On the Complexity of Graph Self-assembly in Accretive Systems

Stanislav Angelov, Sanjeev Khanna, and Mirkó Visontai

Department of Computer and Information Science
School of Engineering and Applied Sciences
University of Pennsylvania, Philadelphia, PA 19104, USA
{angelov, sanjeev, mirko}@cis.upenn.edu

Abstract. We study the complexity of the Accretive Graph Assembly Problem (AGAP). An instance of AGAP consists of an edge-weighted graph G, a seed vertex in G, and a temperature τ. The goal is to determine if there is a sequence of vertex additions which constructs G starting from the seed. The edge weights model the forces of attraction and repulsion, and determine which vertices can be added to a partially assembled graph at the given temperature.

Our first result is that AGAP is NP-complete even on degree 3 planar graphs when edges have only two different types of weights. This resolves the complexity of AGAP in the sense that the problem is polytime solvable when either the degree is bounded by 2 or the number of distinct edge weights is one, and is NP-complete otherwise. Our second result is a dichotomy theorem that completely characterizes the complexity of AGAP on degree 3 bounded graphs with two distinct weights: w_p, w_n. We give a simple system of linear constraints on w_p, w_n, and τ that determines whether the problem is NP-complete or is polytime solvable. In the process of establishing this dichotomy, we give the first polytime algorithm to solve a non-trivial class of AGAP. Finally, we consider the optimization version of AGAP where the goal is to realize a largest-possible subgraph of the given input graph. We show that even on constructible graphs of degree at most 3, it is NP-hard to realize a $(1/n^{1-\epsilon})$-fraction of the input graph for any $\epsilon > 0$; here n denotes the number of vertices in G.

1 Introduction

Self-assembly is a process in which small objects interact autonomously with each other to form intricate complexes. The self-assembly approach is particularly appealing for constructing molecular scale objects with nano-scale features [1]. Examples of its application and practical modeling can be found in [2,3,4,5,6,7,8,9].

Based on the Wang Tiling Models [10], Rothemund and Winfree [11] proposed the *Tile Assembly Model* to formalize and facilitate theoretical study of the self-assembly process. In this model, DNA tiles are abstracted as oriented squares, where each side has a glue type and (non-negative) strength. An assembly starts from a designated *seed* tile and can be augmented by a tile if the sides of the tile match the glue types of its already assembled neighbors and the total glue strength is no less than a threshold parameter τ, referred to as the *temperature* of the assembly.

Recently, Reif, Sahu, and Yin [1] proposed a generalization of the Tile Assembly Model, i.e., assembly on two-dimensional grids, to one on general graphs, named the *Accretive Graph Self-Assembly Model*. The accretive graph assembly is a *sequential* process where a weighted graph is assembled one vertex at a time starting from a seed vertex. The weight of each positive (resp. negative) edge specifies the magnitude of attraction (resp. repulsion) between the incident vertices. Again, a vertex is added to the assembly if the net attraction minus repulsion of the built neighbors is at least τ. *Accretive* here suggests the monotone property of the process, i.e., an added vertex cannot be removed later, in contrast to the *Self-Destructive Graph Assembly Model* [1], or the *Kinetic Tile Assembly Model* where tiles can fall off [12,13].

The Accretive Graph Self-Assembly Model addresses some of the deficiencies of the Tile Assembly Model. Namely, it models repulsion and allows the assembly of general graph structures. A central problem in this model is the *Accretive Graph Assembly Problem* (AGAP): Given a weighted graph on n vertices and a seed vertex, the problem asks for a sequence of vertex additions respecting τ that builds the graph. Among other results, it was shown in [1] that AGAP is NP-complete for graphs with maximum degree 4 and for planar graphs (PAGAP) with maximum degree 5. The authors in [1] posed several natural open problems related to AGAP which we address in this paper. The first question was to determine the precise degree bound for which AGAP and PAGAP change in complexity from polytime solvable to NP-complete, and the second one was to determine the difficulty of the optimization versions of these problems.

Our Results and Techniques. The complexity of a graph assembly system can be measured by the degree of the underlying graph G and the number of possible weights an edge can take. Similarly to the number of different tiles in the Tile Assembly Model, here we can bound the number of different vertex *types*, where a type is determined by the weights of the edges incident to a vertex. A natural question is under what conditions we can solve AGAP in polytime and what is the smallest complexity for which we can show the problem to be NP-complete. Our main results settle open problems posed in [1] and are as follows:

– We show that PAGAP (and hence AGAP) is NP-complete even if the maximum degree of the input graph is 3 and edges can take only two different weights. This result is tight in the sense that AGAP is polytime solvable if either the maximum degree is bounded by two or all edges have identical weights.
– We prove a dichotomy theorem that completely characterizes the complexity of AGAP on degree 3 bounded graphs with two distinct weights: w_p, w_n. We give a simple system of linear constraints on w_p, w_n, and τ that determines whether the problem is NP-complete or polytime solvable. In the process of establishing this dichotomy, we give a polytime algorithm to solve a non-trivial class of AGAP instances.

– We show that MAX AGAP, the optimization version of AGAP, is hard to approximate within a factor of $O(n^{1-\epsilon})$ for any $\epsilon > 0$ even if the degree of the input graph is 3; here n denotes the number of vertices of the underlying graph G. When the graph edges are restricted to only two weights, we show the same hardness of approximation for degree 5 graphs via a novel reduction from the directed Hamiltonian path problem on cubic (degree 3) graphs. The results hold even if the seed vertex is not part of the input.

Our technique for showing NP-hardness extends the reduction from P3SAT shown in [1] with a modular design using gadgets. Note that these gadgets might be easy to understand but are hard to find. We also show that some of the NP-hardness results can be obtained independently by reduction from the Hamiltonian path problem [14]. Our polytime algorithm for AGAP arises by a reduction to a problem called the Rainbow Spanning Tree Problem which is known to be solvable by using a result from matroid theory [15]. The hardness of approximation relies on the NP-hardness of the underlying problem combined with additional gadget constructions.

