# Complexity of Graph Self-Assembly in Accretive Systems and Self-Destructible Systems*

John H. Reif † Sudheer Sahu † Peng Yin †

**Abstract**

Self-assembly is a process in which small objects autonomously associate with each other to form larger complexes. It is ubiquitous in biological constructions at the cellular and molecular scale and has also been identified by nanoscientists as a fundamental method for building molecular scale structures. Recent years see convergent interest and efforts in studying self-assembly from mathematicians, computer scientists, physicists, chemists, and biologists. However most complexity theoretic studies of self-assembly utilize mathematical models with two limitations: 1) only attraction, while no repulsion, is studied; 2) only assembled structures of two dimensional square grids are studied. These restrictions limit the practical impact of the resulting complexity theoretic results. In this paper, we study the complexity of the assemblies resulting from the cooperative effect of repulsion and attraction in a more general setting of graphs. This allows for the study of a more general class of self-assembled structures than the previous tiling model. We define two novel assembly models, namely the accretive graph assembly model and the self-destructible graph assembly model, and identify one fundamental problem in them: the sequential construction of a given graph, referred to as Accretive Graph Assembly Problem (AGAP) and Self-Destructible Graph Assembly Problem (DGAP), respectively. Our main results are: (i) AGAP is $\mathbf{NP}$-complete even if the maximum degree of the graph is restricted to 4 or the graph is restricted to be planar with maximum degree 5; (ii) counting the number of sequential assembly orderings that result in a target graph (#AGAP) is $\#\mathbf{P}$-complete; and (iii) DGAP is $\mathbf{PSPACE}$-complete even if the maximum degree of the graph is restricted to 6 (this is the first $\mathbf{PSPACE}$-complete result in self-assembly). We also extend the accretive graph assembly model to a stochastic model, and prove that determining the probability of a given assembly in this model is $\#\mathbf{P}$-complete.

## 1 Introduction

Self-assembly is the ubiquitous process in which small objects associate autonomously with each other to form larger complexes. For example, atoms can self-assemble into molecules; molecules into crystals; cells into tissues, *etc*. Recently, self-assembly has also been explored as a powerful and efficient mechanism for constructing synthetic molecular scale objects with nano-scale features. This approach is particularly fruitful in DNA based nanoscience, as exemplified by the diverse set of DNA lattices made from self-assembled branched DNA molecules (DNA tiles) [2, 19, 23, 24, 39, 41, 42]. Another nanoscale example is the self-assembly of peptide molecules [10]. Self-assembly is also used for larger scale construction, for example, via the use of capillary forces [9, 29] or magnetic forces [1] to provide attraction and repulsion between meso-scale tiles and other objects.

Building on classical Wang tiling models [28, 37] dating back to 1960s, Rothemund and Winfree [31] in 2000 proposed an elegant discrete mathematical model for complexity theoretic studies of self-assembly known as the *Tile Assembly Model*. In this model, DNA tiles are treated as oriented unit squares (*tiles*). Each of the four sides of a tile has a glue with a positive integral strength. Assembly occurs by accretion of tiles

*The work is supported by NSF ITR Grants EIA-0086015 and CCR-0326157, NSF QuBIC Grants EIA-0218376 and EIA-0218359, and DARPA/AFSOR Contract F30602-01-2-0561.

†Department of Computer Science, Duke University, Durham, NC, USA. {reif, sudheer, py}@cs.duke.edu

iteratively to an existing assembly, starting with a distinguished *seed* tile. A tile can be "glued" to a position in an existing assembly if the tile can fit in the position such that each pair of abutting sides of the tile and the assembly have the same glue and the total strength of the glues is greater than or equal to the *temperature*, a system parameter. Research in this field largely focuses on studying the complexity of and algorithms for (uniquely and terminally) producing assemblies with given properties, such as shape. It has been shown that the construction of $n \times n$ squares has a program size complexity (the minimum number of distinct types of tiles required) of $\Theta(\frac{\log n}{\log \log n})$ [4, 31]. The upper bound is obtained by simulating a binary counter and the lower bound by analyzing the Kolmogorov complexity of the tiling system [21]. The model was later extended by Adleman *et al.* to include the time complexity of generating specified assemblies [4]. Later work studies various topics, including combinatorial optimization, complexity problems, fault tolerance, and topology changes, in the standard Tile Assembly Model as well as some of its variants [4, 5, 7, 8, 11, 12, 13, 15, 16, 17, 14, 20, 27, 30, 38, 33, 14, 35, 34].

Though substantial progress has been made in recent years in the study of self-assembly using the above tile assembly model, which captures many important aspects of self-assembly in nature and in nanofabrications, the complexity of some other important aspects of self-assembly remains unexplored:

- Only attraction, while no repulsion, is studied. However, repulsive forces often occur in self-assembly. For example, there is repulsion between hydrophobic and hydrophilic tiles [9, 29]; between tiles labeled with magnetic pads of the same polarity [1]; and there is also static electric repulsion in molecular systems, *etc.*. Indeed, the study of repulsive forces (negative edge weight) in the self-assembly system was posed as an open question by Adleman and colleagues in [4]. Though there has been previous work on the kinetics of such systems, e.g. Klavin's "waterbug" model [18], no complexity theoretic study has been directed towards such systems.
- Generally only assembled structures of two dimensional square grids are studied. In contrast, many molecular self-assemblies using DNA and other materials involve the assembly of more diverse structures in both two and three dimensions. For example, Seeman's group constructed self-assembled nonregular graphs using DNA junction molecules as vertices and duplex DNA molecules as edges [32].

In this paper, we study the cooperative effect of repulsion and attraction in a graph setting. This approach allows the study of a more general class of assemblies as described above.

We distinguish two systems, namely the *accretive system* and the *self-destructible system*. In an accretive system, an assembled component cannot be removed from the assembly. In contrast, in the self-destructible system, a previously assembled component can be "actively" removed from the assembly by the repulsive force exerted by another newly assembled component. In other words, the assembly can (partially) *destruct* itself. We define the *accretive graph assembly model* for the former and the *self-destructible graph assembly model* for the latter.

We first define an accretive assembly model and study a fundamental problem in this model: the sequential construction of a given graph, referred to as Accretive Graph Assembly Problem (AGAP). Our main result for this model is that AGAP is $\mathbf{NP}$-complete even if the maximum degree of vertices in the graph is restricted to 4; the problem remains $\mathbf{NP}$-complete even for planar graphs (planar AGAPor PAGAP) with maximum degree 5. We also prove that the problem of counting the number of sequential assembly orderings that lead to a target graph (#AGAP) is $\#\mathbf{P}$-complete. We further extend the AGAP model to a stochastic model, and prove that determining the probability of a given assembly in this model (stochastic AGAP or SAGAP) is $\#\mathbf{P}$-complete.

If we relax the assumption that an assembled component always stays in the assembly, repulsive force between assembled components can cause self-destruction in the assembly. Self-destruction is a common phenomenon in nature, at least in biological systems. One renowned example is apoptosis, or programmed cell death [36]. Programmed cell death can be viewed as a self-destructive behavior exercised by a multicellular organism, in which the organism actively kills a subset of its constituent cells to ensure the normal development and function of the whole system. It has been shown that abnormalities in programmed cell

death regulation can cause a diverse range of diseases such as cancer and autoimmunity [36]. It is also conceivable that self-destruction can be exploited in self-assembly based nano-fabrication: the components that serve to generate intermediate products but are unnecessary or undesirable in the final product should be actively removed. We provide an illustrative abstract example in Figure 5 and Figure 6 in Section 5.

To the best of our knowledge, our self-destructible graph assembly model is the first complexity theoretic model that captures and studies the fundamental phenomenon of self-destruction in self-assembly systems. Our model is different from previous work on reversible tiling systems [3, 6, 40]. These previous studies use elegant thermodynamic or stochastic techniques to investigate the reversible process of tile assembly/disassembly: an assembled tile has a probability of "falling" off the assembly in a kinetic system. In contrast, our self-destructible system models the behavior of a self-assembly system that "actively" destructs parts of itself.

To model the self-destructible systems, we define a self-destructible graph assembly model, and consider the problem of sequentially constructing a given graph, referred to as the Self-Destructible Graph Assembly Problem (DGAP). We prove that DGAP is $\mathbf{PSPACE}$-complete even if the graph is restricted to have maximum degree 6.

The rest of the paper is organized as follows. We first define the accretive graph assembly model and the AGAP problem in Section 2. In this model, we first show the $\mathbf{NP}$-completeness of AGAP and PAGAP(planar AGAP) in Section 3 and then show the $\#\mathbf{P}$-completeness of SAGAP (stochastic AGAP) in Section 4. Next, we define the self-destructible graph assembly model and the DGAP problem in Section 5 and show the $\mathbf{PSPACE}$-completeness of DGAP in Section 6. We close with a discussion of our results in Section 7.

## 2 Accretive Graph Assembly Model

Let $\mathbb{N}$ and $\mathbb{Z}$ denote the set of natural numbers and the set of integers, respectively. A *graph assembly system* is a quadruple $\mathcal{T} = \langle G = (V, E), v_s, w, \tau \rangle$, where $G = (V, E)$ is a given graph with vertex set $V$ and edge set $E$, $v_s \in V$ is a distinguished *seed vertex*, $w : E \to \mathbb{Z}$ is a weight function (corresponding to the glue function in the standard tile assembly model [31]), and $\tau \in \mathbb{N}$ is the *temperature* of the system (intuitively temperature provides a tunable parameter to control the stability of the assembled structure). In contrast to the canonical tile assembly model in [31], which allows only positive edge weight, we allow both positive and negative edge weights, with positive (resp. negative) edge weight modeling the attraction (resp. repulsion) between the two vertices connected by this edge. We will see that this simple extension makes the assembly problem significantly more complex.

