Design Automation of Polyomino Set That Self-Assembles into a Desired Shape

Yuta Matsumura
Department of Robotics, Graduate School of Engineering, Tohoku University, Sendai, Japan
matsumura@molbot.mech.tohoku.ac.jp

Ibuki Kawamata
Department of Robotics, Graduate School of Engineering, Tohoku University, Sendai, Japan
Natural Science Division, Faculty of Core Research, Ochanomizu University, Tokyo, Japan
kawamata@molbot.mech.tohoku.ac.jp

Satoshi Murata
Department of Robotics, Graduate School of Engineering, Tohoku University, Sendai, Japan
murata@molbot.mech.tohoku.ac.jp

Abstract

The problem of finding the smallest DNA tile set that self-assembles into a desired pattern or shape is a research focus that has been investigated by many researchers. In this paper, we take a polyomino, which is a non-square element composed of several connected square units, as an element of assembly and consider the design problem of the minimal set of polyominoes that self-assembles into a desired shape. We developed a self-assembly simulator of polyominoes based on the agent-based Monte Carlo method, in which the potential energy among the polyominoes is evaluated and the simulation state is updated toward the direction to decrease the total potential. Aggregated polyominoes are represented as an agent, which can move, merge, and split during the simulation. In order to search the minimal set of polyominoes, two-step evaluation strategy is adopted, because of enormous search space including many parameters such as the shape, the size, and the glue types attached to the polyominoes. The feasibility of the proposed method is shown through three examples with different size and complexity.

2012 ACM Subject Classification Applied computing → Systems biology; Applied computing → Chemistry; Hardware → Biology-related information processing

Keywords and phrases DNA polyomino, DNA nanostructure, DNA tile, Agent based simulation, Self-assembly, Combinatorial optimization, Simulated annealing

Digital Object Identifier 10.4230/LIPIcs.DNA.2020.8

1 Introduction

As a method of creating artificial nanostructures, programed self-assembly of molecules is attracting attentions [3, 5, 7]. Since DNA has an excellent property of double helix formation between complementary base sequences, it is thought to be the most promising molecule for this purpose. One of the methods to make DNA nanostructures is called DNA tile [11, 12, 2]. In this method a unit called DNA tile composed of a few short DNA strands assemble into a large two-dimensional nanostructure. The DNA tile is a rectangle molecule having sticky ends (i.e. bonding edges with sequence specificity) on its sides. By arranging the sticky ends, it is possible to program the connectivity between the tiles. We can design a tile set to assemble periodic or aperiodic patterns, while the production cost depends on the complexity of the tile set (e.g. the number of sticky end types and the number of tile types), also the more complicated the tile set, the lower the quality and yield of the obtained assembly. From this point of view, the problem of finding the smallest tile set that forms the desired pattern (Pattern self-Assembly Tile-set Synthesis, PATS) has been studied [4, 1].

In this paper, we deal with self-assembly problem of a non-square element composed of several connected square units called a polyomino. We propose an algorithm to search for the minimum set of polyominoes required to assemble a desired outer shape. By using a polyomino as an element of assembly, it becomes possible to utilize the shape complementarity of the polyomino, in addition to the complementarity of sticky ends on the polyomino. This enables us to make relatively complex shapes also given as connected polyominoes. Since DNA origami technique enables us to make various three-dimensional shapes, it is expected that such non-square-shaped element made by DNA origami will allow us to construct a large structure with desired shape.

In the following sections, we consider the problem of finding the smallest polyomino set to fill a given shape. In Section 2, we introduce an assembly model that simulates the stochastic process of polyomino assembling. Section 3 describes a searching method for the simplest polyomino set that forms the target shape. In Section 4, we show the results of automatic design for target shapes with different size and complexity to verify the validity of the proposed method. Section 5 gives discussions.

2 Self-assembly model of polyominoes

2.1 Outline

This section explains the mathematical model of a polyomino and then introduces a stochastic simulation technique to predict the behavior of polyominoes. Unlike the abstracted kinetic tile assembly model [9, 10, 6], we employ an agent-based technique for the simulation.

Our model is illustrated in Fig. 1. The following summarises the outline.

- *Polyomino* is represented as a set of connected *square units*.
- An integer number named *glue type* is assigned to each side of the square unit.
- *Agent* is defined as a naive set of polyominoes.
- At the beginning of simulation, agents are randomly distributed over discretized space.
- In each step of simulation, agents can translate or rotate in the space.
- Potential energy computed from the interactions among polyominoes is minimized through the agent-based Monte Carlo simulation.

2.2 Square unit

A polyomino consists of several connected square units. To define the square unit, we need some prerequisite notations. $\mathrm{D} = \{\mathrm{N}, \mathrm{E}, \mathrm{S}, \mathrm{W}\}$ is a set of four cardinal directions (north, east, south, west) such that $\hat{\mathrm{N}} = \mathrm{S}, \hat{\mathrm{S}} = \mathrm{N}, \hat{\mathrm{E}} = \mathrm{W}, \hat{\mathrm{W}} = \mathrm{E}$. The neighboring cell of $\boldsymbol{x} = (x, y) \in \mathbb{N}^2$ in the direction $d \in \mathrm{D}$ is given by $\mathrm{coord}(\boldsymbol{x}, d)$ assuming a periodic boundary condition of the square lattice space.

$$\mathrm{coord}(\boldsymbol{x}, d) = \begin{cases} (x, y - 1 \bmod m_{\mathrm{cell}}) & (d = \mathrm{N}) \\ (x + 1 \bmod n_{\mathrm{cell}}, y) & (d = \mathrm{E}) \\ (x, y + 1 \bmod m_{\mathrm{cell}}) & (d = \mathrm{S}) \\ (x - 1 \bmod n_{\mathrm{cell}}, y) & (d = \mathrm{S}) \end{cases},$$

where $m_{\mathrm{cell}}, n_{\mathrm{cell}} \in \mathbb{N}$ are the total number of rows and columns in the lattice, respectively. Hereafter, m_{cell} and n_{cell} are both set to 32.