Related Work. Much of the theoretical work on self-assembly to date has focused on analyzing the complexity of the original Tile Assembly Model. Adleman et al. [16] showed that determining the minimum number of distinct tiles required to produce a given shape is NP-complete in general, and polytime solvable for trees and squares. The authors also gave an $O(\log n)$-approximation algorithm for determining the relative concentration of tile types that achieved optimal assembly time in *partial order systems*. In the case of $n \times n$ squares, an optimal assembly requiring $\Theta(n)$ time and $\Theta(\frac{\log n}{\log \log n})$ tile types was described in [11,17] based on simulation of binary counters. Extensions to the Tile Assembly Model include consideration of flexible glue-strengths and temperature programming [18,19,20], fault tolerance and self-correction [12,13,21,22,23,24,25,26], patterning (of components) and self-replication [4,6,27,28,29,30].

Among the first works to study the self-assembly process on general graphs are [31,32,33,34,35,36]. It was shown that 3SAT and 3-Vertex-Colorability can be solved by self-assembly of DNA graphs using a constant number of laboratory steps [31,32]. A generalization of the Tile Assembly Model, where *flexible* tiles may connect to more than 4 tiles in a not necessarily planar arrangement, was investigated in [33]. Graph grammars were used to model self-assembly on planar graphs [34,35]. Experiments on construction of non-regular graphs were presented in [36].

Organization. We begin by formally describing the AGAP problem and providing the necessary notations and definitions. Building on the ideas in [1], in Section 3 we show that AGAP is NP-complete on degree 4 planar graphs. In Section 4 we introduce new types of constructions showing that AGAP is NP-complete even on degree 3 planar graphs with two distinct edge weights. In Section 5 we show our hardness of approximation results for AGAP. We summarize our results and discuss some open problems in Section 6.

2 Preliminaries

2.1 Model and Problem Statements

We adopt the *Accretive Graph Self-Assembly Model* introduced in [1]. A graph assembly system is a quadruple $\langle G, v_s, w, \tau \rangle$, where $G = (V, E)$ is undirected weighted graph, $v_s \in V$ is a seed vertex, w is a weight function: $w : E \to \mathbb{Z}$, and $\tau \in \mathbb{N}$ is the temperature of the assembly. Here the weight of an edge represents the strength of attraction between adjacent vertices if positive, and their repulsion if negative. An analogue of the weight function in the Tile Assembly Model [11] is the glue function (cf. glue strength is non-negative).

The self-assembly process in the Accretive Graph Self-Assembly Model proceeds as follows. The graph G serves as a template of construction and initially only the seed vertex v_s of G is built. For a vertex v, let $\Gamma(v)$ represent the set of neighbors of v in G that are already built. A new vertex v of G can be attached to the construction if and only if $\sum_{u \in \Gamma(v)} w(u, v) \geq \tau$, i.e., the sum of the weights from v to its already built neighbors is at least equal to the temperature of assembly. The assembly is *sequential*, i.e., vertices are built one at a time, and *accretive*, i.e., once a vertex is built it cannot be detached from the construction. We will use $u \prec v$ to denote that vertex v is built after vertex u.

We consider the following problems:

Definition 1 (Accretive Graph Assembly Problem (`AGAP`)). *Given an accretive graph assembly system $\langle G, v_s, w, \tau \rangle$, determine if G is* sequentially constructible *(in short,* constructible*) starting from the seed vertex v_s, and provide a feasible order of construction, $\pi : v_s = v_{\pi_1} \prec v_{\pi_2} \prec \ldots \prec v_{\pi_n}$, if one exists.*

Definition 2 (Planar Accretive Graph Assembly Problem (`PAGAP`)). *The* `AGAP` *problem restricted to planar graphs.*

We also consider the following restrictions of `AGAP` (`PAGAP`). The k-`WEIGHT AGAP` (k-`WEIGHT PAGAP`) is a special instance of `AGAP` (`PAGAP`) such that, there are at most k different edge weights in G. When the degree of G is restricted to d, we refer to the problem as d-`DEGREE AGAP` (d-`DEGREE PAGAP`).

Since `AGAP` is NP-complete in general [1], it is natural to consider polytime approximation algorithms that seek to build a largest subset of vertices of the input graph.

Definition 3 (Maximum `AGAP` (`MAX AGAP`)). *Given an instance of* `AGAP` *on a graph G, find a largest subgraph of G which is constructible starting from the seed vertex, and provide an order of construction.*

An *α-approximation* algorithm for `MAX AGAP` is a polytime algorithm that on any given input instance G computes a constructible subgraph of G, say H, such that $|H| \geq |H^*|/\alpha$, where H^* is a largest constructible subgraph of G.

2.2 Background Results and Definitions

The two propositions below characterize some simple cases where `AGAP` is polytime solvable.

Proposition 1. [1] `AGAP` *with only positive edge weights can be solved in* $O(|V|+|E|)$ *time.*

Proposition 2. `2-DEGREE AGAP` *can be solved in* $O(|V|+|E|)$ *time.*

We, therefore, focus on graphs with maximum degree at least 3 and with at least one edge with negative weight. In order G to be constructible there must be an edge of weight at least $\tau \geq 0$, otherwise the first vertex other than the seed cannot be built. Hence, we consider graphs with at least two different weights.

We will only show NP-hardness in the NP-completeness proofs for `PAGAP` since `(P)AGAP` is easily shown to be in NP: given an ordering of the vertices, it can be verified in polynomial time if it is feasible.