Roughly speaking, given a graph assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$, $G$ is *sequentially constructible* if we can attach all its vertices one by one, starting with the seed vertex; a vertex $x$ can be assembled if the *support* to it is equal to or greater than the system temperature $\tau$, where support is the sum of the weights of the edges between $x$ and its assembled neighbors.

Figure 1 gives an example. Here the graph is shown in Figure 1 (a) and the temperature is set to 2. Figure 1 (b) gives a step-by-step illustration of the assembly sequence. Note that if $h$ gets assembled before $e$, then the whole graph can get assembled: an example assembly ordering can be $a \prec b \prec c \prec d \prec f \prec g \prec h \prec i \prec e$. In contrast, if vertex $e$ gets assembled before $h$, the graph cannot be assembled: $h$ can be assembled only if it gets support from *both* $g$ and $i$; while $i$ cannot get assembled without the support from $h$.

Formally, given a graph assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$, $G$ is *sequentially constructible* if there exists an ordering of *all* the vertices in $V$, $\mathcal{O}_{\mathcal{T}} = (v_s = v_0 \prec v_1 \prec v_2 \prec \cdots \prec v_{n-1})$ such that $\sum_{v_j \in N_G(v_i), j<i} w(v_i, v_j) \geq \tau, 0 < i \leq n-1$, where $N_G(v_i)$ denotes the set of vertices adjacent to $v_i$ in $G$. The ordering $\mathcal{O}_{\mathcal{T}}$ is called an *assembly ordering* for $G$. $\sigma_{\mathcal{O}}(v_i) = \sum_{v_j \in N_G(v_i), j<i} w(v_i, v_j)$ is called the *support* of $v_i$ in ordering $\mathcal{O}$. When the context is clear, we simply use $\mathcal{O}$ and $\sigma(v_i)$ to denote assembly ordering and support, respectively.

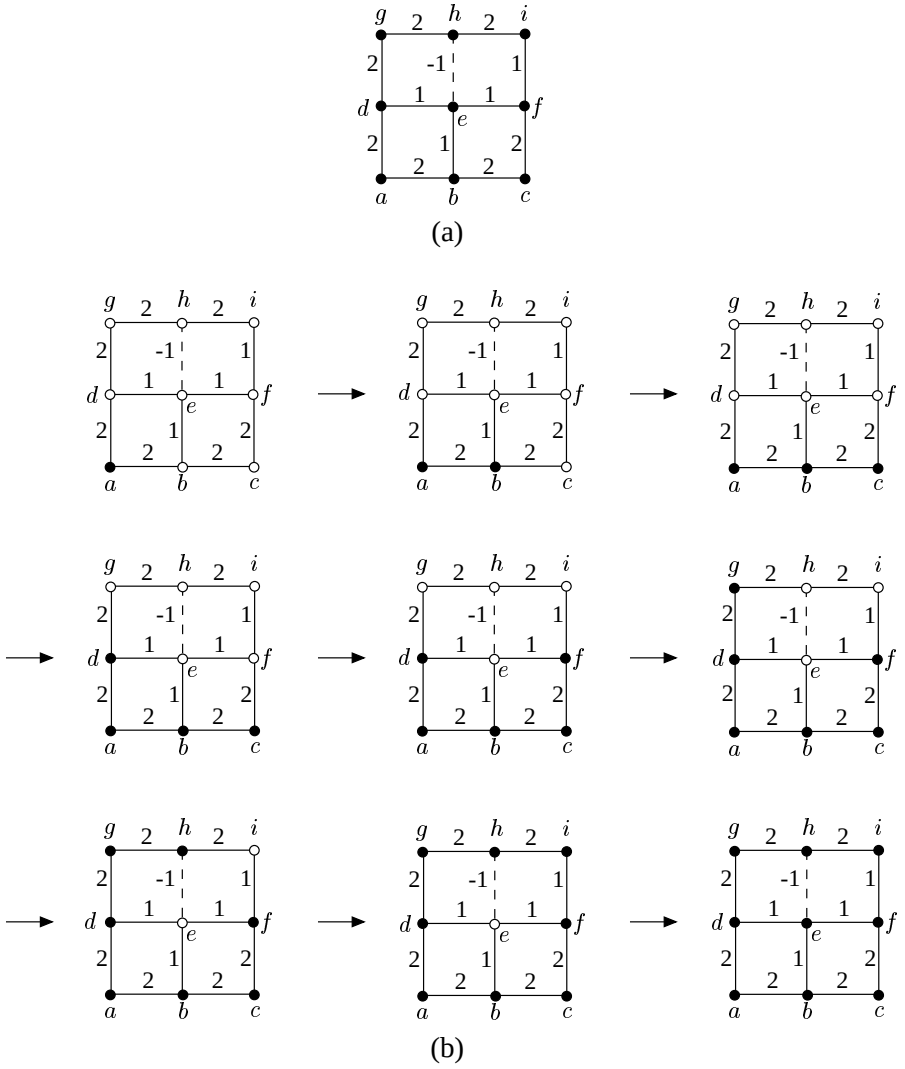


Figure 1: (a) An example of graph assembly in the accretive model. (b) A step-by-step illustration of the example assembly sequence.

We define the *accretive graph assembly problem* as follows,

**Definition 2.1 Accretive Graph Assembly Problem (AGAP)***: Given a graph assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$ in the accretive model, determine whether there exists an assembly ordering $\mathcal{O}$ for $G$.*

The above model is *accretive* in the sense that once a vertex is assembled, it cannot be 'popped off' by the subsequent assembly of any other vertex. If we relax this assumption, we will obtain a self-destructible model, which is described later in Section 5.

# 3 AGAP and PAGAP are NP-complete

## 3.1 4-DEGREE AGAP is NP-complete

**Lemma 3.1** AGAP *is in* **NP**.

**Proof:** Given an assembly ordering of the vertices, sequentially check whether each vertex can be assembled (with sufficient support). This takes polynomial time. □

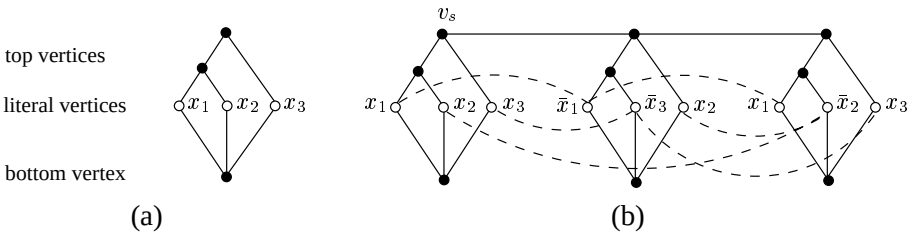


Figure 2: (a) A clause gadget. The top vertices and the bottom vertex are colored black; the literal vertices are white. (b) A graph construction corresponding to an AGAP reduction from 3SAT formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$. An edge between two literal vertices is depicted as a dashed arch and assigned weight -1; all other edges have weight 2.

Recall that the **NP**-complete 3SAT problem asks: Given a Boolean formula $\phi$ in conjunctive normal form with each clause containing 3 literals, determine whether $\phi$ is satisfiable [26]. Also recall that 3SAT remains **NP**-complete for formulas in which each variable appears at most three times, and each literal at most twice [26]. We will reduce this restricted 3SAT to AGAP to prove AGAP is **NP**-hard.

**Lemma 3.2** AGAP *is* **NP***-hard.*

**Proof:** Given a 3SAT formula $\phi$ where each variable appears at most three times, and each literal at most twice, we will construct below an accretive graph assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$ for $\phi$. We will then show that the satisfiability problem of $\phi$ can be reduced (in logarithmic space) to the sequential constructibility problem of $G$ in $\mathcal{T}$.

For each clause in $\phi$, construct a *clause gadget* as in Figure 2 (a). For each literal, we construct a *literal vertex* (colored white in Figure 2 (a)). We further add *top vertices* (black) above and *bottom vertices* (black) below the literal vertices as in Figure 2 (a). We next take care of the structure of formula $\phi$ as follows. Connect all the clause gadgets sequentially via their top vertices as in Figure 2 (b); connect two literal vertices if and only if they correspond to two complement literals. This produces graph $G$. Designate the leftmost top vertex as the seed vertex $v_s$. We next assign weight $-1$ to an edge between two literal vertices and weight 2 to all the other edges. Finally, set the temperature $\tau = 2$. This completes the construction of $\mathcal{T} = \langle G, v_s, w, \tau \rangle$. For a concrete example, see Figure 2 (b).

The following proposition implies the lemma.

**Proposition 3.3** *There is an assembly ordering $\mathcal{O}$ for $\mathcal{T}$ if and only if $\phi$ is satisfiable.*

$\Rightarrow$

First we show that if $\phi$ can be satisfied by truth assignment $T$, then we can derive an assembly ordering $\mathcal{O}$ based on $T$.

**Stage 1.** Starting from the seed vertex, assemble all the top vertices sequentially. This can be easily done since each top vertex will have support 2, which is greater than or equal to $\tau = 2$, the temperature.

**Stage 2.** Assemble all the literal vertices assigned $true$. Since two $true$ literals cannot be complement literals, no two literal vertices to be assembled at this stage can have a negative edge between them. Hence all these $true$ literal vertices will receive a support 2 ($\geq \tau = 2$).

