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

John H. Reif[1], Sudheer Sahu[1], and Peng Yin[1]

Department of Computer Science, Duke University
Box 90129, Durham, NC 27708-0129, USA.
{reif,sudheer,py}@cs.duke.edu

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 nano-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. 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

* 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.

in DNA based nanoscience, as exemplified by the diverse set of DNA lattices made from self-assembled branched DNA molecules (DNA tiles) [9, 15, 23, 25, 30, 43, 44]. Another nanoscale example is the self-assembly of peptide molecules [8]. Self-assembly is also used for mesoscale construction, for example, via the use of capillary forces [29] or magnetic forces [1] to provide attraction and repulsion between mesoscale tiles and other objects.

Building on classical Wang tiling models [40] 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 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 $\Theta(\frac{\log n}{\log \log n})$ [3, 31]. The upper bound is obtained by simulating a binary counter and the lower bound by analyzing the Kolmogorov complexity of the tiling system. The model was later extended by Adleman *et al.* to include the time complexity of generating specified assemblies [3]. 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, 6, 10–14, 19, 27, 33–38, 41, 42].

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 nano-fabrications, the complexity of some other important aspects of self-assembly requires further study:

- 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 [7, 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 in the self-assembly system was posed as an open question by Adleman and colleagues in [3]. Though there has been previous work on the kinetics of such systems [20], no complexity theoretic study has been directed towards such systems.
- Tile Assembly Model captures well assembled structures of two dimensional square grids, but are not well adaptable to study assemblies of general graph structure. However, many molecular self-assemblies using DNA and other materials involve the assembly of more diverse graph-like structures in both two and three dimensions. Pioneer work in modeling DNA self-assembly as graphs include [16–18, 32]. In particular, Jonoska et al studied the computational capacity of the self-assembly of realistic DNA graphs and showed that 3SAT and 3-colorability problems can be solved in constant laboratory steps in theory [16–18]. In addition, Seeman's group

have experimentally constructed topoisomers of self-assembled DNA graphs [32]. Klavins showed how to produce desired topology of self-assembled structures with planar graph structure using graph grammars [21, 22].

In this paper, we study the cooperative effect of repulsion and attraction on the complexity of the self-assembly system in a graph setting. This approach thus allows the study of a more general class of assemblies.

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 AGAP or 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 (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 [39]. 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 [39]. 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.

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 [2, 5]. These previous studies use 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 part 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

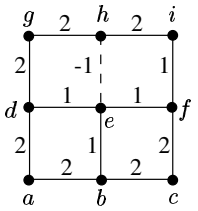

Fig. 1. An example of graph assembly in the accretive model

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 τ, where support is the sum of the weights of the edges between x and its assembled neighbors.

Figure 1 gives an example. Here the temperature is set to 2. 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, $\mathcal{O}_\mathcal{T} = (v_s = v_0 \prec v_1 \prec$

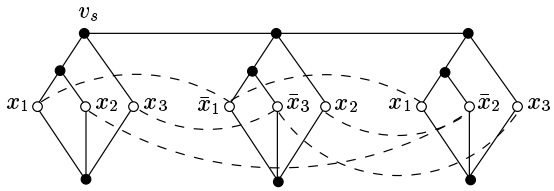

Fig. 2. A graph construction corresponding to an AGAP reduction from 3SAT formula $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$. An edge between two literal vertices is depicted as a dashed arch and assigned weight -1; all other edges have weight 2

$v_2 \prec \cdots \prec v_{n-1}$) 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.

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

Definition 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 "knocked off" by the subsequent assembly of any other vertex. If we relax this assumption, we will obtain a self-destructible model, which is described in Section 5.

3 AGAP and PAGAP are NP-complete

3.1 4-DEGREE AGAP is NP-complete

Lemma 1. AGAP *is in* **NP**.

Proof. Given an assembly ordering of the vertices, sequentially check whether each vertex can be assembled. This takes polynomial time. □

Recall that the **NP**-complete 3SAT problem asks: Given a Boolean formula ϕ in conjunctive normal form with each clause containing 3 literals, determine whether ϕ is satisfiable [26]. 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 2. AGAP *is* **NP***-hard.*

Proof. Given a 3SAT formula ϕ 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 ϕ. We will then show that the satisfiability problem of ϕ can be reduced (in logarithmic space) to the sequential constructibility problem of G in $\mathcal{T}$.