Planar 3SAT. In our results we will mostly use a reduction from planar `3SAT` (`P3SAT`), similar to [1]. Lichtenstein [37] proved that `P3SAT`, i.e., `3SAT` with the restriction that the *identifying graph* is planar, remains NP-complete. The identifying graph of a `3SAT` formula ϕ is a graph $G = (V, E)$ where vertices correspond to literals and clauses; and there is an edge between a literal vertex and a clause vertex if and only if the literal participates in the clause. Also, there is an edge between every literal and its complement. Middleton [38] showed that deciding the satisfiability of a `P3SAT` formula with an identifying graph (see Fig. 1) obeying the following restrictions is still NP-complete:

(1) There is a cyclic path, called the *loop*, that can be drawn in the plane such that it passes between all pairs of complementary literals, but does not intersect any other edges of G.
(2) The boolean formula contains only clauses in which the literals are either all positive or all negative.
(3) The graph G can be arranged so that interior (resp. exterior) clauses have only positive (resp. negative) literals.
(4) Let $C(\ell)$ denote the set of clauses in which a literal ℓ participates, then $|C(\ell)| \leq 2$ for all ℓ in ϕ.

The dashed circle in Fig. 1 corresponds to the loop described above, which we assume to be *directed* and denote by L. The loop provides a natural (cyclic) ordering of the variables, e.g., $L = x, y, z, w, x$. For variables u and v we will use $uv \in L$ to denote that v follows u in L, e.g., $xy \in L$, but $xz \notin L$.

3 `4-DEGREE PAGAP` Is NP-Complete

The construction in this section is similar to that of [1] but it reduces the degree of the resulting graph from 5 to 4. We start our reduction from a `P3SAT` formula ϕ and its identifying graph. For every variable x and its negation $\bar{x}$, we replace the edge $(x, \bar{x})$ in the graph (Fig. 1) with the gadget depicted in Fig. 2. For x and y, $xy \in L$, we connect the corresponding gadgets with edge (t_x, s_y) with weight $\tau = 2$. This gadget ensures two vital properties. First, along the loop L we can build all vertices corresponding to literals which are set to `TRUE`, and

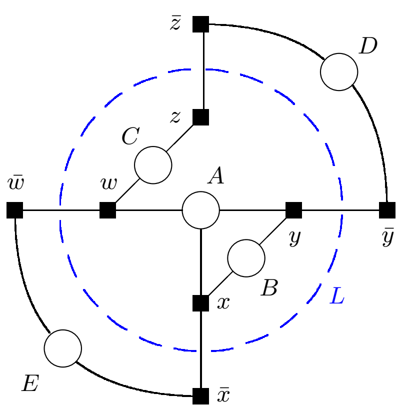


Fig. 1. The identifying graph for the formula $A \wedge B \wedge C \wedge D \wedge E = (x \vee y \vee w) \wedge (x \vee y) \wedge (w \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{w} \vee \bar{x})$

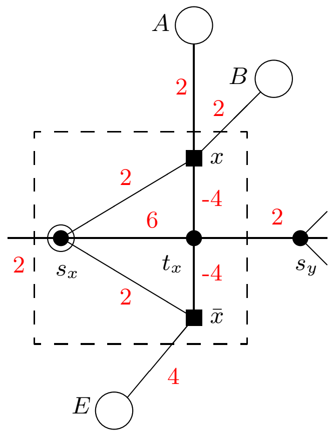


Fig. 2. Gadget for degree 4 planar graphs and $\tau = 2$. Here, s_x is the seed vertex.

therefore all vertices corresponding to clauses. Secondly, we can complete all the remaining vertices afterwards. On the other hand, if a vertex corresponding to a literal is built then we need to build t_x before we build the complementary literal. Now, the edges on the complementary literal are such, that we can only build it if all of its adjacent clauses are already built. This corresponds to the fact, that if we set x to TRUE, the formula is only satisfiable if the clauses in $C(\bar{x})$ can be satisfied independently of $\bar{x}$. We now describe the construction in detail.

3.1 The Gadget Construction

Figure 2 shows the gadget replacing $(x, \bar{x})$ in the identifying graph (for $\tau = 2$), where x and $\bar{x}$ participate in clauses A and B, and E, respectively. Formally, we construct the gadget for a variable x as follows. We use additional nodes s_x and t_x and introduce new edges with the following weights: $w(s_x, t_x) = 3\tau$, $w(x, s_x) = w(\bar{x}, s_x) = \tau$, $w(x, t_x) = w(\bar{x}, t_x) = -2\tau$ and $w(x, c) = 2\tau/|C(x)|$ for all clauses $c \in C(x)$, and $w(\bar{x}, c) = 2\tau/|C(\bar{x})|$ for all clauses $c \in C(\bar{x})$. (Recall that $|C(\ell)| \in \{1, 2\}$ for any literal ℓ.) Also, for $xy \in L$, we add an edge (t_x, s_y) with weight τ to connect to the gadget replacing $(y, \bar{y})$.

The following theorem shows that ϕ is satisfiable if and only if there is an ordering to assemble G.

Theorem 1. 4-DEGREE PAGAP *is* NP*-complete.*

Proof. For the first part, assume there is a satisfying assignment of the underlying formula. Consider the obtained graph G. We show that there is an ordering of vertices in G in which every vertex can be built. Starting from the seed vertex we construct the literals x or $\bar{x}$ depending on whose value is set to TRUE in σ following the loop L. After x or $\bar{x}$ is built, we construct t_x and we proceed to the next variable y in the loop by building s_y. Since σ is a satisfying assignment, each vertex corresponding to a clause is adjacent to a vertex which is built (each clause has a variable which is set to `TRUE`) and the edge weights connecting them are $\geq \tau$. At the final step we can build the literals ℓ which are set to `FALSE` in σ. We have two cases based on the cardinality of $C(\ell)$. If $|C(\ell)| = 1$, the contribution of ℓ's neighbors is $-2\tau + \tau + 2\tau = \tau$ and similarly if $|C(\ell)| = 2$ the contribution is $-2\tau + \tau + \tau + \tau = \tau$.

