The Complexity of Growing a Graph

George B. Mertzios[1], Othon Michail[2], George Skretas[2,3], Paul G. Spirakis[2,4], and Michail Theofilatos[2]

[1] Department of Computer Science, Durham University, UK
george.mertzios@durham.ac.uk
[2] Department of Computer Science, University of Liverpool, UK
{othon.michail,p.spirakis,michail.theofilatos}@liverpool.ac.uk
[3] Hasso Plattner Institute, University of Potsdam, Germany
georgios.skretas@hpi.de
[4] Computer Engineering & Informatics Department, University of Patras, Greece

Abstract. We study a new algorithmic process of graph growth. The process starts from a single initial vertex u_0 and operates in discrete time-steps, called *slots*. In every slot $t \geq 1$, the process updates the current graph instance to generate the next graph instance G_t. The process first sets $G_t = G_{t-1}$. Then, for every $u \in V(G_{t-1})$, it adds at most one new vertex u' to $V(G_t)$ and adds the edge uu' to $E(G_t)$ alongside any subset of the edges $\{vu' \mid v \in V(G_{t-1})$ is at distance at most $d-1$ from u in $G_{t-1}\}$, for some integer $d \geq 1$ fixed in advance. The process completes slot t after removing any (possibly empty) subset of edges from $E(G_t)$. Removed edges are called *excess edges*. G_t is the graph *grown* by the process after t slots. The goal of this paper is to investigate the algorithmic and structural properties of this process of graph growth.

Graph Growth Problem: Given a graph family F, we are asked to design a *centralized* algorithm that on any input *target graph* $G \in F$, will output such a process growing G, called a *growth schedule* for G. Additionally, the algorithm should try to minimize the total number of slots k and of excess edges ℓ used by the process.

We show that the most interesting case is when $d = 2$ and that there is a natural trade-off between k and ℓ. We begin by investigating growth schedules of $\ell = 0$ excess edges. On the positive side, we provide polynomial-time algorithms that decide whether a graph has growth schedules of $k = \log n$ or $k = n - 1$ slots. Along the way, interesting connections to cop-win graphs are being revealed. On the negative side, we establish strong hardness results for the problem of determining the minimum number of slots required to grow a graph with zero excess edges. In particular, we show that the problem (i) is NP-complete and (ii) for any $\varepsilon > 0$, cannot be approximated within $n^{\frac{1}{3}-\varepsilon}$, unless P = NP. We then move our focus to the other extreme of the (k, ℓ)-spectrum, to investigate growth schedules of (poly)logarithmic slots. We show that trees can be grown in a polylogarithmic number of slots using linearly many excess edges, while planar graphs can be grown in a logarithmic number of slots using $O(n \log n)$ excess edges. We also give lower bounds on the number of excess edges, when the number of slots is fixed to $\log n$.

Keywords: Temporal graph · cop-win graph · graph process · polynomial-time algorithm · lower bound · NP-complete · hardness result

1 Introduction

1.1 Motivation

Growth processes are found in a variety of networked systems. In nature, crystals grow from an initial nucleation or from a "seed" crystal and a process known as embryogenesis develops sophisticated multicellular organisms, by having the genetic code control tissue growth [11, 28]. In human-made systems, sensor networks are being deployed incrementally to monitor a given geographic area [12, 19], social-network groups expand by connecting with new individuals [13], DNA self-assembly automatically grows molecular shapes and patterns starting from a seed assembly [14, 31, 34], and high churn or mobility can cause substantial changes in the size and structure of computer networks [4,6]. Graph-growth processes are central in some theories of relativistic physics. For example, in dynamical schemes of causal set theory, causets develop from an initial emptiness via a tree-like birth process, represented by dynamic Hasse diagrams [9,30].

Though diverse in nature, all these are examples of systems sharing the notion of an underlying graph-growth process. In some, like crystal formation, tissue growth, and sensor deployment, the implicit graph representation is bounded-degree and embedded in Euclidean geometry. In others, like social-networks and causal set theory, the underlying graph might be free from strong geometric constraints but still be subject to other structural properties, as is the special structure of causal relationships between events in casual set theory.

Inspired by such systems, we study a high-level, graph-theoretic abstraction of network-growth processes. We do not impose any strong *a priori* constraints, like geometry, on the graph structure and restrict our attention to *centralized* algorithmic control of the graph dynamics. We do include, however, some weak conditions on the permissible dynamics, necessary for non-triviality of the model and in order to capture realistic abstract dynamics. One such condition is "locality", according to which a newly introduced vertex u' in the neighborhood of a vertex u, can only be connected to vertices within a reasonable distance $d-1$ from u. At the same time, we are interested in growth processes that are "efficient", under meaningful abstract measures of efficiency. We consider two such measures, to be formally defined later, the *time* to grow a given target graph and the number of auxiliary connections, called *excess edges*, employed to assist the growth process. For example, in cellular growth, a useful notion of time is the number of times all existing cells have divided and is usually polylogarithmic in the size of the target tissue or organism. In social networks, it is quite typical that new connections can only be revealed to an individual u' through its connection to another individual u who is already a member of a group. Later, u' can drop its connection to u but still maintain some of its connections to u's group. The dropped connection uu' can be viewed as an excess edge, whose creation and removal has an associated cost, but was nevertheless necessary for the formation of the eventual neighborhood of u'.

The present study is also motivated by recent work on dynamic graph and network models [1–3, 7, 8, 10, 15–18, 20, 21, 21, 23–27, 32, 35].