For each clause in ϕ, construct a *clause gadget* as in Figure 2. For each literal, we construct a *literal vertex* (colored white). We further add *top vertices* (black) above and *bottom vertices* (black) below the literal vertices. We next take care of the structure of formula ϕ as follows. Connect all the clause gadgets sequentially via their top vertices as in Figure 2; 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 $\mathcal{T} = \langle G, v_s, w, \tau \rangle$.

The following proposition implies the lemma.

Proposition 1. *There is an assembly ordering $\mathcal{O}$ for $\mathcal{T}$ if and only if ϕ is satisfiable.*

$\Rightarrow$

First we show that if ϕ 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 ϕ implies that every clause in ϕ 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 ϕ). Hence the total support for v will be at least 2 ($\geq \tau$).

$\Leftarrow$

Suppose that there exists an assembly ordering $\mathcal{O}$, then we can derive a satisfying truth assignment T for ϕ. 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 T satisfies ϕ, 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

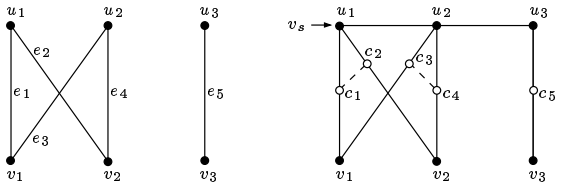

Fig. 3. (a) and (b) show an example bipartite graph B and the corresponding graph G used in the proof of Lemma 4, respectively. In (b), c_i's denote connector vertices (colored white); u_1 is the seed vertex. The weight of an edge connecting two connector vertices (dashed line) is -4; the weight of any other edge is 2

total support of v is thus at most 1, less than temperature $\tau = 2$. Contradiction. Hence T must satisfy ϕ. □

We note that the technique of translating 3SAT formula into graph structure by modeling variables as vertices and connecting complement literals is a classical technique [26], and has also been used powerfully in other different graph self-assembly context [18].

The following theorem follows immediately from Lemma 1 and Lemma 2.

Theorem 1. 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 2 is 4. Hence we have

Corollary 1. *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. Here, we show PAGAP is **NP**-hard by a reduction from the **NP**-hard planar three-satisfiability problem (P3SAT) [24]. The reduction is in similar spirit as that in the proof of Lemma 1. For lack of space, we skip the proof and only state our results.

Theorem 2. PAGAP *is* **NP**-*complete.*

Corollary 2. *5*-DEGREE PAGAP *is* **NP**-*complete.*

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. $\tilde{\mathcal{O}}$ 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. 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 3. #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. #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 3 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. Thus, 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 3 (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, 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 the 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. $\square$

Lemma 3 and Lemma 4 imply

Theorem 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 next extend the accretive graph self-assembly model to stochastic accretive graph self-assembly model. 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. 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 "knocked off" by the later 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 repulses an already assembled vertex b can make b unstable and hence "knock" 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

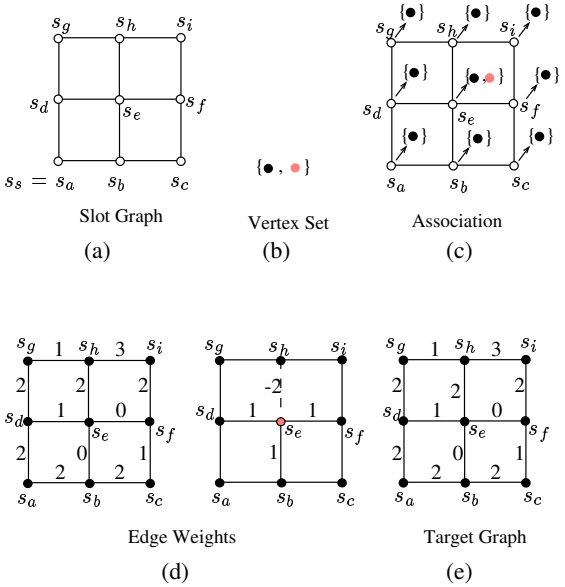

Fig. 4. An example self-destructible graph assembly system

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 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 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 un-occupied. 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 τ, 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 τ – this is exactly the same as the operation in the accretive graph assembly model.