For the second part, consider an ordering in which we complete all the vertices of the graph G. Look at the following assignment: set x to TRUE if x is built before $\bar{x}$ and set x to FALSE otherwise. We claim this is a satisfying assignment to ϕ. We prove by contradiction. Assume there is a clause $A = x \vee y \vee z$ which is not satisfied, hence $x = y = z =$ `FALSE`. Thus, $\bar{x} \prec x$, $\bar{y} \prec y$ and $\bar{z} \prec z$. *W.l.o.g.*, $x \prec y \prec z$ in this ordering. The clause A is adjacent only to x, y, z and thus $x \prec A$. But, due to the construction x can only be built after $C(x)$ is built (since it must be the case that $\bar{x} \prec t_x \prec x$), implying $A \prec x$, which is clearly a contradiction. □

3.2 An Alternative Approach

We can show a stronger version of the above theorem, via a reduction from the directed Hamiltonian path problem in cubic graphs [14]. This new approach only uses 3 distinct weights, as opposed to the 4 weights in the preceding construction. In addition to that, every constructible instance has a *stable* order of construction, i.e., at every step of the construction, each built vertex has a net attraction at least τ. This is in contrast to the previous reduction using gadgets shown in Fig. 2, where no stable order of construction exists since the sum of the weights of edges incident on t_x is less than τ.

Theorem 2. `4-DEGREE 2-WEIGHT PAGAP` *is* NP-*complete.*

4 The Complexity of 3-`DEGREE` 2-`WEIGHT AGAP` (`PAGAP`)

In this section, we prove a dichotomy theorem that completely characterizes the complexity of `AGAP` on degree 3 bounded graphs with two distinct weights w_p and w_n. We assume *w.l.o.g.* that $w_p \geq \tau$ and $w_n < 0$ since the case when both weights are positive is trivially solvable by Proposition 1, and the case when $\max\{w_p, w_n\} < \tau$ has no solution. We give a simple system of linear constraints on w_p, w_n and τ that determines whether the problem is NP-complete or solvable in polynomial time (see Table 1).

Theorem 3. `3-DEGREE 2-WEIGHT (P)AGAP` *with weights* w_p *and* w_n *is* NP-*complete if and only if* $w_p + 2w_n < \tau$ *and* $w_p + w_n \geq \tau$*; otherwise it is solvable in polynomial time.*

We first improve on the NP-completeness result for `4-DEGREE PAGAP` by showing that `3-DEGREE 2-WEIGHT PAGAP` is NP-complete. Our construction is related

Table 1. Complexity of `3-DEGREE 2-WEIGHT AGAP` with weights $w_p \geq \tau$ and $w_n < 0$

$w_p + 2w_n \geq \tau$	$w_p + w_n \geq \tau$	Results
`TRUE`	`TRUE`	Polytime solvable (Lemma 2)
`FALSE`	`TRUE`	NP-complete (planar graphs, Lemma 1)
`FALSE`	`FALSE`	Polytime solvable (Lemma 3 and Lemma 4)

to the degree 4 case but due to the imposed restrictions, it requires careful composition of more sophisticated gadgets. We use two gadgets, the *direction* and *choice* gadgets, depicted in Figs. 3(a) and 3(b), which are put together as shown in Fig. 3(c). The resulting gadget satisfies properties similar to the properties of the gadget in the degree 4 case (Fig. 2). We then give polynomial time algorithms for the remaining cases of `3-DEGREE 2-WEIGHT AGAP`.

4.1 `3-DEGREE 2-WEIGHT PAGAP` is NP-Complete

To show NP-hardness of `3-DEGREE 2-WEIGHT PAGAP` we follow closely the reduction of Section 3. Because of the restriction on the number of distinct edge weights we use different gadgets as building blocks. Their careful composition, however, preserves the desired properties of the above analysis. For ease of presentation we fix edge weights to be $w_p = 3$ and $w_n = -1$ at temperature $\tau = 2$. We note that the construction works in general for any $\langle w_n, w_p, \tau \rangle$ satisfying $w_p + w_n \geq \tau$ and $w_p + 2w_n < \tau$. In other words, building a single neighbor connected with negative edge to a vertex does not by itself make the vertex not constructible (infeasible), but building two such neighbors makes it infeasible. We now describe the gadgets in detail.

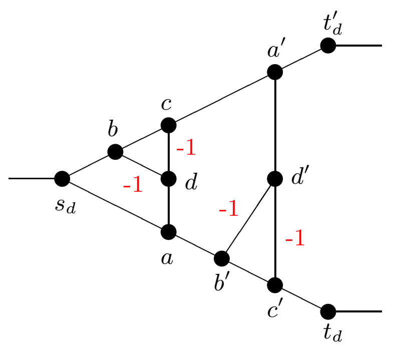
Direction gadget. The properties of the direction gadget shown in Fig. 3(a) are as follows:

- If s_d is built, we can complete the gadget: $s_d \prec a \prec d \prec b \prec c \prec a' \prec d' \prec b' \prec c' \prec \{t_d, t'_d\}$
- If t_d and t'_d are built, we can complete the gadget: $\{t_d, t'_d\} \prec a' \prec d' \prec c' \prec b' \prec a \prec d \prec c \prec b \prec s_d$
- If only t_d or t'_d are built, but not both, we cannot build s_d via the gadget unless we make d' or d infeasible. Observe that if, say, t_d is built the only way to reach s_d is via $t_d \prec c' \prec b' \prec a \prec s_d$ but this will make d' infeasible (d' will have two built neighbors contributing -1 each).

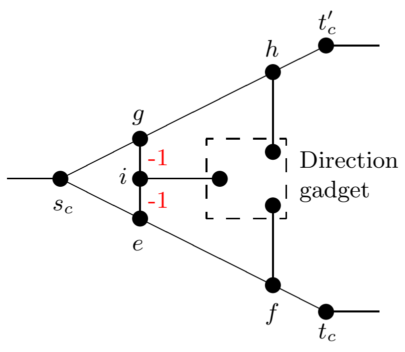
Intuitively, we will use the direction gadget to connect a literal ℓ to the clauses $C(\ell)$ for $|C(\ell)| = 2$. The gadget ensures that if ℓ is built then we can build $C(\ell)$, and if both clauses are built then we can build ℓ.