**Stage 3.** Assemble all the bottom vertices. Note that truth assignment $T$ satisfies $\phi$ implies that every clause in $\phi$ has at least one $true$ literal. Thus every clause gadget in $G$ has at least one literal vertex (a $true$ literal vertex) assembled in stage 2, which in turn allows us to assemble the bottom vertex in that clause gadget.

**Stage 4.** Assemble all the remaining literal vertices (the $false$ literal vertices). Observe that any remaining literal vertex $v$ has support 4 from its already assembled neighboring top vertex and bottom vertex and that $v$ can have negative support at most $-2$ from its assembled literal vertex neighbors (recall that each literal vertex can have at most two literal vertex neighbors since each variable appears at most three times in $\phi$). Hence the total support for $v$ will be at least 2 ($\geq \tau$).

$\Leftarrow$

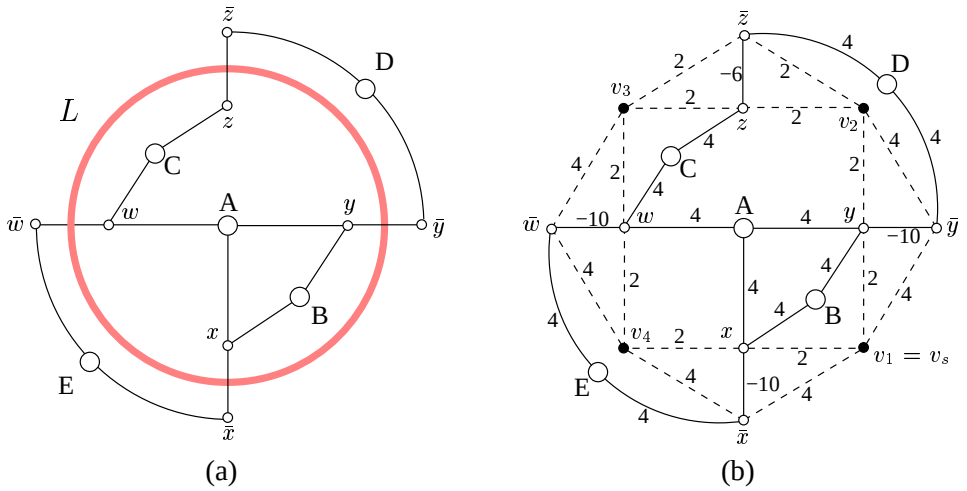


Figure 3: (a) and (b) are respectively an identifying graph and a PAGAP graph construction corresponding to the P3SAT formula $A \wedge B \wedge C \wedge D \wedge E = (x \vee y \vee w) \wedge (x \vee y) \wedge (w \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{w} \vee \bar{x})$. The larger (smaller) white circles represent clauses (literals); black vertices in (b) represent assisting vertices. Note that each clause is adjacent to at most three literals; each literal is adjacent to at most two clauses. The grey loop in (a) is loop $L$; integers in (b) indicate edge weights.

Suppose that there exists an assembly ordering $\mathcal{O}$, then we can derive a satisfying truth assignment $T$ for $\phi$. For each literal vertex, assign its corresponding literal $true$ if it appears in $\mathcal{O}$ before *all* of its literal vertex neighbors (this assures no two complement literals are both assigned $true$); otherwise assign it $false$.

To show that $T$ satisfies $\phi$, we only need to show every clause contains at least one $true$ literal. For contradiction, suppose there exists a clause gadget $A$ with three $false$ literal vertices, where $v$ is the literal vertex assembled first. However, $v$ cannot be assembled: it has support 2 from the top vertex; no support from the bottom vertex ($v$ gets assembled first and hence the bottom vertex in $A$ cannot be assembled before $v$); at least $-1$ negative support from one of its literal vertex neighbors ($v$ is assigned $false$); the total support of $v$ is thus at most 1, less than temperature $\tau = 2$. Contradiction. Hence $T$ must satisfy $\phi$. □

The following theorem follows immediately from Lemma 3.1 and Lemma 3.2.

**Theorem 3.4** AGAP *is* **NP***-complete.*

Let $k$-DEGREE AGAP be the AGAP in which the largest degree of any vertex in graph $G$ is $k$. Observe that the largest degree of any vertex in the graph construction in the proof of Lemma 3.2 is 4. Hence we have

**Corollary 3.5** *4-*DEGREE AGAP *is* **NP***-complete.*

## 3.2 5-DEGREE PAGAP is NP-complete

We next study the planar AGAP (PAGAP) problem, where the graph $G$ in the assembly system $\mathcal{T}$ is planar. First, note that the following lemma is trivially true.

**Lemma 3.6** PAGAP *is in* **NP**.

We show that PAGAP is **NP**-hard by a reduction from the **NP**-hard planar three-satisfiability problem (P3SAT) [22], defined in the following way. Given a 3SAT formula $\phi$, construct its *identifying graph* $G = (V, E)$ as follows: the vertex set $V$ is $\{l |\ l \text{ is a variable }\} \bigcup \{c |\ c \text{ is a clause }\}$; the edge set $E$ is $\{(l,\ c) |\ l \text{ is a variable in clause } c\}$. If $G$ is planar, $\phi$ is referred to as a *planar* 3SAT *(*P3SAT*) formula*. P3SAT problem is to decide the satisfiability of a P3SAT formula $\phi$.

We use the identifying graph construction in [25], which represents each variable $x$ with two vertices (one for $x$ and one for $\bar{x}$) connected by an edge. See Figure 3 (a) for an example. We use the following two properties of this construction in our proof: 1) There exists a loop $L$ that passes between all pairs of

literals without intersecting any edge between a literal and a clause; 2) Any literal can belong to at most two clauses [25].

**Lemma 3.7** PAGAP *is* **NP**-*hard.*

**Proof:** Given an arbitrary P3SAT formula $\phi$, we first construct an assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$. We then show that the satisfiability problem of $\phi$ can be reduced (in logarithmic space) to the sequential constructibility problem of $G$ in $\mathcal{T}$.

We construct a graph $G = (V, E)$ by modifying the identifying graph of $\phi$: along the loop $L$, add an assisting vertex $v_i$ between every two consecutive pairs of literal vertices and connect $v_i$ with all these four vertices as shown in Figure 3 (b). Next, we assign edge weights. The weight of an edge between a literal and a clause is $4$; the weight of an edge between a literal $x$ and its complement $\bar{x}$ is $-6$ if neither of them is connected to more than one clause; it is $-10$ if at least one of the literals is connected to two clause vertices. The weight of an edge connecting an assisting vertex and a literal vertex $x$ is $4$ if the weight of edge $(x, \bar{x})$ is $-10$ and $x$ is connected to only one clause vertex; otherwise it is $2$. Finally, we select an arbitrary assisting vertex, say $v_1$, as the seed vertex $v_s$ and set the temperature $\tau = 2$. This completes the construction of $\mathcal{T}$.

We next prove the following proposition, which completes the proof of the lemma.

**Proposition 3.8** *If and only if $\phi$ is satisfiable, there is an assembly ordering $\mathcal{O}$.*

$\Rightarrow$

Suppose there exists a truth assignment $T$ that satisfies $\phi$, we give the following assembly ordering.

**Stage 1.** Assemble all the assisting vertices and $true$ literals as follows. Starting from the seed vertex, following the clockwise direction along loop $L$, we assemble alternately $true$ literals (one of $x$ and $\bar{x}$ is necessarily $true$) and assisting vertices, till we reach the seed vertex again. For example, a satisfying truth assignment $(x, y, z, w) = (true, true, true, false)$ in Figure 3 (b) will give the assembly ordering $v_s = v_1 \prec y \prec v_2 \prec z \prec v_3 \prec \bar{w} \prec v_4 \prec x$.

**Stage 2.** Assemble all the clauses. Since $T$ satisfies $\phi$, each clause contains at least one $true$ literal and hence is now connected to at least one $true$ literal vertex assembled in stage 1. Thus all the clause vertices can be assembled now.

**Stage 3.** Assemble all the $false$ literals and thus complete the whole graph. Since all the neighbors of each $false$ literal have already been assembled, it is easy to verify that there is enough support for it.

$\Leftarrow$

Suppose that there exists an assembly ordering $\mathcal{O}$, we derive from $\mathcal{O}$ a truth assignment $T$ by assigning a literal vertex $x$ $true$ if it appears before $\bar{x}$ in $\mathcal{O}$; assign it $false$ otherwise. We claim that $T$ satisfies $\phi$.

For contradiction, assume there is a clause, say $A$, unsatisfied, with all its literals $x$, $y$, and $z$ assigned $false$. This implies that $\bar{x}$ (resp. $\bar{y}$, $\bar{z}$) appears before $x$ (resp. $y$, $z$) in $\mathcal{O}$. Assume w.l.o.g. that $x \prec y \prec z$ in $\mathcal{O}$. Since $A$ is adjacent to *only* $x$, $y$, and $z$, vertex $x$ must appear before $A$ in $\mathcal{O}$. However, by the edge weight assignment, if $x$ appears after its complement $\bar{x}$, then it can be assembled only after *all* the clause vertices connected to $x$ are assembled. In particular, we must have clause $A$ appears before $x$. Contradiction. We thus conclude that $T$ must satisfy $\phi$. □

Putting together Lemma 3.6 and Lemma 3.7, we have

**Theorem 3.9** PAGAP *is* **NP**-*complete.*

**Corollary 3.10** 5-DEGREE PAGAP *is* **NP**-*complete.*

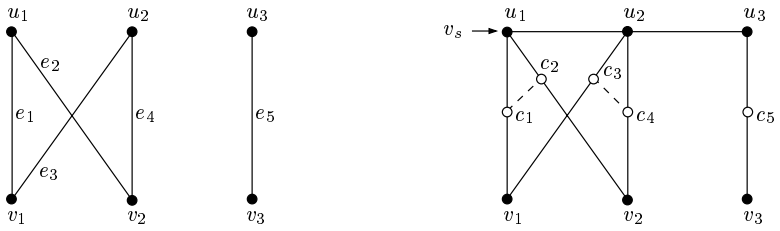


Figure 4: (a) and (b) show an example bipartite graph $B$ and the corresponding graph $G$ used in the proof of Lemma 4.2, respectively. In (b), $c_i$'s denote connector vertices (colored white); $u_1$ is the seed vertex. The weight of an edge connecting two connector vertices (dashed line) is $-4$; the weight of any other edge is 2.