Figure 4 gives a concrete example of a self-destructible graph 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 τ is set to 2. Figure 4 (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 4 (b) gives the vertex set $V = \{black, grey\}$. Figure 4 (c) shows the association rule M: $V(s_e) = \{black, grey\}$; $V(s) = \{black\}$, for $s \in S \setminus s_e$. Figure 4 (d) illustrates w. A numerical value indicates the weight of an edge incident to two occupied slots. The left panel of Figure 4 (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; when s_e is occupied by a grey vertex and s_h by a black vertex, the weight is -2. The negative weight is indicated by a dashed edge. Figure 4 (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 $s \in S$.

Now we are ready to define the Self-Destructible Graph Assembly Problem ($\mathbf{DGAP}$).

Definition 2. 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 5. DGAP *is* $\mathbf{PSPACE}$*-complete.*

Proof. Recall that the $\mathbf{PSPACE}$-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 $\mathbf{PSPACE}$.

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 $\mathbf{PSPACE}$-complete languages; and 2) our new cyclic gadget, which helps the classical TM simulation to reuse space and thus achieve a $\mathbf{PSPACE}$ 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, Σ is a finite set of symbols, δ 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 5). Each glue x is associated with an integral strength $g(x)$. More specifically, we construct the following vertices:

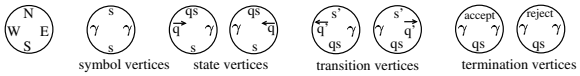

Fig. 5. Vertices used in the basic TM simulation

- For each $s \in \Sigma$, construct a *symbol vertex* $\langle s, \gamma, s, \gamma \rangle$, where γ 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 $\langle \langle q, s \rangle, \overleftarrow{q}, s, \gamma \rangle$.
- 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 that the assembly of the vertices in V on the slot graph $G = (S, E)$ simulates the operation of the TM U. Figure 6 (a) gives a concrete example to illustrate the simulation process as in [31]. Here we assume the bottom row in the assembly in Figure 6 (a) is pre-assembled.

Our modified TM simulation. We add two modifications to the classical simulation and obtain the scheme in Figure 6 (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 6 (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 6 (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 explained later (in the part "Integrating cyclic gadget with TM simulation"), 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 6 (a) and (b), as indicated by the thick grey arrows).

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 the 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 in Figure 7 (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 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 knocks 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 knocked 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 the 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:
$a \cdot b$, $b \cdot x$, $x \dashv a$; $b \cdot c$, $c \cdot y$, $y \dashv b$; ($c \cdot a$, $a \dashv x$, $a \cdot z$, $z \dashv c$; $a \cdot b$, $b \dashv y$, $b \cdot x$, $x \dashv a$; $b \cdot c$, $c \dashv z$, $c \cdot y$, $y \dashv b$)*;

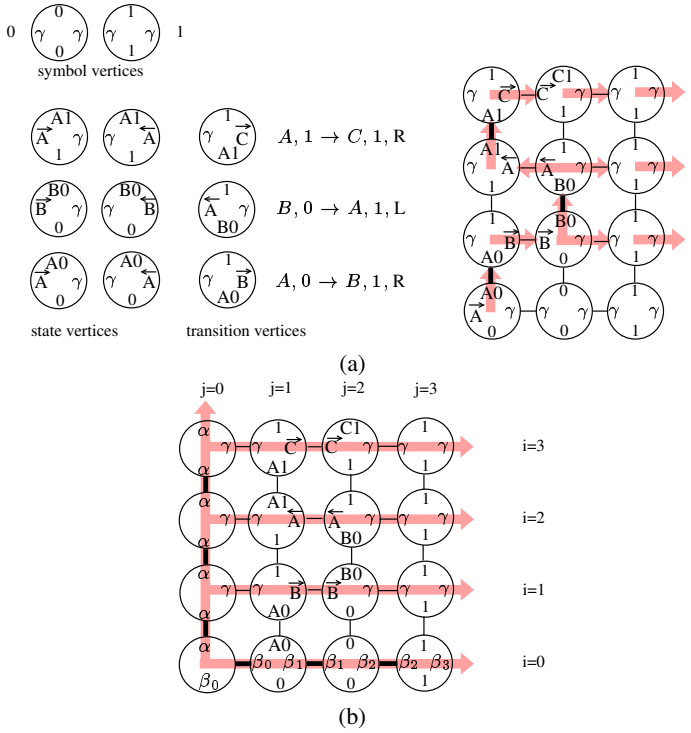

Fig. 6. (a) An example classical simulation of a Turing machine $U(Q, \Sigma, \delta, q_0)$, where $Q = \{A, B, C\}$; $\Sigma = \{0, 1\}$; transition function δ is shown in the figure; $q_0 = A$. The top of the left panel shows two symbol vertices; below are some example transition rules and the corresponding 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