Choice gadget. The properties of the choice gadget shown in Fig. 3(b) are as follows:

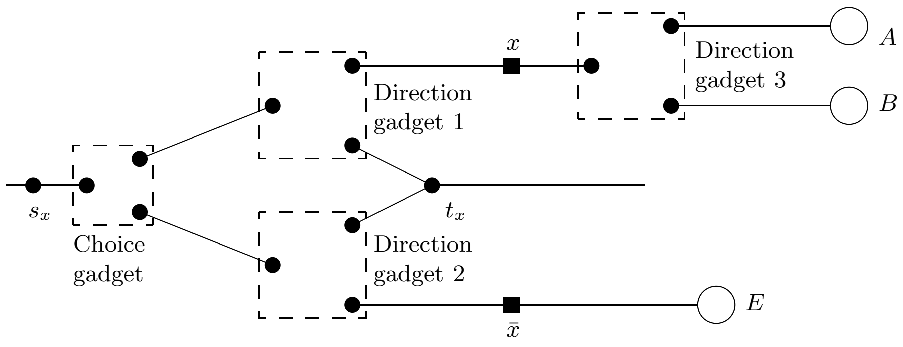
- If s_c is built, we can build either t_c or t'_c but not both via the gadget without making i infeasible.



(a) Direction gadget



(b) Choice gadget



(c) Combined gadget

Fig. 3. Gadgets for 3-DEGREE 2-WEIGHT PAGAP with $w_p = 3$, $w_n = -1$, and temperature $\tau = 2$. Edges without annotation have weight 3.

- If only t_c (resp. t'_c) is built, we cannot build t'_c (resp. t_c) via the gadget without making i infeasible.
- If t_c, t'_c and only one of e and g are built, we can complete the gadget, i.e., if we used the gadget to make a choice to build t_c (or t'_c) from s_c we can complete it once t'_c (or t_c) is built independently.

In the analysis in Section 3, it is argued that given a satisfying assignment of the boolean formula used in the reduction, we can (virtually) walk the loop L and build x or $\bar{x}$ (but not both) depending on which literal is set to `TRUE` in the assignment. The choice gadget is used to obtain this property.

Putting the gadgets together. We compose the direction and choice gadgets to obtain a gadget (Fig. 3(c)) equivalent to the one showed in Fig. 2, decreasing the maximum degree of the resulting graph to 3. Again, we use the gadget to replace the $(x, \bar{x})$ edges in the identifying graph. We connect the gadgets corresponding to x and y, $xy \in L$, with an edge (t_x, s_y) with weight 3. The following properties hold:

– Starting from s_x we can build x or $\bar{x}$ but not both. This property ensures that if there is a satisfying assignment, we can complete G: suppose we build x, then we can build all the clauses $C(x)$ in which x participates, build t_x and continue to s_y of the next variable y in L.
– If x (resp. $\bar{x}$) is built via the gadget, the only way $\bar{x}$ (resp. x) can be built is by building first the clauses in which it participates. This ensures that if the graph is built, the corresponding ϕ formula is satisfiable. Again, we use the following satisfying assignment: x is `TRUE` if and only if $x \prec \bar{x}$.

Using analogous analysis to the degree 4 case, we obtain the following lemma.

Lemma 1. `3-DEGREE 2-WEIGHT PAGAP` *such that* $w_p+2w_n < \tau$ *and* $w_p+w_n \geq \tau$ *is* NP*-complete.*

4.2 Polynomial Time Algorithms for 3-`DEGREE` 2-`WEIGHT AGAP`

We now give polynomial time algorithms to solve `3-DEGREE 2-WEIGHT AGAP` when the weights w_p and w_n are such that either $w_p + 2w_n \geq \tau$ (see Lemma 2) or $w_p + w_n < \tau$. The latter case is subdivided into two sub-cases depending on the relation between $2w_p + w_n$ and τ (see Lemmas 3 and 4).

Lemma 2. `3-DEGREE 2-WEIGHT AGAP` *such that* $w_p + 2w_n \geq \tau$ *can be solved in* $O(|V| + |E|)$ *time.*

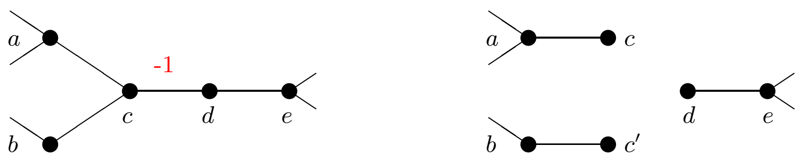
Proof. Note, $w_p+2w_n \geq \tau$ implies negative edges cannot make a vertex infeasible as long as it is reachable through positive edges from the seed vertex. We can therefore use Proposition 1 to solve the problem on the graph induced by the positive edges (negative edges neither help nor obstruct the construction). □

Lemma 3. `3-DEGREE 2-WEIGHT AGAP` *such that* $2w_p + w_n < \tau$ *is solvable in* $O(|V| + |E|)$ *time.*

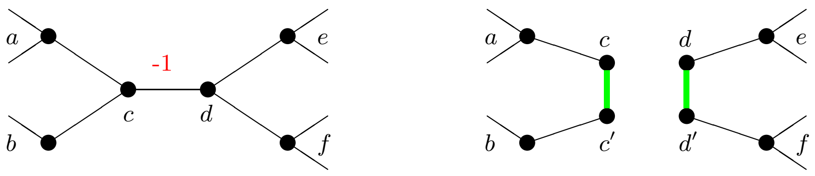
Proof. The condition $2w_p + w_n < \tau$ implies the graph cannot be built if there is a negative edge. For contradiction, assume there is a negative edge (u, v) and the graph can be built. *W.l.o.g.* assume that in a feasible ordering u is built before v. Then, by the choice of weights, v cannot be built; a contradiction. □

We prove the remaining case (Lemma 4) in two steps. We first show a feasibility-preserving transformation that removes any negative edge (u, v) such that either u or v have a single positive edge incident on it. The resulting graph is such that every vertex has more positive than negative edges. We then show that the problem can be viewed as a special case of the Rainbow Spanning Tree Problem (see Definition 4) where at most two edges have the same color.