# 4 #AGAP and SAGAP are #P-complete

## 4.1 #AGAP is #P-complete

We now consider a more general version of AGAP: given an accretive graph assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$ and a *target vertex set* $V_t \subseteq V$, determine if there exists an ordering $\tilde{\mathcal{O}}(V, V_t)$ of $V$ such that $V_t$ is assembled after we *attempt* assembling each vertex $v \in V$ sequentially according to $\tilde{\mathcal{O}}$. Vertex $v$ will be assembled if there is enough support; otherwise it will not be. $\tilde{\mathcal{O}}$ is called the assembly ordering of $V$ for $V_t$. When the context is clear, we simply call it assembly ordering for $V_t$ and denote it by $\tilde{\mathcal{O}}$. Note that the assembly ordering $\tilde{\mathcal{O}}$ is an ordering on all the vertices in $V$, but we only care about the assembly of the target vertex set $V_t$: the assembly of vertices in $V \setminus V_t$ is neither required nor prohibited. For $V_t = V$, the general AGAP is then precisely the standard AGAP we have been studying. The problem of counting the number of assembly orderings for $V_t \subseteq V$ under this general AGAP model is referred to as #AGAP.

**Lemma 4.1** #AGAP *is in* $\#\mathbf{P}$.

We next show #AGAP is $\#\mathbf{P}$-hard, using a reduction from the $\#\mathbf{P}$-complete problem PERMANENT, the problem of counting the number of perfect matchings in a bipartite graph [26].

**Lemma 4.2** #AGAP *is* $\#\mathbf{P}$*-hard.*

**Proof:** Given a bipartite graph $B = (U, V, E)$ with two partitions of vertices $U$ and $V$ and edge set $E$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, and $E = \{e_1, \ldots, e_m\}$ (recall that by definition of bipartite graph, there is no edge between any two vertices in $U$ and no edge between any two vertices in $V$), we construct an assembly system $\mathcal{T} = \langle G, v_s, w, \tau \rangle$. First, we derive graph $G$ by adding vertices and edges to $B$ (see Figure 4 for an example): on each edge $e_k$ add a splitting *connector vertex* $c_k$; add an edge (dashed line) between two connector vertices if they share a same neighbor in $U$; connect $u_i$ and $u_{i+1}$ for $i = 1, \ldots, n-1$. Next, assign weight $-4$ to an edge between two connector vertices; assign weight 2 to all the other edges. Finally, designate $u_1$ as the seed vertex $v_s$, and set the temperature $\tau = 2$. The target vertex set $V_t$ is $U \bigcup V$.

A crucial property of $G$ is that the assembly of one connector vertex $c$ will make all of $c$'s connector vertex neighbors unassemblable, due to the negative edge connecting $c$ and its neighbors. Consequently, starting from a vertex $u \in U$, only one connector vertex and hence only one $v \in V$ can be assembled. For a concrete example, see Figure 4 (b): starting from $u_1$, if we sequentially assemble $c_1$ and $v_1$, vertex $c_1$ will render $c_2$ unassemblable, and hence the assembly sequence $u_1 \prec c_2 \prec v_2$ is not permissible.

We first show that if there is no perfect matching in $B$, there is no assembly ordering for $U \bigcup V$. If there is no perfect matching in $B$, then there exists $S \subseteq V$ s.t. $|N(S)| < |S|$ (Hall's theorem), where $N(S) \subseteq U$ is the set of neighboring vertices to the vertices in $S$ in original graph $B$. However, as argued above, one vertex in $U$ can lead to the assembly of at most one vertex in $V$. Thus $|N(S)| < |S|$ implies that at least one vertex in $S$ remains unassembled. Hence, no assembly ordering exists that can assemble all vertices in $U \bigcup V$.

Next, when there exists perfect matching(s) in $B$, we can show that each perfect matching in $B$ corresponds to a ***fixed*** number of assembly orderings for $U \bigcup V$. First note that the total number of vertices in

graph $G$ is $2n + m$ (recall that $m$ is the number of edges in $B$ and hence number of connector vertices in $G$), giving a total $s = (2n + m)!$ permutations. We divide $s$ by the following factors to get the number of assembly orderings for $U \bigcup V$.

1. For every matching edge $e_k$ between $u \in U$ and $v \in V$, we have to follow the strict order $u \prec c_k \prec v$, where $c_k$ is the connector vertex on $e_k$. This is ensured by our construction as argued above. There are altogether $n$ such matching edges. So we need to further divide $s$ by $(3!)^n$.

2. For the $n$ vertices in $U$, we have to follow the strict order of assembling the vertices from left to right, and hence we need to divide $s$ by $n!$.

3. Denote by $d_i$ the degree of $u_i$ in graph $B$. For the $d_i$ connector vertices corresponding to the $d_i$ edges incident on $u_i$, the connector vertex corresponding to the matching edge must be assembled first, and thus, we need to further divide $s$ by $\Pi_{i=1}^{n} d_i$.

Putting together 1), 2), and 3), we have that each perfect matching in $B$ corresponds to $\frac{(2n+m)!}{(3!)^n (n!)(\Pi_{i=1}^{n} d_i)}$ assembly orderings for $U \bigcup V$ in $G$. This completes the proof. □

Lemma 4.1 and Lemma 4.2 imply

**Theorem 4.3** #AGAP *is* $\#\mathbf{P}$*-complete.*

### 4.2 SAGAP is $\#\mathbf{P}$-complete

An intimately related question to counting the total number of assembly orderings is the problem to calculate the probability of assembling a target structure in a stochastic setting. We extend the accretive graph self-assembly model to stochastic accretive graph self-assembly model as follows. Given a graph $G = (V, E)$, where $|V| = n$, starting with the seed vertex $v_s$, what is the probability that the target vertex set $V_t \subseteq V$ gets assembled if anytime any unassembled vertex can be picked with equal probability? This problem is referred to as stochastic AGAP (SAGAP).

Since any unassembled vertex has equal probability of being selected and the assembly has to start with the seed vertex, the total number of possible orderings are $(n - 1)!$. Then SAGAP asks precisely how many of these $(n - 1)!$ orderings are assembly orderings for the target vertex set $V_t$. Thus, #AGAP can be trivially reduced to SAGAP, and the reduction is obviously a logarithmic space parsimonious reduction. We immediately have

**Theorem 4.4** SAGAP *is* $\#\mathbf{P}$*-complete.*

## 5 Self-Destructible Graph Assembly Model

The assumption in the above accretive model is that once a vertex is assembled, it cannot be "popped off" by the subsequent assembly of another vertex. Next, we relax this assumption and obtain a more general model: the *self-destructible graph assembly model*. In this model, the incorporation of a vertex $a$ that exerts repulsive force on an already assembled vertex $b$ can make $b$ unstable and hence "pop" $b$ off the assembly. This phenomenon renders the assembly system an interesting dynamic property, namely (partial) self-destruction.

The *self-destructible graph assembly system* operates on a *slot graph*. A slot graph $\tilde{G} = (S, E)$ is a set of "slots" $S$ connected by edges $E \subseteq S \times S$. Each "slot" $s \in S$ is associated with a set of vertices $V(s)$. During the assembly process, a slot $s$ is either empty or is occupied by a vertex $v \in V(s)$. A slot $s$ occupied by a vertex $v$ is denoted as $\langle s, v \rangle$.