The steps in the () will keep repeating. Note that the steps in the () are reversible, which will facilitate 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 6 (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 7 (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 6 (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 7 (b). Slots a_0, b_0, and c_0 are dummy slots

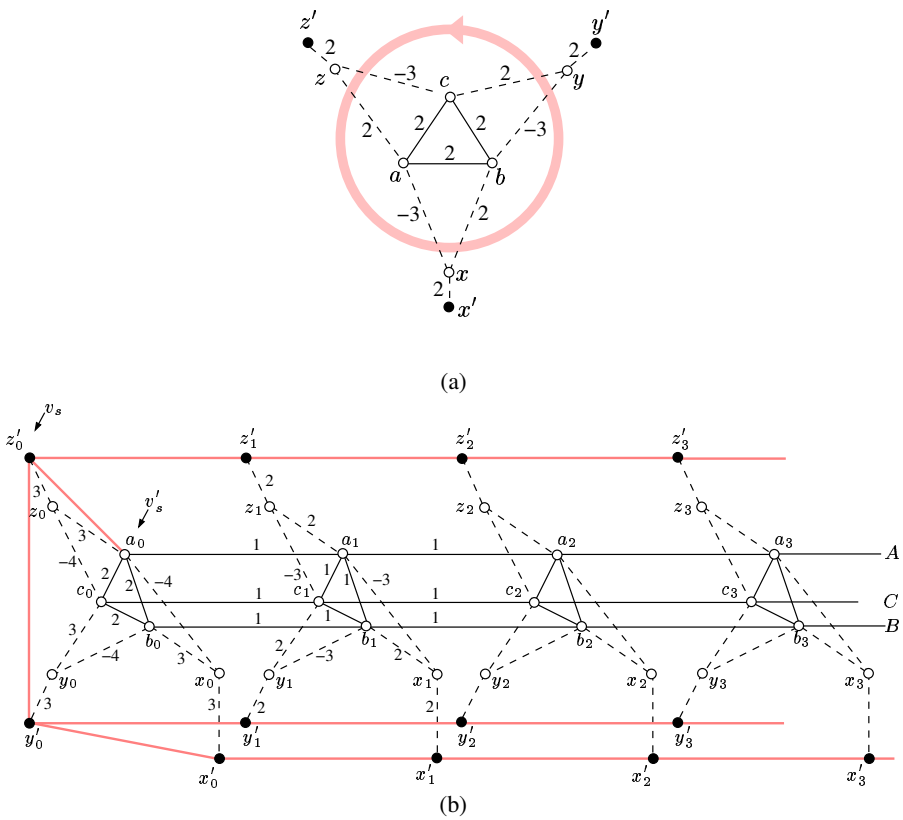

Fig. 7. (a) The construction and operation of our cyclic gadget. The counterclockwise grey cycle indicates the desired sequence of events. (b) The integrated scheme. Grey edges have weight 2. Unlabeled black edges have weight 1. v_s indicates the seed vertex; z_0 is the seed slot. v'_s indicates a distinguished computational "seed"

(corresponding to the dummy column in Figure 6 (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.

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 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 6 (b) and helps to ensure the proper operation of the computational assembly. In contrast, the weight for any other edge is always set to the value shown in Figure 7 (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.

There are some subtle technical points regarding edge weight assignment. First, 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. Second, 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. Third, 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 knocking vertices and anchor vertices would be required.

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 6 (b)) at slot a_0, and v_s' subsequently initiates the assembly of the input row (corresponding to the bottom row in Figure 6 (b)). Then the computational vertices will assemble, simulating the process shown in Figure 6 (b). Meanwhile, the cyclic gadget functions along each layer of a_j, b_j, and c_j (corresponding to column j in Figure 6 (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 6 (b) will be assembled in A, B, and C respectively. Similar to the process in Figure 7 (a), row $i + 1$ gets assembled with the support from row i, and subsequently pulls in knocking 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.