Lemma 4. `3-DEGREE 2-WEIGHT AGAP` *such that* $w_p+w_n < \tau$ *and* $2w_p+w_n \geq \tau$ *can be solved in polynomial time.*



(a) Removing edges with negative weight and 3 positive edge neighbors.



(b) Vertices c and d are replaced by edges (c, c') and (d, d') of the same unique color.

Fig. 4. Graph transformations removing negative weight edges

Claim. An instance of `3-DEGREE 2-WEIGHT AGAP` satisfying the conditions of Lemma 4 can be reduced to an instance where each vertex has more positive edges than negative edges incident on it.

Proof. W.l.o.g. assume that each vertex is adjacent to at least one positive edge, otherwise this vertex cannot be built. Similarly to Lemma 3, since $w_p + w_n < \tau$, we can argue that for a negative edge, at least one of its endpoints should be adjacent to two positive edges. Now consider a negative edge (c, d) where d has only one positive edge (d might have another negative edge). It follows that d must be built before c in order for the graph to be constructible, and c is built after its both neighbors are built. We can therefore remove such negative edges, one by one, making a copy of c, c', connecting each to only one (different) of c's neighbors as shown in Fig. 4(a). In the new graph, we can assume *w.l.o.g.* that $d \prec \{c, c'\}$ since c and c' are not used to build other vertices. Now, it is not hard to see the if and only if correspondence between the two instances. □

We next consider instances where each vertex has more positive edges than negative edges and reduce the problem to the following combinatorial problem.

Definition 4 (Rainbow Spanning Tree Problem). *Given a graph G with colors on its edges, is there a spanning tree of G that has no two edges with the same color?*

The Rainbow Spanning Tree Problem can be solved in polynomial time since it can be formulated as the problem of finding the maximum independent set of the intersection of two matroids [15].

Claim. Given `3-DEGREE 2-WEIGHT AGAP` on graph G where each vertex has more positive than negative edges, we can compute a graph H such that G is constructible if and only if H has a rainbow spanning tree.

Proof. We obtain H from G by performing graph transformations to remove all negative edges. For each removed edge we split its endpoints and introduce two positive edges with the same unique color (see Fig. 4(b)). All other edges are assigned unique colors. Ignoring colors, since $w_p \geq \tau$, to build a vertex it is enough to have an already built neighbor, i.e., to build all vertices we need a spanning tree. The colors enforce that if we use an edge of a given color to build a vertex, we cannot build another vertex using an edge of the same color, i.e., we need a rainbow spanning tree.

Formally, consider a negative edge (c, d) and the transformation described in Fig. 4(b). Vertices c and d have degree 3 by the claim proposition. (Note that c and d may share some of their neighbors, e.g., $a = e$.) Assume in a feasible ordering of G, $d \prec c$. From the choice of weights, it follows that $\{a, b\} \prec c$. Therefore, if G is constructible, we can construct a spanning tree in H which includes (a, c) and (b, c') but not (c, c'). Since the color of edge (c, c') appears at most twice, the obtained tree is a rainbow spanning tree of H. Conversely, consider a rainbow spanning tree of H. If say edge (d, d') is included, we can build d before c in G. It must be the case that a and b are connected with a path of distinct colors that does not include c or c'. Therefore we can defer the building of c after a and b are built in G. The claim follows. □

This concludes the proof of Lemma 4. Combining Lemmas 1, 2, 3, and 4, we obtain Theorem 3.

5 Hardness of Approximation of `MAX AGAP`

We now focus on `MAX AGAP` where the goal is to find the maximum number of vertices that can be sequentially built of a given graph assembly system starting from the specified seed vertex. Since `AGAP` is NP-complete, it is natural to study polytime approximation algorithms for `AGAP`. We show that even in very restricted settings, any non-trivial approximation of `AGAP` is hard. In particular, we show that `AGAP` is hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$. This hardness of approximation holds for degree 3 graphs with three distinct edge weights as well as for degree 5 graphs with two distinct edge weights.

An approach to show hardness of approximation for degree 3 graphs is to use the NP-hardness result of `3-DEGREE 2-WEIGHT PAGAP`. We would like to boost the hardness of the instance by attaching a large graph H to it and argue that a significant fraction of H can be constructed if and only if the original instance can be fully constructed. This, however, will not work since we can attach H (almost) only to the vertices corresponding to variables (see Fig. 3(c)) without increasing the overall degree of the graph. Moreover, we can easily construct all variables making vertex i of each choice gadget (see Fig. 3(b)) infeasible, and construct all of H regardless of whether we are given a hard instance or not to begin with. In fact, there is a 2-approximation algorithm for this case since we can account for the vertices that are made infeasible by their built neighbors.

Lemma 5. *There is a 2-approximation algorithm to* `3-DEGREE 2-WEIGHT MAX AGAP` *when* $w_p + w_n \geq \tau$ *and* $w_p + 2w_n < \tau$.

To show hardness of approximation we allow 3 distinct weights instead of 2: τ, $\tau - 1$ and -1 for any $\tau \geq 2$. We can now define gadgets, shown in Figs. 5(a) and 5(b), equivalent to the direction and choice gadgets (Figs. 3(a) and 3(b)) such that choice and direction are enforced directly, and not by making some of the vertices infeasible. We can show in this setting that the underlying formula in the `P3SAT` instance is satisfiable if and only if we can build all vertices corresponding to literals in the respective instance of `PAGAP`. Furthermore, we can build these vertices if and only if the corresponding `PAGAP` can be built.