1.2 Our Approach

We study the following centralized graph-growth process. The process, starting from a single initial vertex u_0 and applying vertex-generation and edge-modification operations, grows a given target graph G. It operates in discrete time-steps, called slots. In every slot, it generates at most one new vertex u' for every existing vertex u and connects it to u. Then, for every new vertex u', it connects u' to any (possibly empty) subset of the vertices within a "local" radius around u, described by a distance parameter d, essentially representing that radius plus 1, i.e., as measured from u'. Finally, it removes any (possibly empty) subset of edges whose removal does not disconnect the graph, before moving on to the next slot. These edge-modification operations are essentially capturing, at a high level, the local dynamics present in most of the applications discussed previously. In these applications, new entities typically join a local neighborhood or a group of other entities, which then allows them to easily connect to any of the local entities. Moreover, in most of these systems, existing connections can be easily dropped by a local decision of the two endpoints of that connection. The rest of this paper exclusively focuses on $d = 2$; as formally shown in the appendix, the cases $d = 1$ and $d \geq 3$ admit simple and efficient growth processes.

It is not hard to observe that, without additional considerations, any target graph can be grown by the following straightforward process. In every slot t, the process generates a new vertex u_t which it connects to u_0 and to all neighbors of u_0. The graph grown by this process by the end of slot t, is the clique K_{t+1}, thus, any K_n is grown by it within $n - 1$ slots. As a consequence, any target graph G on n vertices can be grown by extending the above process to first grow K_n and then delete from it all edges in $E(K_n) \setminus E(G)$, by the end of the last slot. Such a clique growth process maximizes both complexity parameters that are to be minimized by the developed processes. One is the time to grow a target graph G, to be defined as the number of slots used by the process to grow G, and the other is the total number of deleted edges during the process, called excess edges. The above process always uses $n - 1$ slots and may delete up to $\Theta(n^2)$ edges for sparse graphs, such as a path graph or a planar graph.

There is an improvement of the clique process, which connects every new vertex u_t to u_0 and to exactly those neighbors v of u_0 for which vu_t is an edge of the target graph G. At the end, the process deletes those edges incident to u_0 that do not correspond to edges in G, in order to obtain G. If u_0 is chosen to represent the maximum degree, $d_{\max}$, vertex of G, then it is not hard to see that this process uses $n - 1 - d_{\max}$ excess edges, while the number of slots remains $n - 1$ as in the clique process. However, we shall show that there are (poly)logarithmic-time processes using close to linear excess edges for some of those graphs. In general, processes considered *efficient* in this work will be those using (poly)logarithmic slots and linear (or close to linear) excess edges.

The goal of this paper is to investigate the algorithmic and structural properties of such processes of graph growth, with the main focus being on studying the following problem, which we call the *Graph Growth Problem*. In this problem, a centralized algorithm is provided with a target graph G, usually from a graph family F, and non-negative integers k and ℓ as its input. The goal is for the algorithm to compute, in the form of a *growth schedule* for G, such a process growing G within at most k slots and using at most ℓ excess edges, if one exists. All algorithms we consider are polynomial-time.[5]

For an illustration of the discussion so far, consider the graph family $F_{star} = \{G \mid G \text{ is a star on } n = 2^\delta \text{ vertices}\}$ and assume that edges are activated within local distance $d = 2$. We describe a simple algorithm returning a time-optimal and linear excess-edges growth process, for any target graph $G \in F_{star}$ given as input. To keep this exposition simple, we do not give k and ℓ as input-parameters to the algorithm. The process computed by the algorithm, shall always start from $G_0 = (\{u_0\}, \emptyset)$. In every slot $t = 1, 2, \ldots, \delta$ and every vertex $u \in V(G_t)$ the process generates a new vertex u', which it connects to u. If $t > 1$ and $u \neq u_0$, it then activates the edge u_0u', which is at distance 2, and removes the edge uu'. It is easy to see that by the end of slot t, the graph grown by this process is a star on 2^t vertices centered at u_0 see Figure 1 in the appendix) . Thus, the process grows the target star graph G within $\delta = \log n$ slots. By observing that $2^t/2 - 1$ edges are removed in every slot t, it follows that a total of $\sum_{1 \leq t \leq \log n} 2^{t-1} - 1 < \sum_{1 \leq t \leq \log n} 2^t = O(n)$ excess edges are used by the process. Note that this algorithm can be easily designed to compute and return the above growth schedule for any $G \in F_{star}$ in time polynomial in the size $|\langle G \rangle|$ of any reasonable representation of G.

Note that there is a natural trade-off between the number of slots and the number of excess edges that are required to grow a target graph. That is, if we aim to minimize the number of slots (resp. of excess edges) then the number of excess edges (resp. slots) increases. To gain some insight into this trade-off, consider the example of a path graph G on n vertices $u_0, u_1, ..., u_{n-1}$, where n is even for simplicity. If we are not allowed to activate any excess edges, then the only way to grow G is to always extend the current path from its endpoints, which implies that a schedule that grows G must have at least $\frac{n}{2}$ slots. Conversely, as we shall prove (in the appendix), if the growth schedule has to finish after $\log n$ slots, then G can only be grown by activating $\Omega(n)$ excess edges.

In this paper, we mainly focus on this trade-off between the number of slots and the number of excess edges.

[5] Note that this reference to *time* is about the running time of an algorithm computing a growth schedule. But the length of the growth schedule is another representation of time: the time required by the respective growth process to grow a graph. To distinguish between the two notions of time, we will almost exclusively use the term *number of slots* to refer to the length of the growth schedule and *time* to refer to the running time of an algorithm generating the schedule.

1.3 Contribution

Section 2 presents the model and problem statement and gives two basic subprocesses that are recurrent in our growth processes.