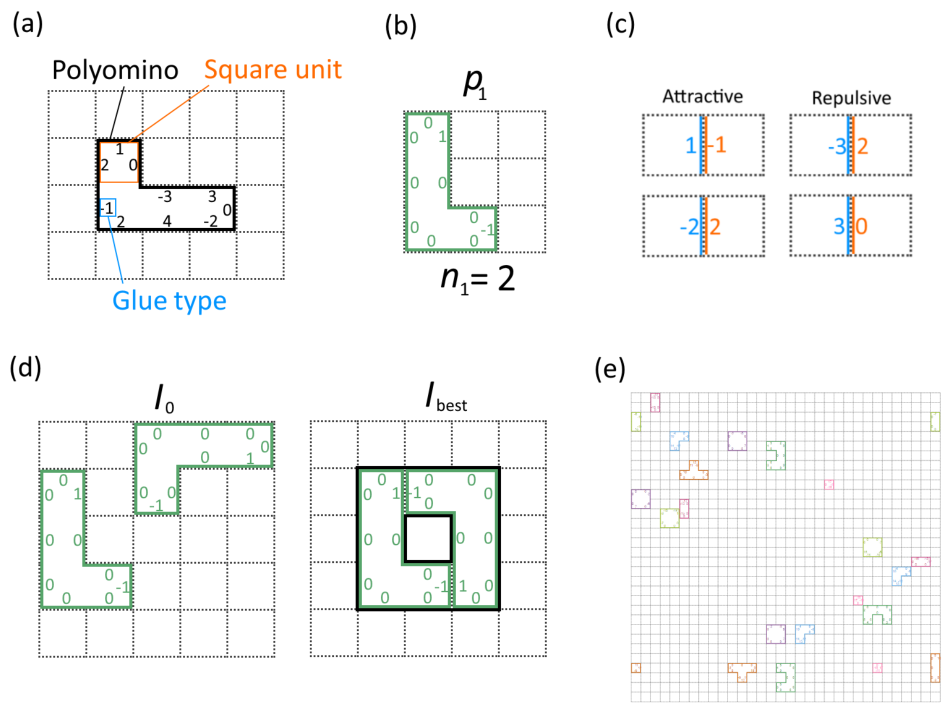

Figure 1 (a) Model of polyomino. (b) polyomino sets. (c) Interaction between square unit. (d) Initial simulation state I_0 and most stable simulation state I_{best}. (e) Snapshot of the simulation.

Square unit u is defined as a tuple of a position $x, y \in \mathbb{N}$ and a map $\mathrm{g} \in \mathbb{Z}^{\mathrm{D}}$ that gives the glue type of the cardinal direction D (i.e. $u = (x, y, \mathrm{g})$). The pair (x, y), and coordinates x and y of a square unit $u = (x, y, \mathrm{g})$ can be obtained by $\mathrm{pos}(u) = (x, y)$, $\mathrm{pos_x}(u) = x$, $\mathrm{pos_y}(u) = y$, respectively. Similarly, the glue type of a square unit $u = (x, y, \mathrm{g})$ in the direction $d \in \mathrm{D}$ can be obtained by $\mathrm{gl}(u, d) = \mathrm{g}(d)$. Non-zero glue types g_1 and g_2 are *complementary* when $g_1 + g_2 = 0$ stands.

2.3 Polyomino

A polyomino p is defined as a nonempty set of connected square units (i.e. $p = \{u_1, u_2, \ldots\}$ such that $\forall u_i, u_j \in p, \exists u'_1 = u_i, u'_2, \ldots, u'_k = u_j \in p, \forall l \in \{z \in \mathbb{N} | 1 \leq z \wedge z \leq k\}, \exists d \in \mathrm{D}, \mathrm{pos}(u'_{l+1}) = \mathrm{coord}(\mathrm{pos}(u'_l), d))$. To avoid an overlap, we assume that a square unit $u_1 \in p_1$ never belongs to other polyomino p_2 (i.e. $\forall u_1 \in p_1, u_2 \in p_2, u_1 = u_2 \to p_1 = p_2$). The center of mass of a polyomino p is defined as $\mathrm{c_M}(p) = (\mathrm{c_x}(p), \mathrm{c_x}(p))$, where $\mathrm{c_x}(p) = \mathrm{round}(\sum_{u \in p} \mathrm{pos_x}(u)/|p|)$ and $\mathrm{c_y}(p) = \mathrm{round}(\sum_{u \in p} \mathrm{pos_y}(u)/|p|))$. The nearest integer is obtained by $\mathrm{round}(x) \in \mathbb{Z}$ from a real number $x \in \mathbb{R}$.

2.4 Movement of polyomino

A polyomino is capable of performing a movement $m \in \mathrm{M_{poly}}$, which is a map from polyominoes to polyominoes. Here, we define 7 possible movements : translation to the north, east, south or west, or rotation to the left, back or right. Here "back rotation" means rotation of 180 degrees. The set of these movements is defined as $\mathrm{M_{poly}} = \{\mathrm{north, east, south, west, right, back, left}\}$. Formal description of the movement is given in Appendix A.1.

Polyominoes p_1 and p_2 are *isomorphic* ($p_1 \equiv p_2$) when there are finite movements that can move p_1 to p_2, which is defined as $p_1 \equiv p_2 \leftrightarrow \exists n \in \mathbb{N}, \exists m_1, m_2, \ldots m_n \in \mathrm{M}_{\mathrm{poly}}, m_1 \circ m_2 \circ \ldots \circ m_n(p_1) = p_2$. When p_1 and p_2 are not isomorphic, they are called *non-isomorphic*.

2.5 Polyomino species

The concept of polyomino set was ambiguously used so far to illustrate the goal of our research. Here, we introduce the formal definition of *polyomino species*, which is more accurate to describe the polyomino set. A polyomino species is a multiset of quotient set of polyominoes by the isomorphic relationship $\equiv$, which is not a naive set of polyomino. Namely, polyomino species P can be expressed as a set of tuples of representative polyomino p_i and its occurrence count n_i (i.e. $P = \{(p_1, n_1), (p_2, n_2), \ldots\}$). The number of representative polyominoes is denoted as $|P|$, and the set of glue types in P is defined as $\mathrm{Gl}(P) = \{\mathrm{gl}(u, d) \in \mathrm{D}, u \in p, (p, n) \in P\}$. We say polyomino set to simply explain the target problem, although it formally means polyomino species throughout this paper.

2.6 Agent

In the proposed simulation model, we introduce a concept of agent which represents a naive set of polyominoes [8]. Instead of applying the movement to each polyomino, we move the agent in order to improve energy convergence.

Agent $a = \{p_1, p_2, \ldots\}$ is a non-empty set of polyominoes, connected by the complementary glue types. (i.e. $a = \{p_1, p_2, \ldots\}$ such that $\forall p_i, p_j \in a, \exists p'_1 = p_i, p'_2, \ldots, p'_k = p_j \in a, \exists u \in p'_{l+1}, \exists u' \in p_l, \forall l \in \{z \in \mathbb{N} | 1 \leq z \wedge z \leq k\}, \exists d \in \mathrm{D}, \mathrm{pos}(u) = \mathrm{coord}(\mathrm{pos}(u'), d) \wedge \mathrm{gl}(u, \hat{d}) + \mathrm{gl}(u', d) = 0 \wedge \mathrm{gl}(u', d) \neq 0$). We define a set of square units in an agent a as $\mathrm{U}(a)$ and the number of square units in the agent a as $|\mathrm{U}(a)|$. Similar to the polyomino, there are also 7 movements $\mathrm{M}_{\mathrm{agent}}$ for the agent (see Appendix A.1).

At the beginning of the simulation, each polyomino is assumed to belong to a different agent, and is able to move independently. Through the simulation process, agents can *merge* or *split*, resulting in a unified movement of several polyominoes. Details of the process is described in the following.

2.7 Simulation state

Simulation state $I = \{a_1, a_2, \ldots\}$ is defined as a set of agent at specific time step. We define a naive set of polyominoes in the simulation state I as $\mathrm{P}(I) = \{p | p \in a, a \in I\}$, and a set of square units as $\mathrm{U}(I) = \{u | u \in p, p \in \mathrm{P}(I)\}$.

The initial simulation state I_0 is defined for a given polyomino species $P = \{(p_1, n_1), (p_2, n_2), \ldots\}$. There are n_i copies of polyomino p_i without any overlap at the beginning. Namely, $\forall u_1, u_2 \in \mathrm{U}(I_0), \mathrm{pos}(u_1) = \mathrm{pos}(u_2) \rightarrow u_1 = u_2$.