A *self-destructible graph assembly system* is defined as $\mathcal{T} = \langle \tilde{G} = (S, E), V, M, w, \langle s_s, v_s \rangle, \tau \rangle$, where $\tilde{G} = (S, E)$ is a given slot graph with slot set $S$ and edge set $E \subseteq S \times S$; $V = \bigcup_{s \in S} V(s)$ is the set of vertices; the association rule $M \subseteq S \times V$ is a binary relation between $S$ and $V$, which maps each slot $s$ to

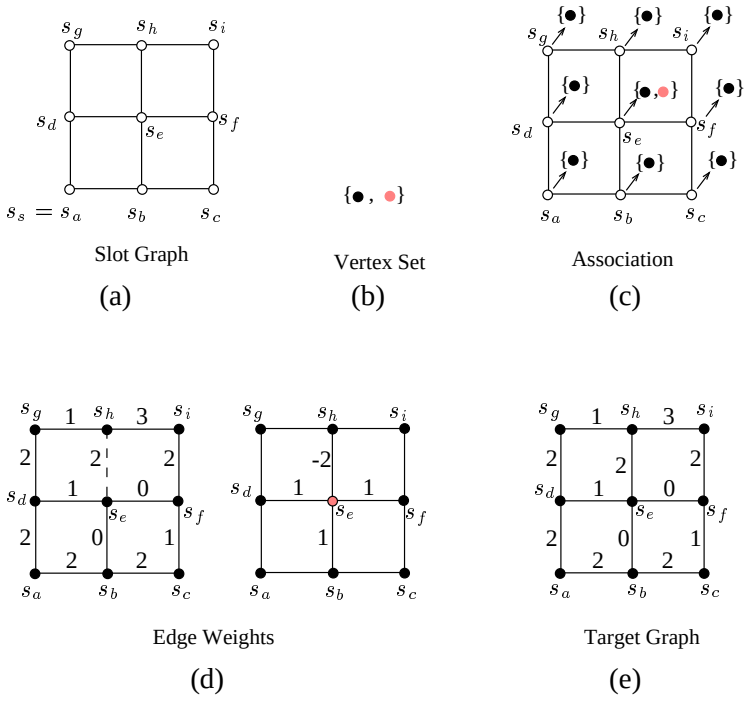


Figure 5: An example self-destructible graph assembly system.

its associated vertex set $V(s)$ (note that the sets $V(s)$ are *not* necessarily disjoint); for any edge $(s_a, s_b) \in E$, we define a weight function $w : V(s_a) \times V(s_b) \to \mathbb{Z}$ (here a weight is determined cooperatively by an edge $(s_a, s_b)$ and the two vertices occupying $s_a$ and $s_b$); $\langle s_s, v_s \rangle$ is a distinguished *seed slot* $s_s$ occupied by vertex $v_s$; $\tau \in \mathbb{N}$ is the *temperature* of the system. The size of a self-destructible graph assembly system is the bit representation of the system.

A *configuration* of $\tilde{G}$ is a function $A : S \to V \bigcup \{empty\}$, where *empty* indicates a slot being unoccupied. For ease of exposition, a configuration is alternatively referred to as a *graph*, denoted as $G$. When the context is clear, we simply refer to a slot occupied by a vertex as a *vertex*, for readability.

Given the above self-destructible graph assembly system, we aim at assembling a target graph, i.e. reaching a target configuration, $G_t$, starting with the seed vertex $\langle s_s, v_s \rangle$ and using the following *unit assembly operations*. In each unit operation, we temporarily attach a vertex $v$ to the current graph $G$ and obtain a graph $G'$, and then repeat the following procedure until no vertex can be removed from the assembly: inspect all the vertices in current graph $G'$; find the vertex $v'$ with the smallest *support*, i.e. the sum of the weights of edges between $v'$ and its assembled neighbors, and break the ties arbitrarily (note that $v'$ can be $v$); if the support to $v'$ is less than $\tau$, remove $v'$. This procedure ensures that when a vertex that repulses its assembled neighbors is incorporated in the existing assembly, all the vertices whose support drops below system temperature will be removed. However, in the case when a vertex to be attached exerts no repulsive force to its already assembled neighbors, the above standard unit assembly operation can be simplified as follows: a vertex can be assembled if the total support it receives from its assembled neighbors is equal to or greater than the system temperature $\tau$ – this is exactly the same as the operation in the accretive graph assembly model.

Figure 5 and Figure 6 give a concrete example of a self-destructible graph assembly system $\mathcal{T}$ and a sequence of unit assembly operations that assemble a target graph $G_t$ in $\mathcal{T}$. Figure 5 illustrates the assembly system $\mathcal{T} = \langle \tilde{G} = (S, E), V, M, w, \langle s_s, v_s \rangle, \tau \rangle$. Here, slot $s_a$ is designated as the distinguished seed slot $s_s$ and temperature $\tau$ is set to 2. Figure 5 (a) depicts the slot graph $\tilde{G} = (S, E)$, where $S = \{s_a, s_b, s_c, s_d, s_e, s_f, s_g, s_h, s_i\}$, $E = \{(s_a, s_b), (s_b, s_c), (s_a, s_d), (s_b, s_e), (s_c, s_f), (s_d, s_e), (s_e, s_f), (s_d, s_g), (s_e, s_h), (s_f, s_i), (s_g, s_h), (s_h, s_i)\}$. Figure 5 (b) gives the vertex set $V = \{black, grey\}$. Figure 5 (c) shows the association

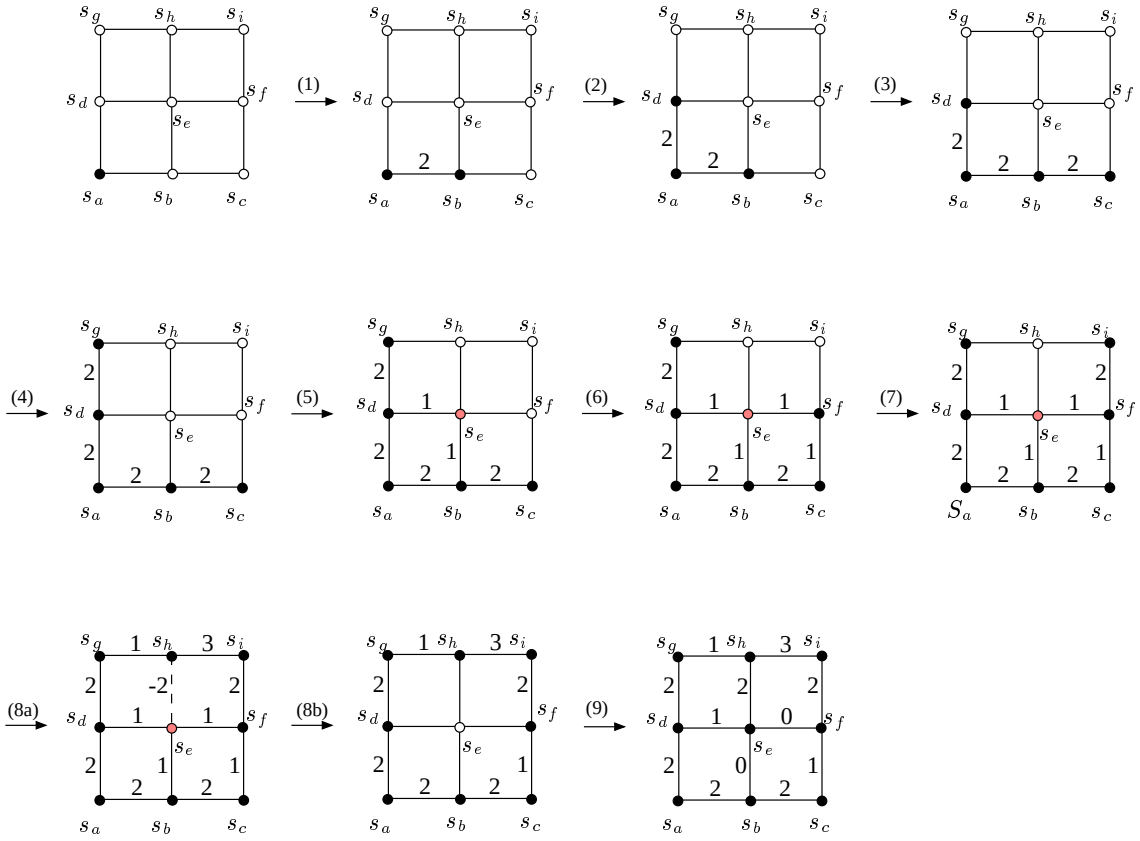


Figure 6: The sequence of operations that assemble the target graph $G_t$.

rule $M$: $V(s_e) = \{black, grey\}$; $V(s) = \{black\}$, for $s \in S \setminus s_e$. Figure 5 (d) illustrates $w$. A numerical value indicates the weight of an edge incident to two occupied slots. The left panel of Figure 5 (d) describes the cases when both slots incident to an edge are occupied by black vertices; the right panel describes the case when slot $s_e$ is occupied by a grey vertex but its neighboring slot is occupied by a black vertex. For example, the weight for edge $(s_e, s_h)$, when both $s_e$ and $s_h$ are occupied with black vertices, is $-2$. This negative weight is further indicated by the dashed edge. Figure 5 (e) depicts the target graph (configuration) $G_t$, where each the slot in $S$ is occupied by a black vertex, i.e. $A(s) = black$ for any $s \in S$.

An example sequence of unit assembly operations that sequentially assembles the target graph $G_t$ is illustrated step by step in Figure 6. We start with $\langle s_s, black \rangle$, where $s_s = s_a$. In step (1), a black vertex is put into slot $s_b$ and stays there, since the support it receives from the black vertex occupying slot $s_a$ is 2, which is greater than or equal to the system temperature $\tau = 2$. In step (5), a grey vertex occupies slot $s_e$ and is attached to existing assembly. It stays in slot $s_e$ since it receives a total support of 2 from its neighboring assembled vertices (support 1 from the black vertex occupying slot $s_b$ and support 1 from the black vertex occupying slot $s_d$). Step (8) has two stages (8a) and (8b). In step (8a), a black vertex is temporarily put into slot $s_h$. Now the grey vertex occupying slot $s_e$ has the least support among all the vertices in this temporary assembly. Since its support 1 is less than temperature $\tau = 2$, the grey vertex in $s_e$ is removed from the assembly in step (8b), according to the unit assembly operation rule. Now no vertex can be removed since all vertices have support greater than or equal to $\tau = 2$. In step (9), a black vertex is put into slot $s_e$ and this completes the assembly of the target graph.