In Section 3, we study the *zero-excess growth schedule* problem, where the goal is to decide whether a graph G has a growth schedule of k slots and $\ell = 0$ excess edges. We define the *candidate elimination ordering* of a graph G as an ordering $v_1, v_2, \ldots, v_n$ of $V(G)$ so that for every vertex v_i, there is some v_j, where $j < i$ such that $N[v_i] \subseteq N[v_j]$ in the subgraph induced by $v_i, \ldots, v_n$, for $1 \leq i \leq n$. We show that a graph has a growth schedule of $k = n - 1$ slots and $\ell = 0$ excess edges if and only if it is has a *candidate elimination ordering.* Our main positive result is a polynomial-time algorithm that computes whether a graph has a growth schedule of $k = \log n$ slots and $\ell = 0$ excess edges. If it does, the algorithm also outputs such a growth schedule. On the negative side, we give two strong hardness results. We first show that the decision version of the zero-excess growth schedule problem is NP-complete. Then, we prove that, for every $\varepsilon > 0$, there is no polynomial-time algorithm which computes a $n^{\frac{1}{3}-\varepsilon}$-approximate zero-excess growth schedule, unless P = NP.

In Section 4, we study growth schedules of (poly)logarithmic slots. We provide two polynomial-time algorithms. One outputs, for any tree graph, a growth schedule of $O(\log^2 n)$ slots and only $O(n)$ excess edges, and the other outputs, for any planar graph, a growth schedule of $O(\log n)$ slots and $O(n \log n)$ excess edges. Finally, in the appendix, we give lower bounds on the number of excess edges required to grow a graph, when the number of slots is fixed to $\log n$.

In the appendix, we also discuss interesting problems opened by this work.

2 Preliminaries

2.1 Model and Problem Statement

A *growing graph* is modeled as an undirected dynamic graph $G_t = (V_t, E_t)$, where $t = 1, 2, \ldots, k$ is a discrete time-step, called *slot.* The dynamics of G_t are determined by a centralized *growth process* (also called *growth schedule*) σ, defined as follows. The process always starts from the initial graph instance $G_0 = (\{u_0\}, \emptyset)$, containing a single initial vertex u_0, called the *initiator.* In every slot t, the process updates the current graph instance G_{t-1} to generate the next, G_t, according to the following vertex and edge update rules. The process first sets $G_t = G_{t-1}$. Then, for every $u \in V_{t-1}$, it adds at most one new vertex u' to V_t (*vertex generation* operation) and adds to E_t the edge uu' alongside any subset of the edges $\{vu' \mid v \in V_{t-1}$ is at distance at most $d - 1$ from u in $G_{t-1}\}$, for some integer *edge-activation distance* $d \geq 1$ fixed in advance (*edge activation* operation). Throughout the rest of the paper, $d = 2$ is always assumed (other edge-activation distances are being studied in the appendix). We call u' the vertex generated by the process for vertex u in slot t. We also say that u is the *parent* of u' and that u' is the *child* of u at slot t and write $u \xrightarrow{t} u'$. The process completes slot t after deleting any (possibly empty) subset of edges from E_t (*edge deletion* operation). We also denote by V_t^+, E_t^+, and E_t^- the set of vertices generated, edges activated, and edges deleted in slot t, respectively. Then, $G_t = (V_t, E_t)$ is also given by $V_t = V_{t-1} \cup V_t^+$ and $E_t = (E_{t-1} \cup E_t^+) \setminus E_t^-$. Deleted edges are called *excess edges* and we restrict attention to excess edges whose deletion does not disconnect G_t. We call G_t the graph *grown* by process σ after t slots and call the final instance, G_k, the *target graph* grown by σ. We also say that σ is a *growth schedule for* G_k that grows G_k in k slots using ℓ *excess edges*, where $\ell = \sum_{t=1}^{k} |E_t^-|$, i.e., ℓ is equal to the total number of deleted edges. This brings us to the main problem studied in this paper:

Graph Growth Problem: Given a target graph G and non-negative integers k and ℓ, compute a growth schedule for G of at most k slots and at most ℓ excess edges, if one exists.

The *target graph* G, which is part of the input, will often be drawn from a given graph family F, e.g., the family of planar graphs. Throughout, n denotes the number of vertices of the target graph G. In this paper, computation is always to be performed by a *centralized* polynomial-time algorithm.

Let w be a vertex generated in a slot t, for $1 \leq t \leq k$. The *birth path* of vertex w is the unique sequence $B_w = (u_0, u_{i_1}, \ldots, u_{i_{p-1}}, u_{i_p} = w)$ of vertices, where $i_p = t$ and $u_{i_{j-1}} \xrightarrow{i_j} u_{i_j}$, for every $j = 1, 2, \ldots, p$. That is, B_w is the sequence of vertex generations that led to the generation of vertex w. Furthermore, the *progeny* of a vertex u is the set P_u of descendants of u, i.e., P_u contains those vertices v for which $u \in B_v$ holds.

2.2 Basic Subprocesses

We start by presenting simple algorithms for two basic growth processes that are recurrent both in our positive and negative results. One is the process of growing any path graph and the other is that of growing any star graph. Both returned growth schedules use a number of slots which is logarithmic and a number of excess edges which is linear in the size of the target graph. Logarithmic being a trivial lower bound on the number of slots required to grow graphs of n vertices, both schedules are optimal w.r.t. their number of slots. As we show in the appendix from Corollary 2 in Section 4.3, they are also optimal w.r.t. the number of excess edges used for this time-bound.