2.8 Cluster

A *cluster* c is a naive set of polyominoes in a simulation state I, such that there are no polyomino $p \in \mathrm{P}(I) \backslash c$ neighboring to c. This is formalized as $\forall p_1 \in c, \forall p_2 \in \mathrm{P}(I) \backslash c, \forall u_1 \in p_1, \forall u_2 \in p_2, \forall d \in \mathrm{D}, \mathrm{coord}(\mathrm{pos}(u_1), d) \neq \mathrm{pos}(u_2)$. Note that a cluster does not necessarily have to contain polyominoes with matching glues. Unlike agents that can translate and rotate, the cluster only refers to an static assembly of polyominoes. They are used to evaluate the state of the simulation. When a simulation state I is given, the set of all clusters are defined as $\mathrm{cl}(I)$.

The same movements $\mathrm{M}_{\mathrm{agent}}$ of agent can be applied to cluster (see Appendix A).

2.9 Potential energy

During the simulation, the total *potential energy* is evaluated as a sum of local energy gained from interactions among the square units. When two units are not neighboring, there is no local energy between them. If they are located in the neighboring cells, there is an attractive force between them when the facing glue types are complementary, otherwise, there is a repulsive force. Given two square units u_1 and u_2, the local energy between them $\mathrm{e}_{\mathrm{unit}}(u_1, u_2)$ is defined as

$$\mathrm{e}_{\mathrm{unit}}(u_1,u_2) = \begin{cases} 0 & (\forall d \in \mathrm{D}, \mathrm{coord}(\mathrm{pos}(u_1), d) \neq \mathrm{pos}(u_2)) \\ e_{\mathrm{att}} & (\exists d \in \mathrm{D}, \mathrm{coord}(\mathrm{pos}(u_1), d) = \mathrm{pos}(u_2) \wedge \mathrm{gl}(u_1, d) + \mathrm{gl}(u_2, \hat{d}) = 0) \wedge \mathrm{gl}(u_1, d) \neq 0) \\ e_{\mathrm{rep}} & (\text{otherwise}) \end{cases},$$

where e_{att} and e_{rep} are local energy caused by the attractive and the repulsive forces, respectively. Hereafter, we use $e_{\mathrm{att}} = -11$ and $e_{\mathrm{rep}} = 2$, referring to a reported agent-based simulation method [8].

The potential energy $\mathrm{e}_{\mathrm{poly}}(p_1, p_2)$ between polyominoes p_1, p_2 is a sum of all energy of the square units in them. Namely,

$$\mathrm{e}_{\mathrm{poly}}(p_1,p_2) = \begin{cases} \sum_{u_1 \in p_1, u_2 \in p_2} \mathrm{e}_{\mathrm{unit}}(u_1,u_2) & (p_1 \neq p_2) \\ 0 & (\text{otherwise}) \end{cases}.$$

Similarly, the potential energy $\mathrm{e}_{\mathrm{agent}}(a_1, a_2)$ between agents a_1, a_2 can be defined as

$$\mathrm{e}_{\mathrm{agent}}(a_1,a_2) = \begin{cases} \sum_{p_1 \in a_1, p_2 \in a_2} \mathrm{e}_{\mathrm{poly}}(p_1,p_2) & (a_1 \neq a_2) \\ 0 & (\text{otherwise}) \end{cases}.$$

For convenience, we also define the potential $\mathrm{e}_{\mathrm{in}}(a)$ of a given agent a as

$$\mathrm{e}_{\mathrm{in}}(a) = \sum_{p_1,p_2 \in a} \mathrm{e}_{\mathrm{poly}}(p_1,p_2)/2.$$

The total potential energy $\mathrm{e}_{\mathrm{state}}(I)$ of a given simulation state I is defined as

$$\mathrm{e}_{\mathrm{state}}(I) = \sum_{a_1,a_2 \in I} \mathrm{e}_{\mathrm{agent}}(a_1,a_2)/2.$$

2.10 Time development of the simulation state

By using the agent-based simulation, we are able to minimize the total energy of a simulation state. From a state I_i of i-th step of the simulation, the next state I_{i+1} can be obtained by the algorithm shown in Fig. 2. First, an agent a_{sel} is randomly selected from the state I_i, and one of the three actions (i.e. split, move or merge) takes place to update the state. If none of the actions are admissible, I_i becomes the next state.

- Split of agent
  Namely, the agent a_n with an energy $\mathrm{e}_{\mathrm{in}}(a_n)$ is split into two, if it is composed of n $(n \geq 2)$ polyominoes that satisfy

  $$\exists i \in \mathbb{N},\, 1 < i \leq n, \mathrm{e}_{\mathrm{in}}(a_n)/n > \mathrm{e}_{\mathrm{min}}(i,n)/i,$$

  where $\mathrm{e}_{\mathrm{min}}(i,n)$ is the smallest energy of the agent with i square units among all the simulation states before the current n-th step. Namely,

  $$\mathrm{e}_{\mathrm{min}}(i,n) = \min(\bigcup_{j \leq n} \{\mathrm{e}_{\mathrm{in}}(a) | a \in I_j \wedge |\mathrm{U}(a)| = i\}).$$

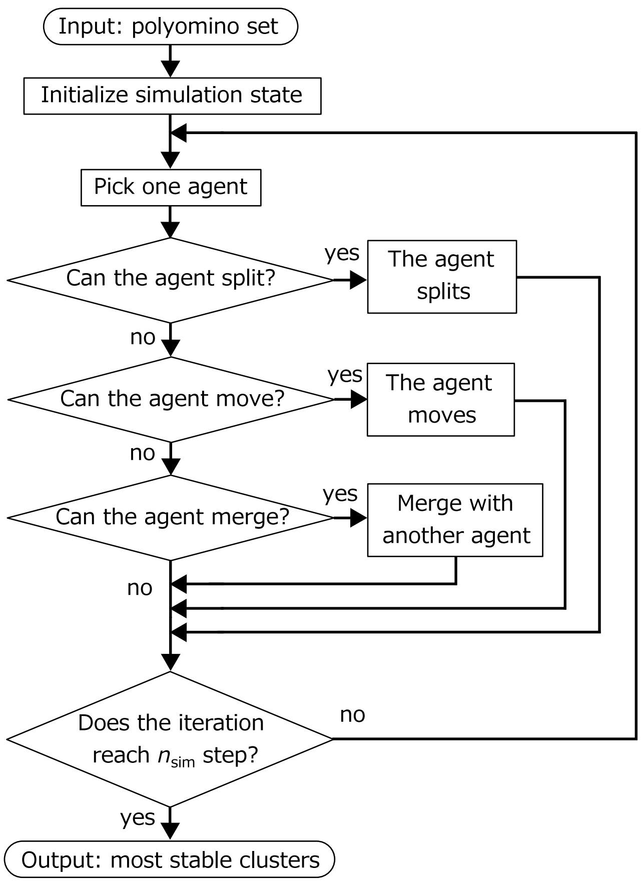

Figure 2 Flowchart of simulation.