We now proceed to establish our hardness result. Fix a parameter $\epsilon > 0$. We consider the following construction which is a composition of two graphs G and H. Consider a `3-DEGREE 3-WEIGHT PAGAP` instance with the graph $G = (V, E)$ obtained from a `P3SAT` formula with n variables. We have $|V| = O(n)$ since each variable participates in constant number of clauses and its corresponding gadget is of constant size. The graph $H = (V', E')$ consists of $n^{2/\epsilon}$ chained copies of the *cooperation* gadget shown in Fig. 5(c). In H, the vertex t_i of the jth copy of the cooperation gadget is connected to the vertex s_i of the $(j+1)$th copy with an edge of weight τ. To compose G and H we connect the ith variable (resp. its negation) to s_{2i-1} (resp. s_{2i}) of the first copy of the cooperation gadget as shown in Fig. 5(d). Note that the resulting graph has degree 3 since literals have degree 2 (see Fig. 3(c)). However, the resulting graph is no longer planar.

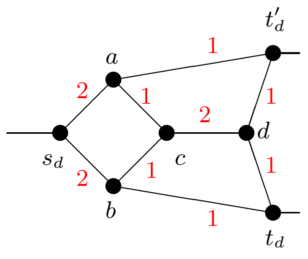
The cooperation gadget has the property that if all s_i vertices are built, we can build all t_i vertices. However, if only m of the s_i's are built, $m < 2n$, we can build at most $m - 1$ of the t_i's. Therefore, if G is constructible we can build all of H, otherwise we can build only $O(n + n^2) = O(n^2)$ vertices of G and H.

The total number of vertices in $V \cup V'$ is $N = \Theta(n^{1+2/\epsilon})$ since each of the $n^{2/\epsilon}$ copies of the cooperation gadget has $O(n)$ vertices. It follows that an $N^{1-\epsilon}$-approximation polytime algorithm for the optimization version of `3-DEGREE 3-WEIGHT AGAP` can be used to decide whether or not the instance is constructible, and therefore decide `P3SAT`. Hence we obtain the following theorem.

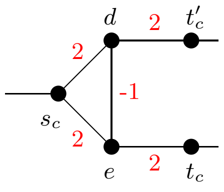
Theorem 4. `3-DEGREE 3-WEIGHT MAX AGAP` *is* NP*-hard to approximate within a factor of* $O(n^{1-\epsilon})$ *for any* $\epsilon > 0$*, where* n *denotes the number of vertices in* G.

If we restrict the number of weights to two, we can show a similar result by allowing the maximum degree to be 5. We replace each edge (x, y) with weight 1 (Figs. 5(a) and 5(c)) by a "triangle", adding a vertex z and using weights $w(x, z) = w(z, y) = 2$ and $w(x, y) = -1$. Note that the degree of the construction is increased by at most 2, hence we have the following corollary.

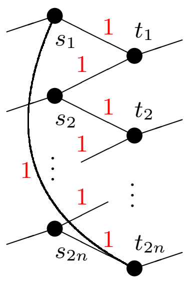
Corollary 1. `5-DEGREE 2-WEIGHT MAX AGAP` *is* NP*-hard to approximate within a factor of* $O(n^{1-\epsilon})$ *for any* $\epsilon > 0$*, where* n *denotes the number of vertices in* G.



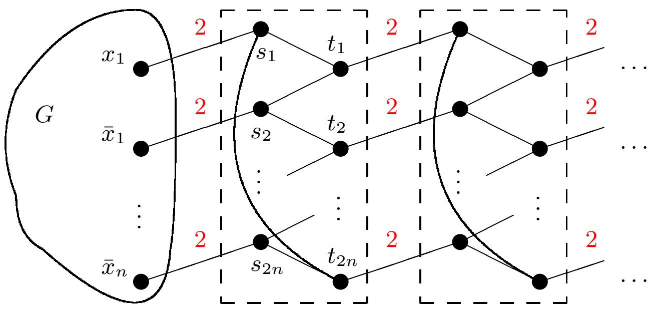
(a) Direction gadget



(b) Choice gadget



(c) Cooperation gadget



(d) Cooperation gadgets (in the dashed boxes) and their composition with the graph G

Fig. 5. Gadgets for degree 3 planar graphs with three possible edge weights for $\tau = 2$

We note that the NP-hardness and hardness of approximation results hold even when the algorithm is allowed to choose any seed vertex of its choice. We also note that the results of this section hold even if we require that each instance has a stable order of construction, as in Section 3.2.

6 Conclusion

In this paper we resolved the complexity of `AGAP` and `PAGAP` by showing that `3-DEGREE PAGAP` and hence `3-DEGREE AGAP` is NP-complete even with two edge weights. We proved a dichotomy theorem completely describing the complexity of `3-DEGREE 2-WEIGHT AGAP`, providing a simple system of linear constraints on the weights and the temperature of assembly to determine whether the problem is NP-complete or polytime solvable. The solution for the polytime case provides the first non-trivial algorithm for `AGAP`. Finally, we proved that both `3-DEGREE 3-WEIGHT AGAP` and `5-DEGREE 2-WEIGHT AGAP` are hard to approximate within a factor of $O(n^{1-\epsilon})$. These negative results motivate the question whether there exist better approximation algorithms for the lower degree cases as well as for planar graphs. Approximating `PAGAP` is especially interesting since it captures the essential geometry of 2D physical systems.

Acknowledgments. This work was supported in part by an NSF Career Award CCR-0093117. The authors would like to thank Péter Biró for helpful discussions. We also thank the anonymous reviewers for their useful suggestions and comments.