***Path* algorithm:** Let u_0 always be the "left" endpoint of the path graph being grown. For any target path graph G on n vertices, the algorithm computes a growth schedule for G as follows. For every slot $1 \leq t \leq \lceil \log n \rceil$ and every vertex $u_i \in V_{t-1}$, for $0 \leq i \leq 2^{t-1} - 1$, it generates a new vertex u_i' and connects it to u_i. Then, for all $0 \leq i \leq 2^{t-1} - 2$, it connects u_i' to u_{i+1} and deletes the edge $u_i u_{i+1}$. Finally, it renames the vertices $u_0, u_1, \ldots, u_{2^t-1}$ from left to right, before moving on to the next slot.

Lemma 1. *For any path graph G on n vertices, the* `path` *algorithm computes in polynomial time a growth schedule σ for G of $\lceil \log n \rceil$ slots and $O(n)$ excess edges.*

***Star* algorithm:** The description of the algorithm can be found in Section 1.2.

Lemma 2. *For any star graph G on n vertices, the* `star` *algorithm computes in polynomial time a growth schedule σ for G of $\lceil \log n \rceil$ slots and $O(n)$ excess edges.*

3 Growth Schedules of Zero Excess Edges

In this section, we study which target graphs G can be grown using $\ell = 0$ excess edges for $d = 2$. We begin by providing an algorithm that decides whether a graph G can be grown by any schedule σ. We build on to that, by providing an algorithm that computes a schedule of $k = \log n$ slots for a target graph G, if one exists. We finish with our main technical result showing that computing the smallest schedule for a graph G is NP-complete and any approximation of the shortest schedule cannot be within a factor of $n^{\frac{1}{3}-\varepsilon}$ of the optimal solution, for any $\varepsilon > 0$, unless $P = NP$. First, we check whether a graph G has a growth schedule of $\ell = 0$ excess edges. Observe that a graph G has a growth schedule if and only if it has a schedule of $k = n - 1$ slots.

Definition 1. *Let $G = (V, E)$ be any graph. A vertex $v \in V$ can be the last generated vertex in a growth schedule σ of $\ell = 0$ for G if there exists a vertex $w \in V \setminus \{v\}$ such that $N[v] \subseteq N[w]$. In this case, v is called a* candidate *vertex and w is called the* candidate parent *of v. Furthermore, the set of candidate vertices in G is denoted by $S_G = \{v \in V : N[v] \subseteq N[w]$ for some $w \in V \setminus \{v\}\}$.*

Definition 2. *A candidate elimination ordering of a graph G is an ordering $v_1, v_2, \ldots, v_n$ of $V(G)$ such that v_i is a candidate vertex in the subgraph induced by $v_i, \ldots, v_n$, for $1 \leq i \leq n$.*

Lemma 3. *A graph G has a growth schedule of $n - 1$ slots and $\ell = 0$ excess edges if and only if G has a candidate elimination ordering.*

The following algorithm can decide whether a graph has a candidate elimination ordering, and therefore, whether it can be grown with a schedule of $n-1$ slots and $\ell = 0$ excess edges. The algorithm computes the slots of the schedule in reverse order. The pseudo-code is provided in the appendix (Algorithm 3).

***Candidate elimination ordering* algorithm**: Given the graph $G = (V, E)$, the algorithm finds all candidate vertices and deletes an arbitrary candidate vertex and its incident edges. The deleted vertex is added in the last empty slot of the schedule σ. The algorithm repeats the above process until there is only a single vertex left. If that is the case, the algorithm produces a growth schedule. If the algorithm cannot find any candidate vertex for removal, it decides that the graph cannot be grown.

Theorem 1. *The* `candidate elimination ordering` *algorithm is a polynomial-time algorithm that, for any graph G, decides whether G has a growth schedule of $n-1$ slots and $\ell = 0$ excess edges, and it outputs such a schedule if one exists.*

The notion of candidate elimination orderings turns out to coincide with the notion of cop-win orderings, discovered in the past in graph theory for a class of graphs, called cop-win graphs [5,22,29]. In particular, it is not hard to show that *a graph has a candidate elimination ordering if and only if it is a cop-win graph.* This implies that our `candidate elimination ordering` algorithm is probably equivalent to some folklore algorithms in the literature of cop-win graphs.

Our next goal is to decide whether a graph $G = (V, E)$ on n vertices has a growth schedule σ of $\log n$ slots and $\ell = 0$ excess edges. For easiness of notation, we assume that $n = 2^\delta$ for some integer δ. This assumption can be easily removed. The `fast growth` algorithm computes the slots of the growth schedule in reverse order. The pseudo-code is provided in the appendix (Algorithm 4).

`Fast growth` **algorithm**: The algorithm finds set S_G of candidate vertices in G. It then tries to find a subset $L \subseteq S_G$ of candidates that satisfies all of the following: 1. $|L| = n/2$. 2. L is an independent set. 3. There is a perfect matching between the candidate vertices in L and their candidate parents in G. Any set L that satisfies the above constraints is called *valid*. The algorithm finds such a set by creating a 2-SAT formula ϕ whose solution is a valid set L. If the algorithm finds such a set L, it adds the vertices in L to the last slot of the schedule. It then removes the vertices in L from graph G along with their incident edges. The above process is then repeated to find the next slots. If at any point, graph G has a single vertex, the algorithm terminates and outputs the schedule. If at any point, the algorithm cannot find a valid set L, it outputs "no".