An agent can be split in several ways. One polyomino p_1 is removed from the agent and becomes a new agent when p_1 has the worst (biggest) contribution to the potential. The polyomino p_1 satisfies $\forall p_2 \in a_{\text{sel}}, \mathrm{e}_{\text{agent}}(a_{\text{sel}}, \{p_1\}) \leq \mathrm{e}_{\text{agent}}(a_{\text{sel}}, \{p_2\})$. When splitting takes place, the next simulation state I_{i+1} becomes $I_i \backslash \{a_{\text{sel}}\} \cup \{a_{\text{sel}} \backslash \{p_1\}, \{p_1\}\}$.

- Move of agent

When the agent cannot split, one or several polyominoes try to move together as a unified agent. When the agent a_{sel} takes a move $m \in \mathrm{M}_{\text{agent}}$, a simulation state transits to $I^m_{i+1} = I_i \backslash \{a_{\text{sel}}\} \cup \{m(a_{\text{sel}})\}$. As there are 7 movements, there are 7 possible simulation states I^{north}_{i+1}, I^{east}_{i+1}, I^{south}_{i+1}, I^{west}_{i+1}, I^{right}_{i+1}, I^{back}_{i+1}, I^{left}_{i+1}. One of them is stochastically selected as the next simulation state I_{i+1} with the probability $\mathrm{P}(I_i, I^m_{i+1})$ given as

$$\mathrm{P}(I_i, I^m_{i+1}) = \begin{cases} \frac{\min(1, \exp((\mathrm{e}_{\text{state}}(I_i) - \mathrm{e}_{\text{state}}(I^m_{i+1}))/k_{\mathrm{B}}\tau_{\text{sim}}))}{|\mathrm{M}_{\text{agent}}|} & (\text{condition A}) \\ 0 & (\text{otherwise}) \end{cases},$$

where τ_{sim} is a temperature parameter introduced to overcome the energetic barrier (i.e. local minima), and k_B is the Boltzmann constant. Hereafter, we use $\tau_{\text{sim}} k_B = 5$, which is an empirically good value for the energy convergence. “Condition A” means that there is no overlap of agents as a result of the movement and the agent a_{sel} is not rotational symmetry, which can be formalized as

$$(\forall p_1, p_2 \in I^m_i, p_1 \neq p_2, \forall u_1 \in p_1, u_2 \in p_2, \mathrm{pos}(u_1) = \mathrm{pos}(u_2) \rightarrow u_1 = u_2) \wedge (\mathrm{m}(a_{\text{sel}}) \neq \mathrm{m}(a_{\text{sel}})).$$

- Merge of agents
  If all the possible movements increase the potential energy (i.e. $\forall m \in \mathrm{M}_{\text{agent}}, \mathrm{P}(I_i, I_{i+1}^m) < 1/|\mathrm{M}_{\text{agent}}|$) and also none of the movements are chosen by the calculated possibilities, the agent then try to merge with a neighboring agent. This condition implies that there is an attractive interaction between a_{sel} and the neighboring agent.
  The agent a_{sel} merges with another agent a_1 that can make the assembly most stable in respect to the potential energy. The agent a_1 satisfies $\forall a_2 \in I_i, \mathrm{e}_{\text{agent}}(a_{\text{sel}}, a_1) \leq \mathrm{e}_{\text{agent}}(a_{\text{sel}}, a_2)$. When the agent a_{sel} merges with the agent a_1, the simulation state becomes

$$I_{i+1} = I_i \backslash \{a_{\text{sel}}, a_1\} \cup \{a_{\text{sel}} \cup a_1\}.$$

The most stable simulation state is predicted by iterating the above state transition for n_{sim} times. When a polyomino species P is given, the resulting assembly is defined as a set of clusters $\mathrm{A}(P)$ such that

$$\mathrm{A}(P) = \mathrm{cl}(I_{\text{best}}) \qquad (\exists I_{\text{best}} \in X, \forall I \in X, \mathrm{e}_{\text{state}}(I_{\text{best}}) \leq \mathrm{e}_{\text{state}}(I)),$$

where X is the set of simulation state through the simulation ($X = \{I_0, I_1, \ldots, I_{n_{\text{sim}}}\}$).

3 Design automation

3.1 Criteria

By using the simulation model in Section 2, we solve a *shape self-assembly polyomino set* (SAP) problem, which is formalized as follows.

- The target assembly is given as a *shape* defined as a finite set of positions $s = \{(x_1, y_1), (x_2, y_2), \ldots\}$ with the size $m_{\text{shape}} = \max(x_1, x_2, \ldots) - \min(x_1, x_2, \ldots)$, and $n_{\text{shape}} = \max(y_1, y_2, \ldots) - \min(y_1, y_2, \ldots)$.
- If a polyomino species P can construct a shape s through self-assembly, then P is said to be an polyomino species of s.
- The size of polyominoes in the polyomino species P is less than or equal to $m_{\text{poly}} \times n_{\text{poly}}$, and must be smaller than that of the target shape s.
- There are no limitations on the number of representative polyominoes $|P|$ and glue types $|\mathrm{Gl(P)}|$.
- An optimum polyomino species for a shape of finite size is the polyomino species of minimum cardinality (i.e., with the smallest number of representative polyominoes).
- The SAP (shape self-assembly polyomino species) problem is defined as a problem to find the optimum polyomino species for a given finite-size shape.

3.2 Outline

To tackle the SAP problem, we employ a simulated-annealing algorithm which is one of the meta-heuristics approaches. The flowchart of the algorithm is given in Fig. 3. An initial polyomino species is randomly generated from a given shape s and evaluated by the simulation. In our optimization strategy, a polyomino species is rated better when the predicted assembly is closer to the target shape, and also the number of representative polyominoes and the number of glue types are smaller. A polyomino species is gradually improved by repeating evolutionary process.

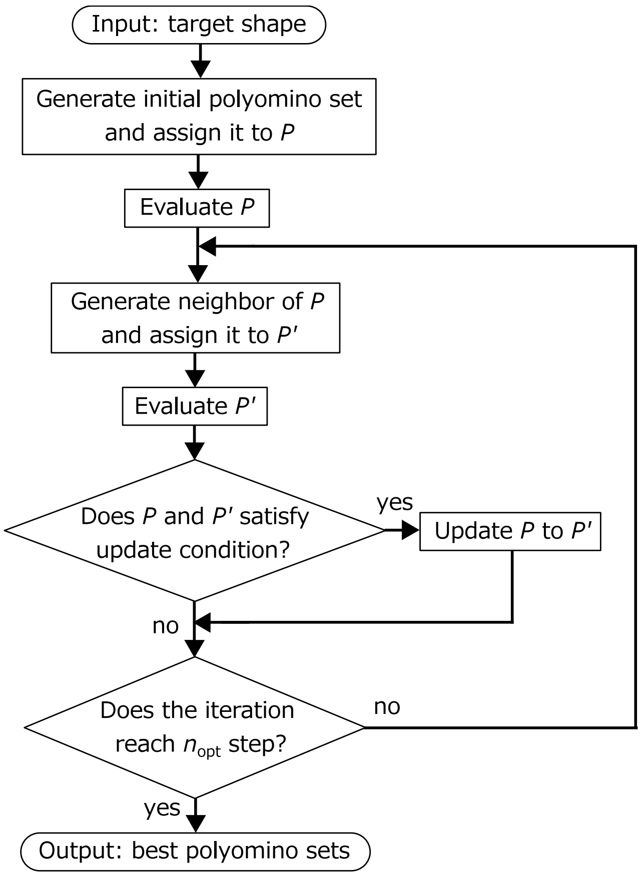
Input: target shape
Generate initial polyomino set and assign it to P
Evaluate P
Generate neighbor of P and assign it to P'
Evaluate P'
Does P and P' satisfy update condition?
yes
Update P to P'
no
Does the iteration reach n_{opt} step?
no
yes
Output: best polyomino sets

Figure 3 Flowchart of automatic design.