References

1. Reif, J.H., Sahu, S., Yin, P.: Complexity of graph self-assembly in accretive systems and self-destructible systems. In: DNA Computing. (2005) 101–112
2. Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C.: Design and self-assembly of two-dimensional DNA crystals. Nature **394** (1998) 539–544
3. Rothemund, P.: Using lateral capillary forces to compute by self-assembly. Proc. Nat. Acad. Sci. U.S.A. **97** (2000) 984–989
4. LaBean, T.H., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J.H., Seeman, N.C.: Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. J. Amer. Chem. Soc. **122** (2000) 1848–1860
5. Yan, H., LaBean, T.H., Feng, L., Reif, J.H.: Directed nucleation assembly of DNA tile complexes for barcode-patterned lattices. Proc. Nat. Acad. Sci. U.S.A. **100** (2003) 8103–8108
6. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology **2** (2004) 2041–2053
7. Chelyapov, N., Brun, Y., Gopalkrishnan, M., Reishus, D., Shaw, B., Adleman, L.M.: DNA triangles and self-assembled hexagonal tilings. J. Amer. Chem. Soc. **126** (2004) 13924–13925
8. He, Y., Chen, Y., Liu, H., Ribbe, A.E., Mao, C.: Self-assembly of hexagonal DNA two-dimensional (2D) arrays. J. Amer. Chem. Soc. **127** (2005) 12202–12203
9. Malo, J., Mitchell, J.C., Vénien-Bryan, C., Harris, J.R., Wille, H., Sherratt, D.J., Turberfield, A.J.: Engineering a 2D protein-DNA crystal. Angewandte Chemie International Edition **44** (2005) 3057–3061
10. Wang, H.: Proving theorems by pattern recognition II. Bell Systems Technical Journal **40** (1961) 1–41
11. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares (extended abstract). In: STOC. (2000) 459–468
12. Winfree, E., Bekbolatov, R.: Proofreading tile sets: Error correction for algorithmic self-assembly. In: DNA Based Computers. (2003) 126–144
13. Chen, H.L., Goel, A.: Error free self-assembly using error prone tiles. In: DNA Computing. (2004) 62–75
14. Plesník, J.: The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. Inform. Process. Lett. **8** (1979) 199–201
15. Broersma, H., Li, X.: Spanning trees with many or few colors in edge-colored graphs. Discuss. Math. Graph Theory **17** (1997) 259–269
16. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.D.A., Kempe, D., de Espanés, P.M., Rothemund, P.W.K.: Combinatorial optimization problems in self-assembly. In: STOC. (2002) 23–32
17. Adleman, L.M., Cheng, Q., Goel, A., Huang, M.D.A.: Running time and program size for self-assembled squares. In: STOC. (2001) 740–748
18. Aggarwal, G., Goldwasser, M., Kao, M.Y., Schweller, R.T.: Complexities for generalized models of self-assembly. In: SODA. (2004) 880–889

19. Sahu, S., Yin, P., Reif, J.H.: A self-assembly model of DNA tiles with time dependent glue strength. In: DNA Computing. (2005) 113–124
20. Kao, M.Y., Schweller, R.: Reducing tile complexity for self-assembly through temperature programming. In: SODA. (2006) 571–580
21. Chen, H.L., Cheng, Q., Goel, A., Huang, M.D.A., de Espanés, P.M.: Invadable self-assembly: combining robustness with efficiency. In: SODA. (2004) 890–899
22. Fujibayashi, K., Murata, S.: A method of error suppression for self-assembling DNA tiles. In: DNA Computing. (2004) 113–127
23. Reif, J.H., Sahu, S., Yin, P.: Compact error-resilient computational DNA tiling assemblies. In: DNA Computing. (2004) 293–307
24. Schulman, R., Winfree, E.: Programmable control of nucleation for algorithmic self-assembly. In: DNA Computing. (2004) 319–328
25. Soloveichik, D., Winfree, E.: Complexity of self-assembled shapes. In: DNA Computing. (2004) 344–354
26. Soloveichik, D., Winfree, E.: Complexity of compact proofreading for self-assembled patterns. In: DNA Computing. (2005) 125–135
27. Lagoudakis, M.G., LaBean, T.H.: 2D DNA self-assembly for satisfiability. In: DNA Based Computers. (1999) 139–152
28. Cook, M., Rothemund, P.W.K., Winfree, E.: Self-assembled circuit patterns. In: DNA Based Computers. (2003) 91–107
29. Schulman, R., Lee, S., Papadakis, N., Winfree, E.: One dimensional boundaries for DNA tile self-assembly. In: DNA Based Computers. (2003) 108–126
30. Barish, R.D., Rothemund, P.W.K., Winfree, E.: Two computational primitives for algorithmic self-assembly: Copying and counting. Nano Letters **5** (2005) 2586–2592
31. Jonoska, N., Karl, S.A., Saito, M.: Three dimensional DNA structures in computing. BioSystems **52** (1999) 143–153
32. Jonoska, N., Sa-Ardyen, P., Seeman, N.C.: Computation by self-assembly of DNA graphs. Genetic Programming and Evolvable Machines **4** (2003) 123–137
33. Jonoska, N., McColm, G.L.: A computational model for self-assembling flexible tiles. In: UC. (2005) 142–156
34. Klavins, E., Ghrist, R., Lipsky, D.: A grammatical approach to self-organizing robotic systems. IEEE Trans. Automat. Control **51** (2006) 949–962
35. Klavins, E.: Directed self-assembly using graph grammars. In: FNANO. (2004)
36. Sa-Ardyen, P., Jonoska, N., Seeman, N.C.: Self-assembling DNA graphs. Natural Computing **2** (2003) 427–438
37. Lichtenstein, D.: Planar formulae and their uses. SIAM J. Comput. **11** (1982) 329–343
38. Middleton, A.A.: Computational complexity of determining the barriers to interface motion in random systems. Phys. Rev. E **59** (1999) 2571–2577