Theorem 2. *For any graph G on 2^δ vertices, the* `fast growth` *algorithm computes in polynomial time a growth schedule σ for G of $\log n$ slots and $\ell = 0$ excess edges, if one exists.*

We will now show that the problem of computing the minimum number of slots required for a graph G to be grown is NP-complete, and that it cannot be approximated within a $n^{\frac{1}{3}-\varepsilon}$ factor for any $\varepsilon > 0$, unless P = NP.

Definition 3. *Given any graph G and a natural number κ, find a growth schedule of κ slots and $\ell = 0$ excess edges. We call this problem* zero-excess growth schedule.

Theorem 3. *The decision version of the zero-excess graph growth problem is NP-complete.*

Theorem 4. *Let $\varepsilon > 0$. If there exists a polynomial-time algorithm, which, for every graph G, computes a $n^{\frac{1}{3}-\varepsilon}$-approximate growth schedule (i.e., a growth schedule with at most $n^{\frac{1}{3}-\varepsilon}\kappa(G)$ slots), then P = NP.*

Proof. The reduction is from the minimum coloring problem. Given an arbitrary graph $G = (V, E)$ with n vertices, we construct in polynomial time a graph $G' = (V', E')$ with $N = 4n^3$ vertices, as follows: We create $2n^2$ isomorphic copies of G, which are denoted by $G_1^A, G_2^A, \ldots, G_{n^2}^A$ and $G_1^B, G_2^B, \ldots, G_{n^2}^B$, and we also add n^2 clique graphs, each of size $2n$, denoted by $C_1, C_2, \ldots, C_{n^2}$. We define $V' = V(G_1^A) \cup \ldots \cup V(G_{n^2}^A) \cup V(G_1^B) \cup \ldots \cup V(G_{n^2}^B) \cup V(C_1) \cup \ldots \cup V(C_{n^2})$. Initially we add to the set E' the edges of all graphs $G_1^A, \ldots, G_{n^2}^A$, $G_1^B, \ldots, G_{n^2}^B$, and $C_1, \ldots, C_{n^2}$. For every $i = 1, 2, \ldots, n^2 - 1$ we add to E' all edges between $V(G_i^A) \cup V(G_i^B)$ and $V(G_{i+1}^A) \cup V(G_{i+1}^B)$. For every $i = 1, \ldots, n^2$, we add to E' all edges between $V(C_i)$ and $V(G_i^A) \cup V(G_i^B)$. Furthermore, for every $i = 2, \ldots, n^2$, we add to E' all edges between $V(C_i)$ and $V(G_{i-1}^A) \cup V(G_{i-1}^B)$. For every $i = 1, \ldots, n^2 - 1$, we add to E' all edges between $V(C_i)$ and $V(C_{i+1})$. For every $i = 1, 2, \ldots, n^2$ and for every $u \in V(G_i^B)$, we add to E' the edge uu', where $u' \in V(G_i^A)$ is the image of u in the isomorphism mapping between G_i^A and G_i^B. To complete the construction, we pick an arbitrary vertex a_i from each C_i. We add edges among the vertices $a_1, \ldots, a_{n^2}$ such that the resulting induced graph $G'[a_1, \ldots, a_{n^2}]$ is a graph on n^2 vertices which can be grown by a *path* schedule within $\lceil \log n^2 \rceil$ slots and with zero excess edges (see Lemma 1[6]). This completes the construction of G'. Clearly, G' can be constructed in time polynomial in n.

Now we will prove that there exists a growth schedule σ' of G' of length at most $n^2\chi(G) + 4n - 2 + \lceil 2 \log n \rceil$. The schedule will be described inversely, that is, we will describe the vertices generated in each slot starting from the last slot of σ' and finishing with the first slot. First note that every $u \in V(G_{n^2}^A) \cup V(G_{n^2}^B)$ is a candidate vertex in G' Indeed, for every $w \in V(C_{n^2})$, we have that $N[u] \subseteq V(G_{n^2}^A) \cup V(G_{n^2}^B) \cup V(G_{n^2-1}^A) \cup V(G_{n^2-1}^A) \cup V(C_{n^2}) \subseteq N[w]$. To provide the desired growth schedule σ', we assume that a minimum coloring of the input graph G (with $\chi(G)$ colors) is known. In the last $\chi(G)$ slots, σ' generates all vertices in $V(G_{n^2}^A) \cup V(G_{n^2}^B)$, as follows. At each of these slots, one of the $\chi(G)$ color classes of the minimum coloring c_{OPT} of $G_{n^2}^A$ is generated on sufficiently many vertices among the first n vertices of the clique C_{n^2}. Simultaneously, a different color class of the minimum coloring c_{OPT} of $G_{n^2}^B$ is generated on sufficiently many vertices among the last n vertices of the clique C_{n^2}.

Similarly, for every $i = 1, \ldots, n^2 - 1$, once the vertices of $V(G_{i+1}^A) \cup \ldots \cup V(G_{n^2}^A) \cup V(G_{i+1}^B) \cup \ldots \cup V(G_{n^2}^B)$ have been added to the last $(n^2 - i)\chi(G)$ slots of σ', the vertices of $V(G_i^A) \cup V(G_i^B)$ are generated in σ' in $\chi(G)$ more slots. This is possible because every vertex $u \in V(G_i^A) \cup V(G_i^B)$ is a candidate vertex after the vertices of $V(G_{i+1}^A) \cup \ldots \cup V(G_{n^2}^A) \cup V(G_{i+1}^B) \cup \ldots \cup V(G_{n^2}^B)$ have been added to slots. Indeed, for every $w \in V(C_i)$, we have that $N[u] \subseteq V(G_i^A) \cup V(G_i^B) \cup V(G_{i-1}^A) \cup V(G_{i-1}^A) \cup V(C_i) \subseteq N[w]$. That is, in total, all vertices of $V(G_1^A) \cup \ldots \cup V(G_{n^2}^A) \cup V(G_1^B) \cup \ldots \cup V(G_{n^2}^B)$ are generated in the last $n^2\chi(G)$ slots.