3.3 Evaluation of polyomino species

To evaluate a polyomino species P, we introduce an inaccurate but light-cost function $\mathrm{loss}_{\mathrm{light}}$ and an accurate but heavy-cost function $\mathrm{loss}_{\mathrm{heavy}}$. The function is named "loss" because the smaller the value, the better the polyomino species. In order to minimize the time of computation, the light-cost function is first used for rough evaluation, then heavy-cost function is further used when it meets a certain criteria.

The light cost function is defined as

$$\mathrm{loss}_{\mathrm{light}}(P) = |P|^2 + \frac{1}{2}|\mathrm{Gl}(P)|.$$

When $\mathrm{loss}_{\mathrm{light}}(P) < \alpha_{\mathrm{th}}$ holds, the heavy-cost function is applied, where $\alpha_{\mathrm{th}} \in \mathbb{R}$ is a threshold parameter updated when $\mathrm{loss}_{\mathrm{heavy}}(P, s)$ is computed. By introducing α_{th}, the algorithm can efficiently search for polyomino species with a smaller loss value than current best value. The initial value of α_{th} is $|s|^2 + 2|s|$, which is the maximum value of $\mathrm{loss}_{\mathrm{light}}(P)$ for given target shape s. The algorithm to update α_{th} is

$$\alpha_{\mathrm{th}} := \begin{cases} \alpha_{\mathrm{th}} & (\mathrm{loss}_{\mathrm{heavy}}(P, s) - \mathrm{loss}_{\mathrm{light}}(P) > 0) \\ \min(\alpha_{\mathrm{th}}, \mathrm{loss}_{\mathrm{light}}(P)) & (\text{otherwise}) \end{cases} .$$

The condition indicates that α_{th} is updated when all the clusters in $\mathrm{A}(P)$ have exactly the same shape as the target s.

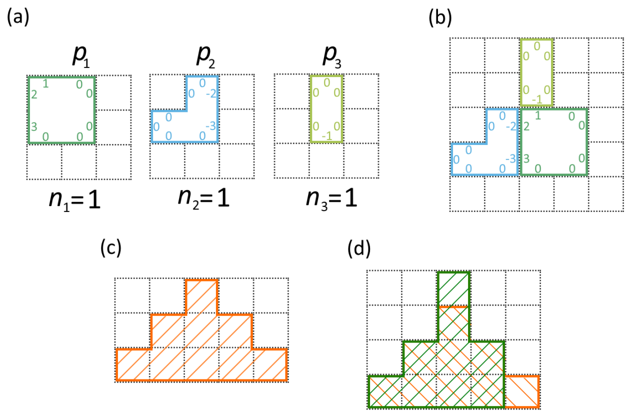

Figure 4 Example of loss value calculation. (a) Polyomino species P. (b) Cluster $\mathrm{A}(P)$ which P self-assembles into. (c) Target shape s. (d) A state that gives maximum overlap between the cluster and the shape.

The heavy-cost function is computationally heavy because it is necessary to estimate the formed clusters $\mathrm{A}(P)$ by the simulation. In order to define the heavy-cost function, we need to introduce a function to evaluate similarity between shapes.

The shape of a given cluster c is represented as a set of x, y coordinates in the cluster, $\mathrm{shape}(c) = \{\mathrm{pos}(u) \mid u \in p,\ p \in c\}$. The similarity $\mathrm{V_{ss}}$ between a cluster c and a shape s is defined as the number of square units that does not belong to the overlap, which is

$$\mathrm{V_{ss}}(c,\ s) = \sum_{\boldsymbol{x} \in \mathrm{shape}(c)} \mathrm{incl}(\boldsymbol{x},\ s) + \sum_{\boldsymbol{y} \in s} \mathrm{incl}(\boldsymbol{y},\ \mathrm{shape}(c)),$$

where

$$\mathrm{incl}(p,\ s) = \begin{cases} 0 & (p \in s) \\ -1 & (\text{otherwise}) \end{cases}.$$

The maximum volume of the similarity $\mathrm{V_{ss}^{max}}(c, s)$ is then defined by moving cluster c to have the maximum overlap, which means

$$\mathrm{V_{ss}^{max}}(c,\ s) = \max\Big(\bigcup_{n \in \mathbb{N}} \{\mathrm{V_{ss}}(c', s) | \forall m_1, m_2, \ldots m_n \in \mathrm{M_{agent}}, c' = m_1 \circ m_2 \circ \ldots \circ m_n(c)\}\Big).$$

Using the above definitions, the heavy-cost function is defined as

$$\mathrm{loss_{heavy}}(P, s) = |P|^2 + \frac{1}{2}|\mathrm{Gl}(P)| + \Big(\sum_{c \in \mathrm{A}(P)} \frac{\mathrm{V_{ss}^{max}}(c, s)}{|\mathrm{A}(P)|}\Big)^2.$$

As the result, a polyomino species P is evaluated as

$$\mathrm{loss}(P,\ s) = \begin{cases} \mathrm{loss_{heavy}}(P, s) & (\mathrm{loss_{light}}(P) < \alpha_{\mathrm{th}}) \\ |s|^2 + 3|s| + n_{\mathrm{cell}} \times m_{\mathrm{cell}} & (\text{otherwise}) \end{cases}.$$

3.4 Search of polyomino species with low loss value

Fig. 5 illustrates an example of initial and neighbor polyomino species generation. The initial polyomino species is generated by randomly decomposing the target shape into smaller polyominoes. This process is realized by repeating following two actions after generating a naive set of polyomino $S = \{p\}$ that has only one element p with the shape s.