Here we emphasize that in the above example, the grey vertex at slot $s_e$ serves as a "stepping stone" for assembling the target graph: its incorporation into the assembly enables the subsequent assembly of a black

vertex at slot $s_f$, which in turn effects the assembly of a black vertex at $s_i$. However, at this stage, the grey vertex at slot $s_e$ becomes a barrier for the progress of the assembly towards the target configuration – it must be popped off the assembly to evacuate slot $s_e$ for the assembly of a black vertex at $s_e$. This is achieved by the incorporation of a black vertex at slot $s_h$. This is precisely the power of (partial) self-destruction: the system actively gets rid of the undesirable components to ensure the progress of further assembly. Finally, we point out that the grey vertex associated with $s_e$ is indispensable for the assembly of the target graph. The reader can verify that without this grey vertex, the target graph cannot be sequentially constructed.

In the above example, the assembly of black vertex at slot $s_h$ "deterministically" and "irreversibly" pops off the grey vertex at slot $s_e$. However, the self-destructible graph assembly model can also exhibit interesting non-deterministic, reversible behavior under the following circumstance: the assembly of component $a$ pops off component $b$, while the immediate re-assembly of component $b$ can in turn knock off the newly assembled component $a$. For a concrete example, now assume that in Figure 6, the weight for edge $(s_h, s_i)$ (when slots $s_h$ and $s_i$ are occupied by black vertices) is 2 instead of 3. Then at step (8b) either the black vertex at slot $s_h$ or the grey vertex at slot $s_e$ can be removed, since both vertices have support $1 < 2 = \tau$ and we break ties arbitrarily. For the same reason, in the case when the grey vertex at $s_e$ is removed, an immediate reassembly of a grey vertex at slot $s_e$ can result in the disassembly of the black vertex at $s_h$. In this sense, the system at this stage becomes "non-deterministic" and "reversible". This property is used in the construction of a cyclic gadget, which provides the basis for our **PSPACE**-complete proof in Section 6.

Now we are ready to define the Self-Destructible Graph Assembly Problem (**DGAP**).

**Definition 5.1 Self-Destructible Graph Assembly Problem (DGAP)***: Given a self-destructible graph assembly system $\mathcal{T} = \langle G = (S, E), V, M, w, \langle s_s, v_s \rangle, \tau \rangle$ and a target graph (configuration) $G_t$, determine whether there exists a sequence of assembly operations such that $G_t$ can be assembled starting from $\langle s_s, v_s \rangle$.*

# 6 DGAP is PSPACE-complete

**Theorem 6.1** DGAP *is* **PSPACE***-complete.*

**Proof:** Recall that the **PSPACE**-complete problem IN-PLACE ACCEPTANCE is as follows: given a deterministic Turing machine (TM for short) $U$ and an input string $x$, does $U$ accept $x$ without leaving the first $|x| + 1$ symbols of the string [26]? We reduce IN-PLACE ACCEPTANCE to DGAP using a direct simulation of a deterministic TM $U$ on $x$ with self-destructible graph assembly in **PSPACE**.

The proof builds on 1) a classical technique for simulating TM using self-assembly of square tiles [28, 31], which takes exponential space for deciding **PSPACE**-complete languages; and 2) our new cyclic gadget, which helps the classical TM simulation to reuse space and thus achieve a **PSPACE** simulation. We will first reproduce the classical simulation; next introduce our modification to the classical simulation; then describe our cyclic gadget; finally integrate the cyclic gadget with the modified TM simulation to obtain a **PSPACE** simulation and thus conclude the proof.

**Classical TM simulation.** The classical scheme uses the assembly of vertices on a 2D square grid to mimic a TM's transition history [28, 31]. Consecutive configurations of TM are represented by successive horizontal rows of assembled-vertices.

Given a TM $U(Q, \Sigma, \delta, q_0)$, where $Q$ is a finite set of states, $\Sigma$ is a finite set of symbols, $\delta$ is the transition function, and $q_0 \in Q$ is the initial state, we construct a self-destructible assembly system $\mathcal{T} = \langle G = (S, E), V, M, w, \langle s_s, v_s \rangle, \tau \rangle$ as follows. The slot graph $G = (S, E)$ is an infinite 2D square grid; each node of the grid corresponds to a slot $s \in S$. A vertex $v \in V$ is represented as a quadruple $v = \langle a, b, c, d \rangle$, where $a$, $b$, $c$, and $d$ are referred to as the North, East, South, and West *'glues'* (see Figure 7). Each glue $x$ is associated with an integral strength $g(x)$. More specifically, we construct the following vertices:

- For each $s \in \Sigma$, construct a *symbol vertex* $\langle s, \gamma, s, \gamma \rangle$, where $\gamma$ is a special symbol $\notin \Sigma$.
- For each $\langle q, s \rangle \in Q \times \Sigma$, construct *state vertices* $\langle \langle q, s \rangle, \gamma, s, \overrightarrow{q} \rangle$ and $\langle \langle q, s \rangle, \overleftarrow{q}, s, \gamma \rangle$.

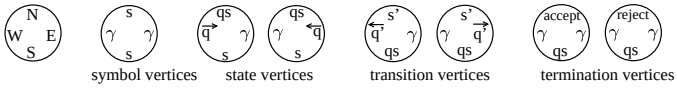


Figure 7: Vertices used in the basic TM simulation.

- For each transition $\langle q, s\rangle \to \langle q', s', \mathrm{L}\rangle$ (resp. $\langle q, s\rangle \to \langle q', s', \mathrm{R}\rangle$), where L (resp. R) is the head moving direction "Left" (resp. "Right"), construct a *transition vertex* $\langle s', \gamma, \langle q, s\rangle, \overleftarrow{q'}\rangle$ (resp. $\langle s', \overrightarrow{q'}, \langle q, s\rangle, \gamma\rangle$).
- For transition $\langle q, s\rangle \to$ ACCEPT (resp. REJECT), construct a *termination vertex* $\langle \mathrm{ACCEPT}, \gamma, \langle q, s\rangle, \gamma\rangle$ (resp. $\langle \mathrm{REJECT}, \gamma, \langle q, s\rangle, \gamma\rangle$).

The glue strength $g(\langle q, s\rangle)$ is set to 2; all other glue strengths are 1. Mapping relation $M$: every vertex in $V$ can be mapped to every slot in $S$. We next describe weight function $V \times V \times E \to \mathbb{Z}$. Consider two vertices $v_1 = \langle a, b, c, d\rangle$ and $v_2 = \langle a', b', c', d'\rangle$ connected by edge $e$, if $e$ is horizontal and $v_1$ lies to the East (resp. West) of $v_2$, the weight function is $g(b', d)$ (resp. $g(b, d')$); if $e$ is vertical and $v_1$ lies to the North (resp. South) of $v_2$, the weight function is $g(c, a')$ (resp. $g(a, c')$); where $g(x, y) = g(x)$ (resp. 0) if $x = y$ (resp. $x \neq y$). In other words, the edge weight for two neighboring vertices is the strength of the abutting glues, if the abutting glues are the same; otherwise it is 0.

It is straightforward to show the assembly of the vertices in $V$ on the slot graph $G = (S, E)$ simulates the operation of the TM $U$. Figure 8 (a) gives a concrete example to illustrate the simulation process as in [31]. Here we assume the bottom row in the assembly in Figure 8 (a) is pre-assembled.

**Our modified TM simulation.** We add two modifications to the classical simulation and obtain the scheme in Figure 8 (b): 1) a set of vertices are added to assemble an input row (bottom row in the figure) and 2) a dummy column is added to the leftmost of the assembly. For the construction, see the self-explanatory Figure 8 (b). The leftmost bottom vertex is the seed vertex and a thick line indicates a weight 2 edge. The reason for adding the dummy column is as follows. The glue strength $g(\langle q, s\rangle)$ is 2 in Figure 8 (a); this is necessary to initiate the assembly of a new row and hence a transition to next configuration. However, due to a subtle technical point discussed in Appendix A, we cannot allow weight 2 edge(s) in a column unless all the edges in this column have weight 2. So we add the leftmost dummy column of vertices connected by weight 2 edges, and this enables us to set $g(\langle q, s\rangle) = 1$ and thus avoid weight 2 edge other than those in the dummy column.

The modified scheme simulates a TM on input $x$ with the head initially residing at $s_0$ and never moving to the left of $s_0$. The assembly proceeds from bottom to top; within each row, it starts from the leftmost dummy vertex and proceeds to the right (note the difference in the assembly sequence in Figure 8 (a) and (b), as indicated by the thick grey arrows). Unfortunately, this assembly sequence can introduce error: e.g. in place of a state vertex $v_1 = \langle\langle q, s_i\rangle, \overleftarrow{q}, s_i, \gamma\rangle$, a symbol vertex $v_2 = \langle s_i, \gamma, s_i, \gamma\rangle$ can get assembled since $v_1$ and $v_2$ share the same South glue and the same West glue. Fortunately, this mistake can be corrected by our final assembly system that performs a *self-destructible* simulation of a Turing machine, as described later.