The remaining vertices of $V(C_1) \cup \ldots \cup V(C_{n^2})$ are generated in σ' in $4n - 2 + \lceil \log n^2 \rceil$ additional slots. First, for every odd index i and for $2n - 1$ consecutive slots, for vertex a_i of $V(C_i)$ exactly one other vertex of $V(C_i)$ is generated. This is possible because for every vertex $u \in V(C_i) \setminus a_i$, $N[u] \subseteq V(C_i) \cup V(C_{i-1}) \cup V(C_{i+1}) \subseteq N[a_i]$. Then, for every even index i and for $2n-1$ further consecutive

[6] From Lemma 1 it follows that the path on n^2 vertices can be constructed in $\lceil \log n^2 \rceil$ slots using $O(n^2)$ excess edges. If we put all these $O(n^2)$ excess edges back to the path of n^2 vertices, we obtain a new graph on n^2 vertices with $O(n^2)$ edges. This graph is the induced subgraph $G'[a_1, \ldots, a_{n^2}]$ of G' on the vertices $a_1, \ldots, a_{n^2}$.

slots, for vertex a_i of $V(C_i)$ exactly one other vertex of $V(C_i)$ is generated. That is, after $4n-2$ slots only the induced subgraph of G' on the vertices $a_1, \ldots, a_{n^2}$ remains. The final $\lceil \log n^2 \rceil$ slots of σ' are the ones obtained by Lemma 1. To sum up, G' is grown by the growth schedule σ' in $k = n^2\chi(G) + 4n - 2 + \lceil \log n^2 \rceil$ slots, and thus $\kappa(G') \leq n^2\chi(G) + 4n - 2 + \lceil 2 \log n \rceil$ (1).

Suppose that there exists a polynomial-time algorithm A which computes an $N^{\frac{1}{3}-\varepsilon}$-approximate growth schedule σ'' for graph G' (which has N vertices), i.e., a growth schedule of $k \leq N^{\frac{1}{3}-\varepsilon}\kappa(G')$ slots. Note that, for every slot of σ'', all different vertices of $V(G_i^A)$ (resp. $V(G_i^B)$) which are generated in this slot are independent. For every $i = 1, \ldots, n^2$, denote by χ_i^A (resp. χ_i^B) the number of different slots of σ'' in which at least one vertex of $V(G_i^A)$ (resp. $V(G_i^B)$) appears. Let $\chi^* = \min\{\chi_i^A, \chi_i^B : 1 \leq i \leq n^2\}$. Then, there exists a coloring of G with at most χ^* colors (i.e., a partition of G into at most χ^* independent sets).

Now we show that $k \geq \frac{1}{2}n^2\chi^*$. Let $i \in \{2, \ldots, n^2-1\}$ and let $u \in V(G_i^A) \cup V(G_i^B)$. Assume that u is generated at slot t in σ''. Then, either all vertices of $V(G_{i-1}^A) \cup V(G_{i-1}^B)$ or all vertices of $V(G_{i+1}^A) \cup V(G_{i+1}^B)$ are generated at a later slot $t' \geq t+1$ in σ''. Indeed, it can be easily checked that, if otherwise both a vertex $x \in V(G_{i-1}^A) \cup V(G_{i-1}^B)$ and a vertex $y \in V(G_{i+1}^A) \cup V(G_{i+1}^B)$ are generated at a slot $t'' \leq t$ in σ'', then u cannot be a candidate vertex at slot t, which is a contradiction to our assumption. That is, in order for a vertex $u \in V(G_i^A) \cup V(G_i^B)$ to be generated at some slot t of σ'', we must have that i is either the currently smallest or largest index for which some vertices of $V(G_i^A) \cup V(G_i^B)$ have been generated until slot t. On the other hand, by definition of χ^*, the growth schedule σ'' needs at least χ^* different slots to generate all vertices of the set $V(G_i^A) \cup V(G_i^B)$, for $1 \leq i \leq n^2$. Therefore, since at every slot, σ'' can potentially generate vertices of at most two indices i (the smallest and the largest respectively), it needs to use at least $\frac{1}{2}n^2\chi^*$ slots to grow the whole graph G'. Therefore $k \geq \frac{1}{2}n^2\chi^*$ (2).

Recall that $N = 4n^3$. It follows by Eq. (1) and Eq. (2) that

$$\begin{aligned}
\frac{1}{2}n^2\chi^* &\leq k \leq N^{\frac{1}{3}-\varepsilon}\kappa(G') \\
&\leq N^{\frac{1}{3}-\varepsilon}(n^2\chi(G) + 4n - 2 + \lceil 2 \log n \rceil) \\
&\leq 4n^{1-3\varepsilon}(n^2\chi(G) + 6n)
\end{aligned}$$

and thus $\chi^* \leq 8n^{1-3\varepsilon}\chi(G) + 48n^{-3\varepsilon}$. Note that, for sufficiently large n, we have that $8n^{1-3\varepsilon}\chi(G) + 48n^{-3\varepsilon} \leq n^{1-\varepsilon}\chi(G)$. That is, given the $N^{\frac{1}{3}-\varepsilon}$-approximate growth schedule produced by the polynomial-time algorithm A, we can compute in polynomial time a coloring of G with χ^* colors such that $\chi^* \leq n^{1-\varepsilon}\chi(G)$. This is a contradiction since for every $\varepsilon > 0$, there is no polynomial-time $n^{1-\varepsilon}$-approximation for minimum coloring, unless P = NP [36]. □

4 Growth Schedules of (Poly)logarithmic Slots

In this section, we study graphs that have growth schedules of (poly)logarithmic slots, for $d = 2$. As we have proven in the previous section, an integral factor in computing a growth schedule for any graph G, is computing a k-coloring for G. Since we consider polynomial-time algorithms, we have to restrict ourselves to graphs where the k-coloring problem can be solved in polynomial time and, additionally, we want small values of k since we want to produce fast growth schedules. Therefore, we investigate tree, planar and k-degenerate graph families since there are polynomial-time algorithms that solve the k-coloring problem for graphs drawn from these families.