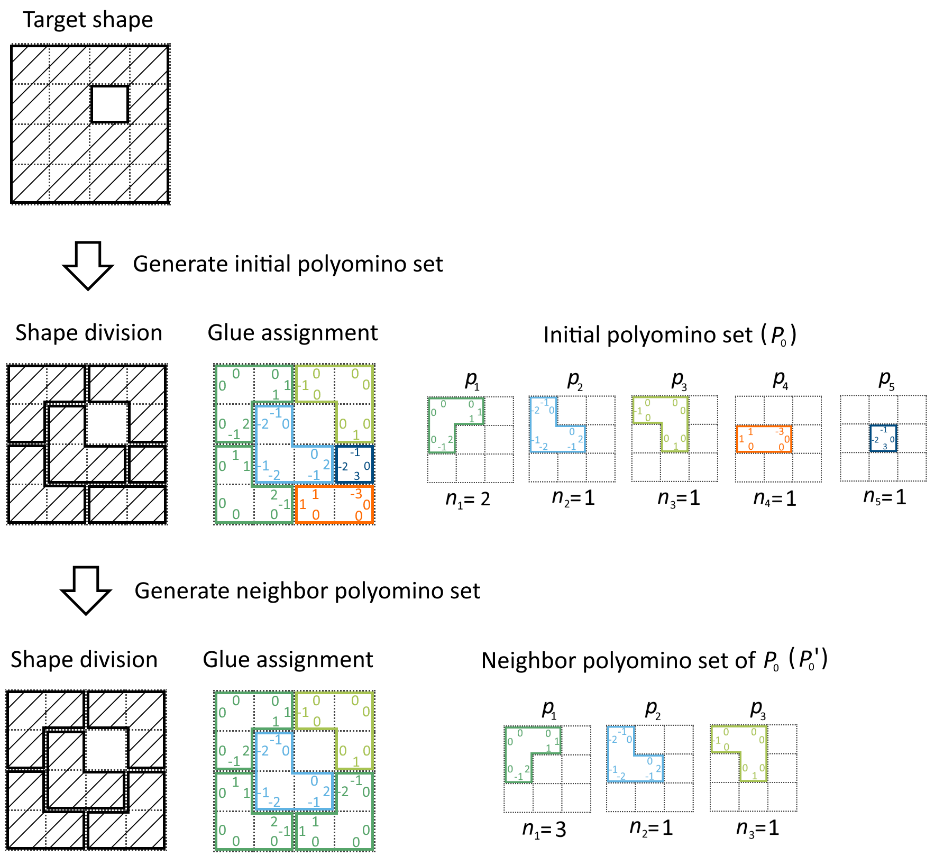
Target shape
Generate initial polyomino set
Shape division
Glue assignment
Initial polyomino set (P_0)
p_1 p_2 p_3 p_4 p_5
n_1= 2 n_2=1 n_3=1 n_4=1 n_5=1
Generate neighbor polyomino set
Shape division
Glue assignment
Neighbor polyomino set of P_0 (P_0')
p_1 p_2 p_3
n_1=3 n_2=1 n_3=1

Figure 5 Generation of initial polyomino species and neighbor polyomino species.

- Action1
  A square unit $u \in p$ is randomly selected. If randomly selected polyomino $p \in S$ satisfies "condition B", the algorithm removes the square unit u from the polyomino p and generate a new polyomino $\{u\}$. Here, condition B for a given polyomino p means that there are no other polyominoes with the same shape, or the size of p is smaller than or equal to $m_{\text{fix}} \times n_{\text{fix}}$. The S is updated to $S \backslash \{p\} \cup \{p \backslash \{u\}, \{u\}\}$. The probability to perform this action is r_{gen}.
- Action2
  A pair of neighboring polyominoes p and p' are randomly selected from S. If the polyominoes p and p' satisfy condition B, the algorithm removes a randomly selected square unit $u \in p$ from p that is adjacent to p', and add it to p'. The S is updated to $S \backslash \{p, p'\} \cup \{p \backslash \{u\}, p' \cup \{u\}\}$. The probability to perform this action is $1 - r_{\text{gen}}$.

If there are no polyominoes which meet condition B, one of these two actions takes place ignoring condition B. These two actions are repeated more than n_{new} steps, so that the sizes of all the polyominoes become smaller or equal to $m_{\text{poly}} \times n_{\text{poly}}$. Hereafter, we use $m_{\text{fix}} = 2$ and $n_{\text{fix}} = 1$ and $r_{\text{gen}} = 0.05$, $n_{\text{new}} = 100$.

Next, the algorithm assigns the glue types of polyominoes. Each glue type of polyominoes is set to all different value such that the square units have complementary glue types in contacting face with another polyomino. Glue types which are not contacting with any other polyomino are fixed to 0. Polyominoes with the equivalent shape are converted to equivalent polyominoes by assigning glue types properly (see Appendix A.2). From the set of polyominoes, corresponding polyomino species can be trivially constructed.

To make a neighbor polyomino species, one of the two decomposing actions is applied as a mutation and then new glue types are assigned.

When the current polyomino species is P and its new neighbor is P', the possibility to accept P' is

$$\mathrm{P}(P,\, P') = \begin{cases} 1 & (\mathrm{loss}(P,\, s) - \mathrm{loss}(P',\, s) \geq 0) \\ \exp((\mathrm{loss}(P,\, s) - \mathrm{loss}(P',\, s))\,/\,\tau_{\mathrm{sa}}) & (\text{otherwise}) \end{cases},$$

where τ_{sa} is a constant temperature parameter. Hereafter, we use $\tau_{\mathrm{sa}} = 10$, which empirically accepts 20% of transitions that increase the loss values. The total iteration n_{opt} is set to 100.

4 Result

To demonstrate the validity of proposed algorithm, we tested 3 target shapes with different complexities (small, medium, and large), where m_{poly} and n_{poly} are both set to 3. For each case, we run 100 searches for statistical analysis. The small target is given in 4×4 lattice (Fig. 6(a)). For this target, reasonably good polyomino species were always obtained such as the example in Fig. 6(b,c). The loss function development of 10 representative searches are shown in Fig. 6(d). The average loss value over 100 searches was 12.5 with a standard deviation of ± 0.37, which is smaller than 67.3 ± 14.6 of 100 random searches that find the best candidate from randomly generated 100 polyomino species.

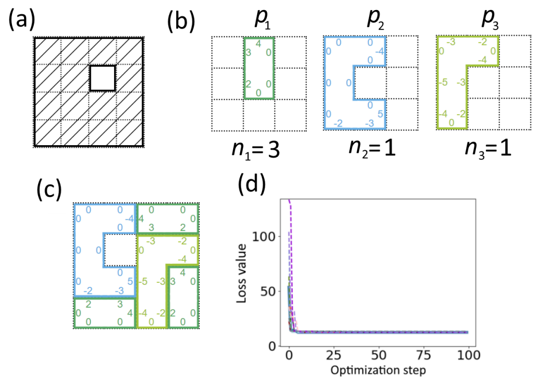

Figure 6 (a) Target shape. (b) An example of polyomino species P. (c) Cluster which P self-assembles into. (d) Development of loss function. The illustrated solution is shown in bold.

The medium target is given in a 5×5 lattice (Fig. 7(a)). A polyomino species that self-assembles into the target shape was also found as expected (Fig. 7(b,c)). The average loss value was 15.4 ± 2.7, which is significantly smaller than 134.2 ± 29.0 of random search. Ten representative results are shown in Fig. 7(d).

The large target is given in a 6×6 lattice (Fig. 8(a)). Some of the searches succeeded in finding a polyomino species that self-assembles into the target shape as in the example of Fig. 8(b,c). The polyominoes in the set, however, were all different and none of them were recycled in different places. The average loss value was 144.7 ± 87.7, which is smaller than 201.0 ± 121.6 of random search. Ten representative results are shown in Fig. 8(d).