Concluding the proof. We set the target graph G_t as a *complete* row of vertices containing 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 $\mathbf{PSPACE}$. □

Corollary 3. *6*-DEGREE DGAP *is* $\mathbf{PSPACE}$*-complete.*

7 Conclusion

In this paper, we define two new models of self-assembly and obtain the following complexity results: 4-DEGREE AGAP is $\mathbf{NP}$-complete; 5-DEGREE PAGAP is $\mathbf{NP}$-complete; #AGAP and SAGAP are $\#\mathbf{P}$-complete; 6-DEGREE DGAP is $\mathbf{PSPACE}$-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 $\mathbf{NP}$-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 re-

sult 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. L. Adleman. Towards a mathematical theory of self-assembly. Technical Report 00-722, University of Southern California, 2000.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. N. Chelyapov, Y. Brun, M. Gopalkrishnan, D. Reishus, B. Shaw, and L. Adleman. DNA triangles and self-assembled hexagonal tilings. *J. Am. Chem. Soc.*, 126:13924–13925, 2004.
10. 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.
11. H.L. Chen and A. Goel. Error free self-assembly using error prone tiles. In *DNA Based Computers 10*, pages 274–283, 2004.
12. 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.
13. M. Cook, P.W.K. Rothemund, and E. Winfree. Self-assembled circuit patterns. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 91–107, 2004.
14. K. Fujibayashi and S. Murata. A method for error suppression for self-assembling DNA tiles. In *DNA Based Computing 10*, pages 284–293, 2004.
15. Y. He, Y. Chen, H. Liu, A.E. Ribbe, and C. Mao. Self-assembly of hexagonal DNA two-dimensional (2D) arrays. *J. Am. Chem. Soc.*, 127:12202–12203, 2005.
16. N. Jonoska, S.A. Karl, and M. Saito. Three dimensional DNA structures in computing. *BioSystems*, 52:143–153, 1999.
17. N. Jonoska and G.L. McColm. A computational model for self-assembling flexible tiles. *Unconventional Computing*. To appear, 2005.
18. N. Jonoska, P. Sa-Ardyen, and N.C. Seeman. Genetic programming and evolvable machines. *Computation by Self-assembly of DNA Graphs*, 4.
19. M. Kao and R. Schweller. Reduce complexity for tile self-assembly through temperature programming. In *Proceedings of 17th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. To appear. ACM Press, 2006.

20. E. Klavins. Toward the control of self-assembling systems. In *Control Problems in Robotics*, volume 4, pages 153–168. Springer Verlag, 2002.
21. E. Klavins. Directed self-assembly using graph grammars. In *Foundations of Nanoscience: Self Assembled Architectures and Devices*, Snowbird, UT, 2004.
22. 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.
23. 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.
24. D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.
25. J. Malo, J.C. Mitchell, C. Venien-Bryan, J.R. Harris, H. Wille, D.J. Sherratt, and A.J. Turberfield. Engineering a 2D protein-DNA crystal. *Angew. Chem. Intl. Ed.*, 44:3057–3061, 2005.
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, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA sierpinski triangles. *PLoS Biology 2 (12)*, 2:e424, 2004.
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. P. Sa-Ardyen, N. Jonoska, and N.C. Seeman. Self-assembling DNA graphs. *Lecture Notes in Computer Science*, 2568:1–9, 2003.
33. S. Sahu, P. Yin, and J.H. Reif. A self assembly model of time-dependent glue strength. In *Proc. 11th International Meeting on DNA Computing*, pages 113–124, 2005.
34. R. Schulman, S. Lee, N. Papadakis, and E. Winfree. One dimensional boundaries for DNA tile self-assembly. In *DNA Based Computers 9*, volume 2943 of *LNCS*, pages 108–125, 2004.
35. R. Schulman and E. Winfree. Programmable control of nucleation for algorithmic self-assembly. In *DNA Based Computers 10*, LNCS, 2005.
36. R. Schulman and E. Winfree. Self-replication and evolution of DNA crystals. In *The 13th European Conference on Artificial Life (ECAL)*, 2005.
37. D. Soloveichik and E. Winfree. Complexity of compact proofreading for self-assembled patterns. In *Proc. 11th International Meeting on DNA Computing*, pages 125–135, 2005.
38. D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. In *DNA Based Computers 10*, LNCS, 2005.
39. A. Strasser, L. O'Connor, and V.M. Dixit. Apoptosis signaling. *Annu. Rev. Biochem.*, 69:217–245, 2000.
40. H. Wang. Proving theorems by pattern recognition ii. *Bell Systems Technical Journal*, 40:1–41, 1961.
41. E. Winfree. Self-healing tile sets. Draft, 2005.
42. 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.
43. 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.
44. 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.