4.1 Trees

We now provide an algorithm that computes growth schedules for tree graphs. Let G be the target tree graph. The algorithm applies a decomposition strategy on G, where vertices and edges are removed in phases, until a single vertex is left. We can then grow the target graph G by reversing its decomposition phases, using the ***path*** and ***star*** schedules as subroutines.

Tree **algorithm**: Starting from a tree graph G, the algorithm keeps alternating between two phases, a *path-cut* and a *leaf-cut* phase. Let G_{2i}, G_{2i+1}, for $i \geq 0$, be the graphs obtained after the execution of the first i pairs of phases and an additional path-cut phase, respectively.

Path-cut phase: For each path subgraph $P = (u_1, u_2, \ldots, u_\nu)$, for $2 < \nu \leq n$, of the current graph G_{2i}, where $u_2, u_3, ..., u_{\nu-1}$ have degree 2 and u_1, u_ν have degree $\neq 2$ in G_{2i}, edge $u_1 u_\nu$ between the endpoints of P is activated and vertices $u_2, u_3, ...u_{\nu-1}$ are removed along with their incident edges. If a single vertex is left, the algorithm terminates; otherwise, it proceeds to the leaf-cut phase.
Leaf-cut phase: Every leaf vertex of the current graph G_{2i+1} is removed along with its incident edge. If a single vertex is left, the algorithm terminates; otherwise, it proceeds to the path-cut phase.

Finally, the algorithm reverses the phases (by decreasing i) to output a growth schedule for the tree G as follows. For each path-cut phase $2i$, all path subgraphs that were decomposed in phase i are regrown by using the ***path*** schedule as a subprocess. These can be executed in parallel in $O(\log n)$ slots. The same holds true for leaf-cut phases $2i + 1$, where each can be reversed to regrow the removed leaves by using ***star*** schedules in parallel in $O(\log n)$ slots. In the last slot, the schedule deletes every excess edge. By proving that a total of $O(\log n)$ phases are sufficient to decompose any tree G and that at most one excess edge per vertex of G is activated, the next theorem follows.

Theorem 5. *For any tree graph G on n vertices, the* `tree` *algorithm computes in polynomial time a growth schedule σ for G of $O(\log^2 n)$ slots and $O(n)$ excess edges.*

4.2 Planar Graphs

In this section, we provide an algorithm that computes a growth schedule for any target planar graph $G = (V, E)$. The algorithm first computes a 5-coloring of G

and partitions the vertices into color-sets V_i, $1 \le i \le 5$. The color-sets are used to compute the growth schedule for G. The schedule contains five sub-schedules, each sub-schedule i generating all vertices in color-set V_i. In every sub-schedule i, we use a modified version of the *star* schedule to generate set V_i.

Pre-processing: By using the algorithm of [33], the pre-processing step computes a 5-coloring of the target planar graph G. This creates color-sets $V_i \subseteq V$, where $1 \le i \le 5$, every color-set V_i containing all vertices of color i. W.l.o.g., we can assume that $|V_1| \ge |V_2| \ge |V_3| \ge |V_4| \ge |V_5|$. Note that every color-set V_i is an independent set of G.

***Planar* algorithm**: The algorithm picks an arbitrary vertex from V_1 and makes it the initiator u_0 of all sub-schedules. Let $V_i = \{u_1, u_2, \ldots, u_{|V_i|}\}$. For every sub-schedule i, $1 \le i \le 5$, it uses the *star* schedule with u_0 as the initiator, to grow the vertices in V_i in an arbitrary sequence, with some additional edge activations. In particular, upon generating vertex $u_x \in V_i$, for all $1 \le x \le |V_i|$:

1. Edge vu_x is activated if $v \in \bigcup_{j<i} V_j$ and $u_y v \in E$, for some $u_y \in V_i \cap P_{u_x}$, both hold (recall that P_{u_x} contains the descendants of u_x).
2. Edge wu_x is activated if $w \in \bigcup_{j<i} V_j$ and $wu_x \in E$ both hold.

Once all vertices of V_i have been generated, the schedule moves on to generate V_{i+1}. Once all vertices have been generated, the schedule deletes every edge $uv \notin E$. Note that every edge activated in the growth schedule is an excess edge with the exception of edges satisfying (2). For an edge wu_x from (2) to satisfy the edge-activation distance constraint it must hold that every vertex in the birth path of u_x has an edge with w. This holds true for the edges added in (2), due to the edges added in (1).

The edges of the *star* schedule are used to quickly generate the vertices, while the edges of (1) are used to enable the activation of the edges of (2). By proving that the *star* schedule activate $O(n)$ edges, (1) activates $O(n \log n)$ edges, and by observing that the schedule contains *star* sub-schedules that have $5 \times O(\log n)$ slots in total, the next theorem follows.