The performance of the proposed algorithm is summarized in Fig. 9. In the small and medium cases, the loss values of proposed algorithm got significantly smaller than those of random search. In the large case, however, the difference between the proposed algorithm and random search was not as significant. This may due to insufficient iteration of the

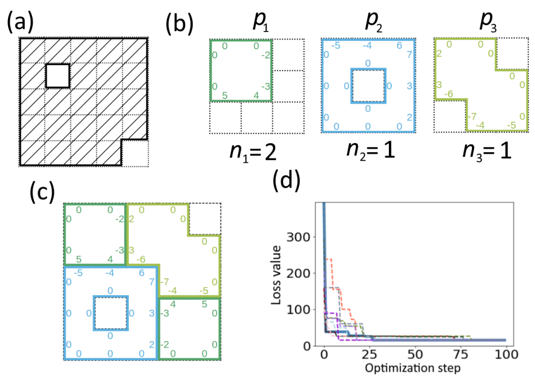

Figure 7 (a) Target shape. (b) An example of polyomino species P. (c) Cluster assembled by P. (d) Development of loss function. The illustrated solution is shown in bold.

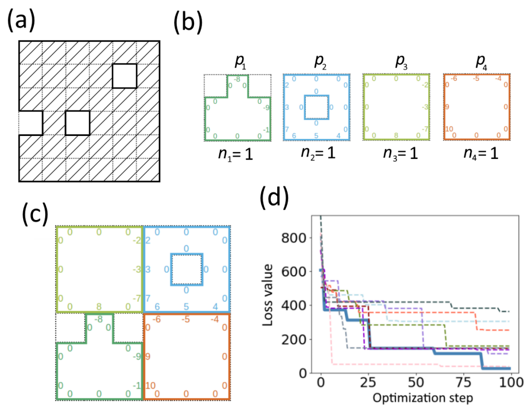

Figure 8 (a) Target shape. (b) An example of polyomino species P. (c) Cluster assembled by P. (d) Development of loss function. The illustrated solution is shown in bold.

search. We further quantified the convergence speed of the search using a logarithmic fit. The development of loss values in respect to the logarithmic optimization step with estimated slopes are shown in inset of Fig. 9(b). The number of iteration that is necessary to optimize the polyomino species using our strategy may grow exponentially to the size of the target.

5 Discussion

In this paper, we consider the problem of finding minimum set of polyominoes that assemble into a desired shape. A simulator developed on the agent-based Monte Carlo method evaluates the potential energy among the polyominoes and updates the simulation state to decrease the total potential. Since the geometrical interactions between polyominoes have to be taken into account, the developed simulator become complicated compared with the simulators for homogeneous units such as kTAM, where a set of polyominoes is represented as an agent, which can move, merge, and split during the simulation. With this framework, a self-assembly processes of polyominoes can be efficiently simulated.

In the proposed algorithm, meta-heuristic method called simulated annealing was adopted. Because of the enormous search space for the design problem, a two-step evaluation strategy was adopted to prune unpromising solution spaces. Automatic design for three example targets with different size and complexity was tested to show the feasibility of the proposed method.

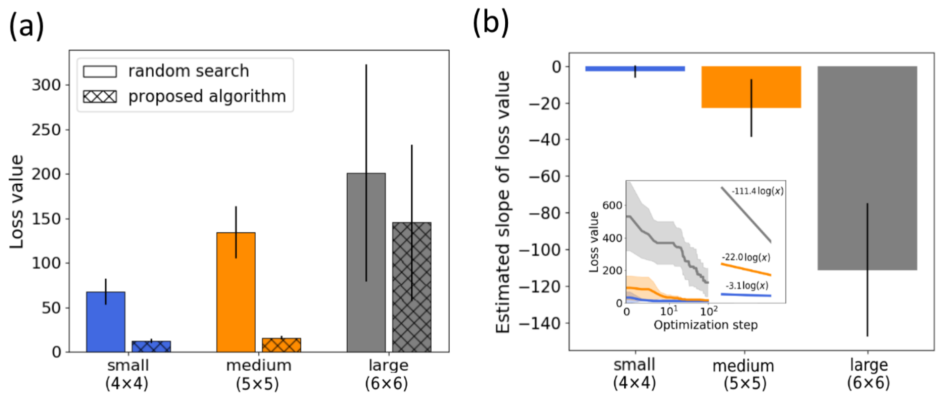

Figure 9 (a) The average of loss values at the last iteration of the searches of small (4×4), medium (5×5), and large (6×6) cases. The results of random search and proposed algorithms are compared. (b) Convergence speed of the proposed algorithm using a logarithmic fit. The inset shows the log-scale mean development of loss values, where standard deviation is illustrated as transparent area. The bars summarize the estimated slopes in the log-scale graph. The algorithms are run 100 times, and error bar indicates the standard deviation.

In order to solve a larger problem, we need to improve the efficiency of the algorithm, especially to reduce the computational cost of Monte Carlo simulation. For this purpose, it is necessary to redesign the potential energy between polyominoes to avoid kinetic traps. Introducing a new criterion to terminate the simulation at an appropriate step will be also effective. Larger-scale problems can be solved by introducing parallel computing hardware such as GPU along with the above improvements of the algorithms. From a computer science point of view, whether or not the automatic design problem of the polyomino set is NP is an interesting issue. Also, extending the problem to three-dimensional polycube is remained for a future work.

References

1 Mika Göös and Pekka Orponen. Synthesizing minimal tile sets for patterned dna self-assembly. In *International Workshop on DNA-Based Computers*, pages 71–82. Springer, 2010. doi:10.1007/978-3-642-18305-8_7.

2 Yu He, Yi Chen, Haipeng Liu, Alexander E Ribbe, and Chengde Mao. Self-assembly of hexagonal dna two-dimensional (2d) arrays. *Journal of the American Chemical Society*, 127(35):12202–12203, 2005. doi:10.1021/ja0541938.

3 Chenxiang Lin, Yan Liu, Sherri Rinker, and Hao Yan. Dna tile based self-assembly: building complex nanoarchitectures. *ChemPhysChem*, 7(8):1641–1647, 2006. doi:10.1002/cphc.200600260.

4 Xiaojun Ma and Fabrizio Lombardi. Synthesis of tile sets for dna self-assembly. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(5):963–967, 2008. doi:10.1109/tcad.2008.917973.

5 Sung Ha Park, Robert Barish, Hanying Li, John H Reif, Gleb Finkelstein, Hao Yan, and Thomas H LaBean. Three-helix bundle dna tiles self-assemble into 2d lattice or 1d templates for silver nanowires. *Nano letters*, 5(4):693–696, 2005. doi:10.1021/nl050108i.

6 Matthew J Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014. URL: https://link.springer.com/content/pdf/10.1007/s11047-013-9379-4.pdf.

7 Paul WK Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006. doi:10.1038/nature04586.

8 Alessandro Troisi, Vance Wong, and Mark A Ratner. An agent-based approach for modeling molecular self-organization. *Proceedings of the National Academy of Sciences*, 102(2):255–260, 2005. doi:10.1073/pnas.0408308102.

9 Erik Winfree. Simulations of computing by self-assembly. In *Fourth International Meeting on DNA-Based Computing*. California Institute of Technology, 1998. doi:10.7907/Z9TB14X7.

10 Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *International Workshop on DNA-Based Computers*, pages 126–144. Springer, 2003. doi:10.1007/978-3-540-24628-2_13.

11 Erik Winfree, Furong Liu, Lisa A Wenzler, and Nadrian C Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998. doi:10.1038/28998.