**Our cyclic gadget.** The above strategy to simulate TM by laying out its configurations one above another can result in a graph with height exponential in the size of the input ($|x|$): the height of graph is precisely the number of transitions plus one. A crucial observation is that once row $i$ is assembled, row $i - 1$ is no longer needed: row $i$ holds sufficient information for assembling row $i + 1$ and hence for the simulation to proceed. Thus, we can evacuate row $i - 1$ and reuse the space to assemble a future row, say row $i + 2$. Using this trick, we can shrink the number of rows from an exponential number to a constant. The self-destructible graph assembly model can provide us with precisely this power. To realize this power of evacuating and reusing space, we construct a *cyclic gadget*, shown Figure 9 (a). The gadget contains three kinds of vertices: the *computational vertices* ($a$, $b$, and $c$) that carry out the actual simulation of the Turing machine; the *knocking vertices* ($x$, $y$, and $z$) that serve to knock off the computational vertices and thus

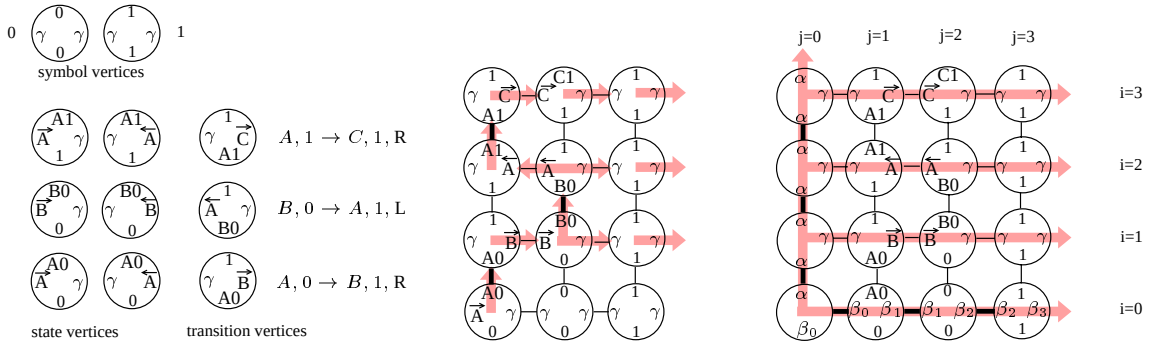


Figure 8: (a) An example classical simulation of a Turing machine $U(Q, \Sigma, \delta, q_0)$, where $Q = \{A, B, C\}$; $\Sigma = \{0, 1\}$; transition function $\delta$ is shown in the figure; $q_0 = A$. The top of the left panel shows two symbol vertices; below are some example transition rules and the corresponding state vertices and transition vertices. The right panel illustrates the simulation of $U$ on input $001$ (simulated as the bottom row, which is assumed to be preassembled), according to the transition rules in the figure; the head's initial position is on the leftmost vertex. Each transition of $U$ adds a new row. (b) Our modified scheme. The leftmost bottom vertex is the seed vertex. The leftmost column is the dummy column. In both (a) and (b), a thick line indicates a weight 2 edge; a thin line indicates weight 1; thick grey arrows indicate the assembly sequence.

release the space; the *anchor vertices* ($x'$, $y'$, and $z'$) that anchor the knocking vertices. Edge weights are labeled in the figure.

For ease of exposition, we introduce a little more notation. The event in which a new vertex $b$ is attached to a pre-assembled vertex $a$ is denoted as $a \cdot b$; the event in which $a$ pops off $b$ is denoted as $a \dashv b$.

We next describe the operation of the cyclic gadget. We require that anchor vertices $x'$, $y'$, and $z'$ and computational vertex $a$ are pre-assembled. The anchor vertices and computational vertices will keep getting assembled and then popped off in a counterclockwise fashion. First, $b$ is attached to $a$ (event $a \cdot b$). Then $x$ is attached to $b$ (event $b \cdot x$). At this point, $x$ has total support 1 from $b$, $x'$, and $a$ (providing support 2, 2, and $-3$, respectively); $a$ has total support $-1$ from $b$ and $x$ (providing support 2 and -3, respectively). Since temperature is 2, $x$ will knock off $a$ ($x \dashv a$). Next, we have $b \cdot c$ followed by $c \cdot y$. At this point, $y$ has total support 1 from $c$ and $y'$; $b$ has total support 1 from $x$ and $c$. Therefore, either $y \dashv b$ or $b \dashv y$ can happen, but $y \dashv b$ is in the desired counterclockwise direction. Next, we will have cycles of (reversible) events. In summary, the following sequence of events occur, providing the desired cyclicity:
$a \cdot b$, $b \cdot x$, $x \dashv a$; $b \cdot c$, $c \cdot y$, $y \dashv b$; $(c \cdot a$, $a \dashv x$, $a \cdot z$, $z \dashv c$; $a \cdot b$, $b \dashv y$, $b \cdot x$, $x \dashv a$; $b \cdot c$, $c \dashv z$, $c \cdot y$, $y \dashv b)^*$;

The steps in the $()$ will keep repeating. Note that the steps in the $()$ are reversible, which facilitates our reversible simulation of a Turing machine below.

**Integrating cyclic gadget with TM simulation.** We next integrate the cyclic gadget with the modified $TM$ simulation in Figure 8 (b). In the resulting scheme, we obtain a reversible simulation of a deterministic TM on a slot graph of constant height, by evacuating old rows and reusing the space: row $i$ is evacuated after the assembly of row $i + 1$, providing space for the assembly of row $i + 3$.

Figure 9 (b) illustrates the integrated scheme. Slot rows $A$, $B$, and $C$ correspond to rows $i = 3r$, $i = 3r + 1$, and $i = 3r + 2$ in Figure 8 (b), respectively. Let $|x| = n$. $A$ is a sequence of slots $A = [a_0, a_1, \ldots, a_{n+1}]$; similarly, $B = [b_0, b_1, \ldots, b_{n+1}]$ and $C = [c_0, c_1, \ldots, c_{n+1}]$ as in Figure 9 (b). Slots $a_0$, $b_0$, and $c_0$ are dummy slots (corresponding to the dummy column in Figure 8 (b)). For each $a_j$, $b_j$, and $c_j$, we construct a cyclic gadget by introducing slots $x_j$, $y_j$, $z_j$, $x'_j$, $y'_j$, and $z'_j$.

The edge weights are shown in the figure. We emphasize that the weight for an edge between two computational vertices (vertices in $A$, $B$, and $C$) $u$ and $v$ is set to the glue strength if $u$ and $v$ have the same glue on their abutting sides; otherwise it is 0. This is consistent with the scheme in Figure 8 (b) and helps to ensure the proper operation of the computational assembly. In contrast, the weight for any other edge is

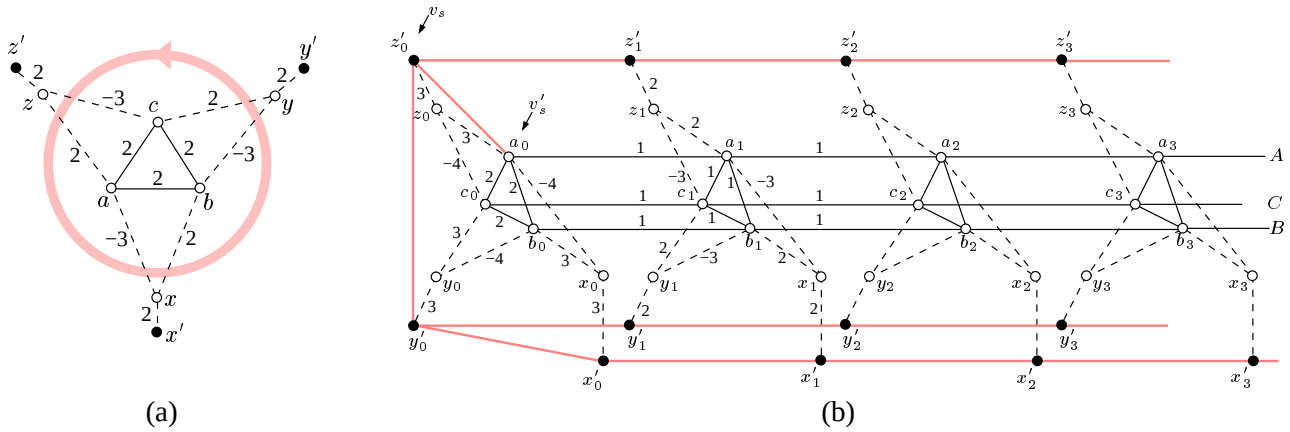


Figure 9: (a) The construction and operation of our cyclic gadget. The counterclockwise grey cycle indicates the desired sequence of events. (b) The integrated scheme. Grey edges have weight 2. Unlabeled black edges have weight 1. $v_s$ indicates the seed vertex; $z_0$ is the seed slot. $v'_s$ indicates a distinguished computational "seed".

always set to the value shown in Figure 9 (b), regardless of the actual computational vertices present in the slots in $A$, $B$, and $C$; this ensures the proper operation of the cyclic gadget.

Slot $z'_0$ is designated as the seed slot $s_s$ and one of its associated vertices as the seed vertex $v_s$ and the temperature is again set to 2.

The assembly proceeds as follows. First, the frame of anchoring vertices (subgraph with grey edges) will be assembled, starting from the seed vertex at $z'_0$. The seed vertex at $z'_0$ will pull in a distinguished computational vertex $v'_s$ (corresponding to the seed vertex in Figure 8 (b)) at slot $a_0$, and $v'_s$ subsequently initiates the assembly of the input row (corresponding to the bottom row in Figure 8 (b)). Then the computational vertices will assemble, simulating the process shown in Figure 8 (b). Meanwhile, the cyclic gadget functions along each layer of $a_j$, $b_j$, and $c_j$ (corresponding to column $j$ in Figure 8 (b)), effecting the reusing of space. More specifically, vertices corresponding to those in rows $i = 3r$, $i = 3r + 1$, and $i = 3r + 2$ in Figure 8 (b) will be assembled in $A$, $B$, and $C$ respectively. Similar to the process in Figure 9 (a), row $i + 1$ gets assembled with the support from row $i$, and subsequently pulls in popper vertices, which knock off row $i$ and thus evacuate space for future row $i + 3$ to assemble. Within a row, the vertices are knocked off sequentially from left to right, starting with the dummy vertex.