Theorem 6. *For any planar graph G on n vertices, the* `planar` *algorithm computes in polynomial time a growth schedule for G of $O(\log n)$ slots and $O(n \log n)$ excess edges.*

Definition 4. *A k-degenerate graph G is an undirected graph in which every subgraph has a vertex of degree at most k.*

Corollary 1. *The* `planar` *algorithm can be extended to compute, for any graph G on n vertices and in polynomial time, a growth schedule of $O((k_1 + 1)\log n)$ slots, $O(k_2 n \log n)$ and excess edges, where (i) $k_1 = k_2$ is the degeneracy of graph G, or (ii) $k_1 = \Delta$ is the maximum degree of graph G and $k_2 = |E|/n$.*

References

1. Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *Journal of Computer and System Sciences*, 107:108–123, 2020.
2. Dana Angluin, James Aspnes, Jiang Chen, Yinghua Wu, and Yitong Yin. Fast construction of overlay networks. In *Proceedings of the 17th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 145–154, 2005.
3. James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37:1–37:25, 2007.
4. John Augustine, Gopal Pandurangan, Peter Robinson, and Eli Upfal. Towards robust and efficient computation in dynamic peer-to-peer networks. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 551–569, 2012.
5. Hans-Jürgen Bandelt and Erich Prisner. Clique graphs and helly graphs. *Journal of Combinatorial Theory, Series B*, 51(1):34–45, 1991.
6. Luca Becchetti, Andrea Clementi, Francesco Pasquale, Luca Trevisan, and Isabella Ziccardi. Expansion and flooding in dynamic random networks with node churn. In *Proceedings of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 976–986, 2021.
7. Kenneth A Berman. Vulnerability of scheduled networks and a generalization of menger's theorem. *Networks: An International Journal*, 28(3):125–134, 1996.
8. Béla Bollobás. *Random graphs*. Number 73 in Cambridge studies in advanced mathematics. Cambridge University Press, 2nd edition, 2001.
9. Luca Bombelli, Joohan Lee, David Meyer, and Rafael D Sorkin. Space-time as a causal set. *Physical Review Letters*, 59(5):521, 1987.
10. Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.
11. Michelle M Chan, Zachary D Smith, Stefanie Grosswendt, Helene Kretzmer, Thomas M Norman, Britt Adamson, Marco Jost, Jeffrey J Quinn, Dian Yang, Matthew G Jones, et al. Molecular recording of mammalian embryogenesis. *Nature*, 570(7759):77–82, 2019.
12. Ioannis Chatzigiannakis, Athanasios Kinalis, and Sotiris Nikoletseas. Adaptive energy management for incremental deployment of heterogeneous wireless sensors. *Theory of Computing Systems*, 42:42–72, 2008.
13. Mário Cordeiro, Rui P. Sarmento, Pavel Brazdil, and João Gama. Evolving networks and social network analysis methods and techniques. In *Social Media and Journalism*, chapter 7. 2018.
14. David Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.
15. Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021.
16. Seth Gilbert, Gopal Pandurangan, Peter Robinson, and Amitabh Trehan. Dconstructor: Efficient and robust network construction with polylogarithmic overhead. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*, 2020.
17. Thorsten Götte, Kristian Hinnenthal, and Christian Scheideler. Faster construction of overlay networks. In *Proceedings of the 26th International Colloquium on*

Structural Information and Communication Complexity (SIROCCO), pages 262–276. Springer, 2019.
18. Thorsten Götte, Kristian Hinnenthal, Christian Scheideler, and Julian Werthmann. Time-optimal construction of overlay networks. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 457–468, 2021.
19. Andrew Howard, Maja J. Matarić, and Gaurav S Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, pages 113–126, 2002.
20. David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.
21. Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S." Tomkins. The web as a graph: Measurements, models, and method. In *Computing and Combinatorics*, pages 1–17. Springer Berlin Heidelberg, 1999.
22. Min Chih Lin, Francisco J. Soulignac, and Jayme Luiz Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theoretical Computimg Science*, 426:75–90, 2012.
23. George B Mertzios, Othon Michail, and Paul G Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019.
24. Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.
25. Othon Michail, George Skretas, and Paul G Spirakis. On the transformation capability of feasible mechanisms for programmable matter. *Journal of Computer and System Sciences*, 102:18–39, 2019.
26. Othon Michail, George Skretas, and Paul G Spirakis. Distributed computation and reconfiguration in actively dynamic networks. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 448–457, 2020.
27. Othon Michail and Paul G Spirakis. Elements of the theory of dynamic networks. *Communications of the ACM*, 61(2):72–72, 2018.
28. Hugo A. Méndez-Hernández, Maharshi Ledezma-Rodríguez, Randy N. Avilez-Montalvo, Yary L. Juárez-Gómez, Analesa Skeete, Johny Avilez-Montalvo, Clelia De-la Peña, and Víctor M. Loyola-Vargas. Signaling overview of plant somatic embryogenesis. *Frontiers in Plant Science*, 10, 2019.
29. Tim Poston. *Fuzzy Geometry*. PhD thesis, University of Warwick, 1971.
30. David Porter Rideout and Rafael D Sorkin. Classical sequential growth dynamics for causal sets. *Physical Review D*, 61(2):024002, 1999.
31. Paul WK Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.
32. Christian Scheideler and Alexander Setzer. On the complexity of local graph transformations. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
33. Howard Williams. A linear algorithm for colouring planar graphs with five colours. *The Computer Journal*, 28:78–81, 1985.
34. Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.
35. Philipp Zschoche, Till Fluschnik, Hendrik Molter, and Rolf Niedermeier. The complexity of finding small separators in temporal graphs. *Journal of Computer and System Sciences*, 107:72–92, 2020.

36. David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.