12 Hao Yan, Sung Ha Park, Gleb Finkelstein, John H Reif, and Thomas H LaBean. Dna-templated self-assembly of protein arrays and highly conductive nanowires. *science*, 301(5641):1882–1884, 2003. doi:10.1126/science.1089389.

A Appendix

A.1 Movements

Each movement of the polyomino p is defined as

$$\begin{aligned}
\mathrm{north}(p) &= \bigcup_{u\in p}\{(\mathrm{coord}(\mathrm{pos}(u),\mathrm{N}),\mathrm{gl}(u))\},\\
\mathrm{east}(p) &= \bigcup_{u\in p}\{(\mathrm{coord}(\mathrm{pos}(u),\mathrm{E}),\mathrm{gl}(u))\},\\
\mathrm{south}(p) &= \bigcup_{u\in p}\{(\mathrm{coord}(\mathrm{pos}(u),\mathrm{S}),\mathrm{gl}(u))\},\\
\mathrm{west}(p) &= \bigcup_{u\in p}\{(\mathrm{coord}(\mathrm{pos}(u),\mathrm{W}),\mathrm{gl}(u))\},\\
\mathrm{right}(p) &= \bigcup_{u\in p}\{(-(\mathrm{pos_y}(u)-\mathrm{c_y}(p))+\mathrm{c_x}(p),\mathrm{pos_x}(u)-\mathrm{c_x}(p)+\mathrm{c_y}(p),\mathrm{gl}(u)|_\mathrm{r})\},\\
\mathrm{back}(p) &= \bigcup_{u\in p}\{(-(\mathrm{pos_x}(u)-\mathrm{c_x}(p))+\mathrm{c_x}(p),(\mathrm{pos_y}(u)-\mathrm{c_y}(p))+\mathrm{c_y}(p),\mathrm{gl}(u)|_\mathrm{b})\},\\
\mathrm{left}(p) &= \bigcup_{u\in p}\{(\mathrm{pos_y}(u)-\mathrm{c_y}(p)+\mathrm{c_x}(p),-(\mathrm{pos_x}(u)-\mathrm{c_x}(p))+\mathrm{c_y}(p),\mathrm{gl}(u)|_\mathrm{l})\},
\end{aligned}$$

where $\mathrm{gl}(u)$ is the glue types g of $u=(x,y,\mathrm{g})$ and $\mathrm{g}|_\mathrm{r}$, $\mathrm{g}|_\mathrm{b}$, $\mathrm{g}|_\mathrm{l}$ are the glue types which can be obtained by rotating g. Namely,

$$\mathrm{g}|_\mathrm{r}(d)=\begin{cases}\mathrm{g}(\mathrm{W}) & (d=\mathrm{N})\\ \mathrm{g}(\mathrm{N}) & (d=\mathrm{E})\\ \mathrm{g}(\mathrm{E}) & (d=\mathrm{S})\\ \mathrm{g}(\mathrm{S}) & (d=\mathrm{W})\end{cases},$$

$$\mathrm{g}|_\mathrm{b}(d)=\begin{cases}\mathrm{g}(\mathrm{S}) & (d=\mathrm{N})\\ \mathrm{g}(\mathrm{W}) & (d=\mathrm{E})\\ \mathrm{g}(\mathrm{N}) & (d=\mathrm{S})\\ \mathrm{g}(\mathrm{E}) & (d=\mathrm{W})\end{cases},$$

$$\mathrm{g}|_{\mathrm{l}}(d) = \begin{cases} \mathrm{g}(\mathrm{E}) & (d = \mathrm{N}) \\ \mathrm{g}(\mathrm{S}) & (d = \mathrm{E}) \\ \mathrm{g}(\mathrm{W}) & (d = \mathrm{S}) \\ \mathrm{g}(\mathrm{N}) & (d = \mathrm{W}) \end{cases} .$$

Similarly, each movement of the agent a is defined as

$$\begin{aligned}
\mathrm{north}(a) &= \bigcup_{p\in a}\{\mathrm{north}(p)\},\\
\mathrm{east}(a) &= \bigcup_{p\in a}\{\mathrm{east}(p)\},\\
\mathrm{south}(a) &= \bigcup_{p\in a}\{\mathrm{south}(p)\},\\
\mathrm{west}(a) &= \bigcup_{p\in a}\{\mathrm{west}(p)\},\\
\mathrm{right}(a) &= \bigcup_{p\in a}\{\bigcup_{u\in p}\{(-(\mathrm{pos_y}(u) - \mathrm{c_y}(p)) + \mathrm{c_x}(a), \mathrm{pos_x}(u) - \mathrm{c_x}(a) + \mathrm{c_y}(a), \mathrm{gl}(u)|_{\mathrm{r}})\}\},\\
\mathrm{back}(a) &= \bigcup_{p\in a}\{\bigcup_{u\in p}\{(-(\mathrm{pos_x}(u) - \mathrm{c_x}(a)) + \mathrm{c_x}(a), -(\mathrm{pos_y}(u) - \mathrm{c_y}(a)) + \mathrm{c_y}(a), \mathrm{gl}(u)|_{\mathrm{b}})\}\},\\
\mathrm{left}(a) &= \bigcup_{p\in a}\{\bigcup_{u\in p}\{(\mathrm{pos_y}(u) - \mathrm{c_y}(a) + \mathrm{c_x}(a), -(\mathrm{pos_x}(u) - \mathrm{c_x}(a)) + \mathrm{c_y}(a), \mathrm{gl}(u)|_{\mathrm{l}})\}\},
\end{aligned}$$

where $\mathrm{c_x}(a)$ and $\mathrm{c_y}(a)$ are the center of mass of an agent a, which is defined as $\mathrm{c_x}(a) = \mathrm{round}(\sum_{u\in p, p\in a} \mathrm{pos_x}(u)/|a|)$ and $\mathrm{c_y}(a) = \mathrm{round}(\sum_{u\in p, p\in a} \mathrm{pos_y}(u)/|a|))$.

A.2 Glue type assignment

Given a naive set of polyominoes S, the assignment of glue types satisfies the conditions;

$$\begin{aligned}
&\forall p_1, p_2 \in S, \forall u_1 \in p_1, \forall u_2 \in p_2, \forall d \in \mathrm{D}, && \mathrm{gl}(u_1, d) = 0 \to \\
& && \mathrm{coord}(\mathrm{pos}(u_1), d) \neq \mathrm{pos}(u_2), \text{ and}\\
&\forall p_1, p_2 \in S, \forall u_1 \in p_1, \forall u_2 \in p_2, \forall d \in \mathrm{D}, && \mathrm{coord}(\mathrm{pos}(u), d) = \mathrm{pos}(u_2) \to \\
& && \mathrm{gl}(u_1, d)) + \mathrm{gl}(u_2, \hat{d}) = 0 \wedge \mathrm{gl}(u_1, d)) \neq 0.
\end{aligned}$$

The first condition means that the glue type is 0 when there are no neighboring square units. The second condition guarantees that the the neighboring units have complementary glue types. Finally, the number of representative polyominoes are decreased as much as possible by assigning glue types through an ad-hoc trial and error. The assignment of glue types is applied to construct the initial and neighbor polyomino species.