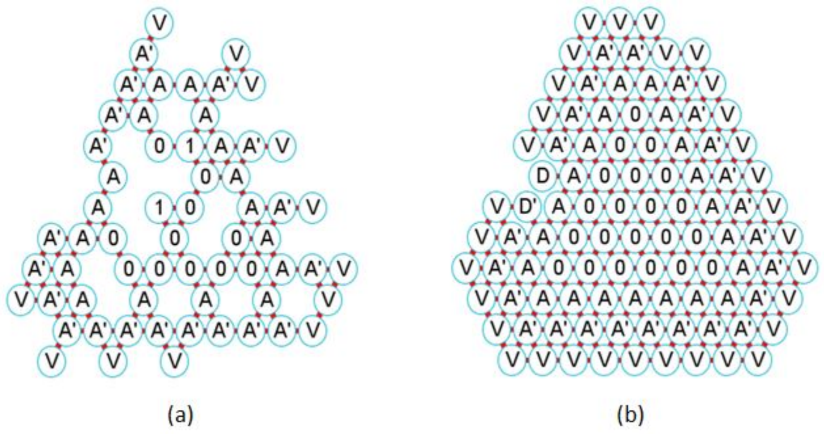

Fig. 4. Damaged triangle configuration. (a) Initial configuration(40.2%). (b) Final Configuration(3.92%)

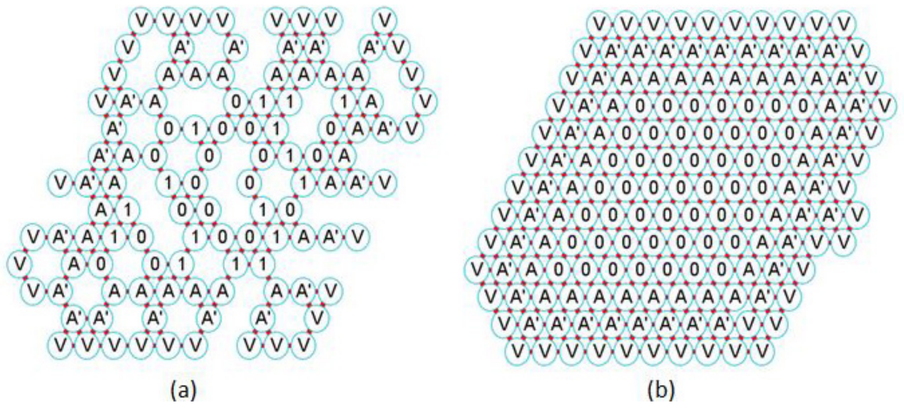

Fig. 5. Damaged square configuration. (a) Initial configuration(31%). (b) Final Configuration(3%)

From these examples, we notice that the shape has been able to save the information and its shape. Even when it has been subject to a considerable amount of random noise, the percentage of damage that is found in the final configuration is very small.

4 Conclusions and Future Work

This poster presents the preliminary results on how to solve the storage problem on a nubot system, that does not make use of movement. The expected time that is needed to reach a configuration where the information and the shape is reconstructed needs to be estimated.

It is still not known if nubot model has the intrinsic universality property: for any nubot program N, there is a nubot program F_N that acts just like N, but with some $m \times m$ scale-up in space, and a moderate slowdown in time, where m and the slowdown are independent of N and its input. Consequently, it is a major open question if the above problem is solvable when F_N is subjected to faults that occur independently of each other with some small probability.

References

1. Çapuni, I.: A Fault-tolerant Turing Machine. PhD thesis, Boston University, Commonwealth avenue, Boston MA 02215 (2012)
2. Chen, M., Xin, D. and Woods, D.: Parallel computation using active self- assembly. In: Soloveichik, D. and Yurke, B., (eds.) DNA Computing and Molecular Programming. vol. 8141 of Lecture Notes in Computer Science, pp. 16–30, Springer International Publishing (2013)
3. Gács, P.: Reliable computation with cellular automata. Journal of Computer System Science, 32(1), 15–78, Conference version at STOC' 83 (1986)
4. Gács, P.: A new version of toom's proof. Technical report, Boston University Computer Science Department (1995)
5. Gács, P.: Reliable cellular automata with self-organization. Journal of Statistical Physics, vol. 103(1/2), pp. 45–267, See also arXiv:math/0003117 [math.PR] and the proceedings of STOC '97 (2001)
6. Gács, P. and Reif, J.: A simple three-dimensional real-time cellular array. In Proceedings of the seventeenth annual ACM symposium on Theory of computing, pp. 388–395, ACM (1985)
7. Toom, A.L.: Stable and attractive trajectories in multicomponent systems. Advances in Probability, 6(1), 549–575 (1980)
8. Woods, D., Chen, H.-L., Goodfriend, S., Dabby, N., Winfree, E. and Yin, P.: Active self-assembly of algorithmic shapes and patterns in polyloga- rithmic time. In Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, ITCS '13, pp. 353–354, New York, NY, USA, ACM (2013)