Errors that occur during the assembly can be subsequently corrected as described in Appendix A. Some other subtle technical points are also discussed in Appendix A.

**Concluding the proof.** We set the target graph $G_t$ as a *complete* row of vertices that contains the ACCEPT termination vertex $\langle \text{ACCEPT}, \gamma, \langle s, q \rangle, \gamma \rangle$. Then $G_t$ can be assembled if and only if TM $U$ accepts $x$. We insist $G_t$ to be a complete row of vertices (occupying $s_0$, $s_1$, ..., $s_{|x|+1}$, where $s \in \{a, b, c\}$) to avoid false positives. Note the size of the slot graph used in the proof is polynomial in the size of the input $|x|$ and hence our simulation is in **PSPACE**. □

**Corollary 6.2** *6-*DEGREE DGAP *is* **PSPACE***-complete.*

## 7 Conclusion

In this paper, we define two new models of self-assembly and obtain the following complexity results: 4-DEGREE AGAP is **NP**-complete; 5-DEGREE PAGAP is **NP**-complete; #AGAP and SAGAP are #**P**-complete; 6-DEGREE DGAP is **PSPACE**-complete. One immediate open problem is to determine the complexity of these problems with lower degrees. In addition, it would be nice to find approximation algorithms for the optimization version of the **NP**-hard problems. Note AGAP can be solved in polynomial

time if only positive edges are permitted in graph $G$, using a greedy heuristic. In contrast, when negative edges are allowed, for each negative edge $e = (v_1, v_2)$, we need to decide the relative order for assembling $v_1$ and $v_2$. Thus $k$ negative edges will imply $2^k$ choices, and we have to find out whether any of these $2^k$ choices can result in the assembly of the target graph. This is the component that makes the problem hard.

# References


1. http://mrsec.wisc.edu/edetc/selfassembly/.
2. DNA triangles and self-assembled hexagonal tilings. *J. Am. Chem. Soc.*, 126:13924–13925, 2004.
3. L. Adleman. Towards a mathematical theory of self-assembly. Technical Report 00-722, University of Southern California, 2000.
4. L. Adleman, Q. Cheng, A. Goel, and M. D. Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748. ACM Press, 2001.
5. L. Adleman, Q. Cheng, A. Goel, M. D. Huang, D. Kempe, P. M. de Espans, and P. W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 23–32. ACM Press, 2002.
6. L. Adleman, Q. Cheng, A. Goel, M. D. Huang, and H. Wasserman. Linear self-assemblies: Equilibria, entropy, and convergence rate. In *Sixth International Conference on Difference Equations and Applications*, 2001.
7. L. Adleman, J. Kari, L. Kari, and D. Reishus. On the decidability of self-assembly of infinite ribbons. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 530–537, 2002.
8. G. Aggarwal, M. H. Goldwasser, M. Y. Kao, and R. T. Schweller. Complexities for generalized models of self-assembly. In *Proceedings of 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 880–889. ACM Press, 2004.
9. N. Bowden, A. Terfort, J. Carbeck, and G. M. Whitesides. Self-assembly of mesoscale objects into ordered two-dimensional arrays. *Science*, 276(11):233–235, 1997.
10. R. F. Bruinsma, W. M. Gelbart, D. Reguera, J. Rudnick, and R. Zandi. Viral self-assembly as a thermodynamic process. *Phys. Rev. Lett.*, 90(24):248101, 2003 June 20.
11. H. L. Chen, Q. Cheng, A. Goel, M. D. Huang, and P. M. de Espanes. Invadable self-assembly: Combining robustness with efficiency. In *Proceedings of the 15th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 890–899, 2004.
12. Q. Cheng and P. M. de Espanes. Resolving two open problems in the self-assembly of squares. Technical Report 03-793, University of Southern California, 2003.
13. Q. Cheng, A. Goel, and P. Moisset. Optimal self-assembly of counters at temperature two. In *Proceedings of the first conference on Foundations of nanoscience: self-assembled architectures and devices*, 2004.
14. Matthew Cook, Paul W. K. Rothemund, and Erik Winfree. Self-assembled circuit patterns. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 91–107, 2004.
15. K. Fujibayashi and S. Murata. A method for error suppression for self-assembling DNA tiles. In *DNA Based Computing 10*, pages 284–293, 2004.
16. E. Klavins. Directed self-assembly using graph grammars. In *Foundations of Nanoscience: Self Assembled Architectures and Devices*, 2004.
17. E. Klavins, R. Ghrist, and D. Lipsky. Graph grammars for self-assembling robotic systems. In *Proceedings of the International Conference on Robotics and Automation*, 2004.
18. Eric Klavins. Toward the control of self-assembling systems. In *Control Problems in Robotics*, volume 4, pages 153–168. Springer Verlag, 2002.
19. T. H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J. H. Reif, and N. C. Seeman. The construction, analysis, ligation and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, 122:1848–1860, 2000.
20. M. G. Lagoudakis and T. H. LaBean. 2-D DNA self-assembly for satisfiability. In *DNA Based Computers V*, volume 54 of *DIMACS*, pages 141–154. American Mathematical Society, 2000.
21. M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, New York, second edition, 1997.
22. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.
23. D. Liu, M. S. Wang, Z. X. Deng, R. Walulu, and C. D. Mao. Tensegrity: Construction of rigid DNA triangles

with flexible four-arm dna junctions. *J. Am. Chem. Soc.*, 126:2324–2325, 2004.
24. C. Mao, W. Sun, and N. C. Seeman. Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. *J. Am. Chem. Soc.*, 121:5437–5443, 1999.
25. A. A. Middleton. Computational complexity of determining the barriers to interface motion in random systems. *Phys. Rev. E*, 59(3):2571–2577, 1999.
26. C. M. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Inc., 1st edition, 1994.
27. J. H. Reif, S. Sahu, and P. Yin. Compact error-resilient computational DNA tiling assemblies. In *Proc. 10th International Meeting on DNA Computing*, pages 248–260, 2004.
28. R. M. Robinson. Undecidability and non periodicity of tilings of the plane. *Inventiones Math*, 12:177–209, 1971.
29. P. W. K. Rothemund. Using lateral capillary forces to compute by self-assembly. *Proc. Natl. Acad. Sci. USA*, 97(3):984–989, 2000.
30. P. W. K. Rothemund. *Theory and Experiments in Algorithmic Self-Assembly.* PhD thesis, University of Southern California, 2001.
31. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 459–468. ACM Press, 2000.
32. Phiset Sa-Ardyen, Natasa Jonoska, and Nadrian C. Seeman. Self-assembling DNA graphs. *Lecture Notes in Computer Science*, 2568:1–9, 2003.
33. Rebecca Schulman, Shaun Lee, Nick Papadakis, and Erik Winfree. One dimensional boundaries for DNA tile self-assembly. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 108–125, 2004.
34. Rebecca Schulman and Erik Winfree. Programmable control of nucleation for algorithmic self-assembly. In *DNA Based Computers 10*, LNCS, 2005.
35. David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. In *DNA Based Computers 10*, LNCS, 2005.
36. A. Strasser, L. O'Connor, and V.M. Dixit. Apoptosis signaling. *Annu. Rev. Biochem.*, 69:217–245, 2000.
37. H. Wang. Proving theorems by pattern recognition ii. *Bell Systems Technical Journal*, 40:1–41, 1961.
38. E. Winfree and R. Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 126–144, 2004.
39. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–544, 1998.
40. E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. F. Landweber and E. B. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS*, pages 191–213. American Mathematical Society, 1999.
41. H. Yan, T. H. LaBean, L. Feng, and J. H. Reif. Directed nucleation assembly of DNA tile complexes for barcode patterned DNA lattices. *Proc. Natl. Acad. Sci. USA*, 100(14):8103–8108, 2003.
42. H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301(5641):1882–1884, 2003.

# Appendices

## A Some technical issues in proof of Theorem 6.1

This section discusses some technical points regarding the operation of the integrated scheme in Figure 9 (b).

**Error elimination.** During the assembly of computational vertices errors can happen as described in "our modified TM simulation". However, such errors will not disrupt the correct operation of our assembly system for the following two reasons. First, the error cannot propagate horizontally. Second, thanks to the reversible nature of our assembly system, the incorrectly assembled vertices will be popped off and eventually only the correct simulation process can proceed to its full extent. As a consequence, if TM $U$ accepts $x$, our simulation can guarantee that our simulation will eventually follow a path to the final acceptance state; while if TM $U$ rejects $x$, no such path exists.

**Edge weight assignment.** 1). The weight for the edge connecting vertices $v_s = z_0$ and $v'_s$ is 2; while the weight for an edge connecting $z'_0$ and subsequent vertices other than $v'_s$ that occupy slot $a_0$ is 0. This ensures the correct operation of the cyclic gadget for the dummy slots. 2).The assembly of the first row (input row) involves computational vertices with glue strength 2 (rather than 1) and hence weight 2 edges between neighboring vertices in this row. However *no* modification on the edge weight of the edges incident to the knocking vertices and anchor vertices is required to accommodate this edge weight difference: the initial step $(a \cdot b, b \cdot x, x \dashv a)$ is irreversible and it is straightforward to check that $x \dashv a$ can occur successfully. 3). Except for the edges connecting dummy vertices, no weight 2 edge exists between the computational vertices after the evacuation of the input row. This is essential for upper bounding the number of vertices associated with each slot: otherwise, an exponential number of popper vertices and anchor vertices would be required.