Algorithmica (2023) 85:2454–2481
https://doi.org/10.1007/s00453-023-01107-1

Certifying Fully Dynamic Algorithms for Recognition and Hamiltonicity of Threshold and Chain Graphs

Jesse Beisegel[1] · **Ekkehard Köhler**[1] · **Robert Scheffler**[1] · **Martin Strehler**[2]

Received: 8 November 2021 / Accepted: 7 February 2023 / Published online: 28 February 2023

Abstract

Solving problems on graphs dynamically calls for algorithms to function under repeated modifications to the graph and to be more efficient than solving the problem for the whole graph from scratch after each modification. Dynamic algorithms have been considered for several graph properties, for example connectivity, shortest paths and graph recognition. In this paper we present fully dynamic algorithms for the recognition of threshold graphs and chain graphs, which are optimal in the sense that the costs per modification are linear in the number of modified edges. Furthermore, our algorithms also consider the addition and deletion of sets of vertices as well as edges. In the negative case, i.e., where the graph is not a threshold graph or chain graph anymore, our algorithms return a certificate of constant size. Additionally, we present optimal fully dynamic algorithms for the Hamiltonian cycle problem and the Hamiltonian path problem on threshold and chain graphs which return a vertex cutset as certificate for the non-existence of such a path or cycle in the negative case.

Keywords Fully dynamic algorithms · Threshold graphs · Chain graphs · Difference graphs · Hamiltonian cycles · Hamiltonian paths

Mathematics Subject Classification 05C85 · 05C45 · 68R10

✉ Robert Scheffler
robert.scheffler@b-tu.de

Jesse Beisegel
jesse.beisegel@b-tu.de

Ekkehard Köhler
ekkehard.koehler@b-tu.de

Martin Strehler
martin.strehler@fh-zwickau.de

[1] Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

[2] Department of Mathematics, Westsächsische Hochschule Zwickau, Zwickau, Germany

1 Introduction

In many graph-theoretical applications graphs are not static, but change over time. For example in a social network it is common that vertices (representing members of the network) as well as edges (representing a connection between two members) can be added or deleted at any given time. As these types of networks can be very large, it is important to update certain properties of a network efficiently when making such a modification. Due to the size of some of these networks, however, even a running time linear in the size of the network can be prohibitive, begging the question whether it is possible to make such an update either in time logarithmic in the input or ideally with a running time linear in the size of the modification. In particular, we can ask if it is possible to use such a scheme when checking whether a given graph is part of some graph class. Such schemes are called dynamic recognition algorithms and are categorized depending on the modification operations they support: An *edges-only dynamic recognition algorithm* supports both edge additions and edge deletions. A *fully dynamic recognition algorithm* supports edge modifications as well as vertex additions and deletions. Note that for vertex additions one usually is also given a set of neighbors of the new vertex. A dynamic recognition algorithm is said to be *optimal* if the running time of each operation is linear in the number of edges involved in that operation.

Dynamic recognition algorithms have been developed for many different graph classes. For example, Ibarra [20] presented edges-only fully dynamic recognition algorithms for chordal graphs and split graphs. Similar results were attained by Hell et al. [16] for proper interval graphs, by Crespelle and Paul [9] for permutation graphs and by Soulignac [28] for proper circular arc graphs. Furthermore, Shamir and Sharan [27] formulated a fully dynamic algorithm for the recognition of cographs. The combination of this algorithm with Ibarra's for split graphs implies a fully dynamic recognition algorithm for threshold graphs, as these form the intersection of split and cographs [27]. However, this algorithm does not yield a certificate for the negative case and does not fully represent the special structure of threshold graphs, as it relies on algorithms for other graph classes. Besides the recognition of graph classes, fully dynamic algorithms have also been developed for many other graph-theoretical problems such as connectivity [18, 19, 29], shortest paths [10, 21] and matching approximation [2], as well as for minimal integral separators for threshold and difference graphs [4].

Threshold graphs were first introduced by Hammer and Chvátal [5, 6], where they were used to solve the aggregation problem for set-packing inequalities, and, independently by Henderson and Zalcstein [17]. Furthermore, Hammer and Chvátal [6] showed that threshold graphs can be characterized as the graphs which do not contain an induced subgraph isomorphic to $2K_2$, P_4 or C_4. Since then, there has been considerable interest in this graph class leading to many interesting results. A survey can be found in the book by Peled and Mahadev [23]. Apart from linear integer programming, threshold graphs have also been used in applications as diverse as mathematical psychology [7, 8], scheduling [22] and parallel computing [24, 25].

In the context of fully dynamic recognition algorithms, the relationship between threshold graphs and the aggregation problem for set packing inequalities is of particular interest. In this problem one is given a linear integer program containing a

collection of set-packing inequalities and wishes to find an equivalent program in which this collection of inequalities is covered by a single equation. In [6], Hammer and Chvátal showed that this problem is in fact equivalent to the recognition of threshold graphs. To this end, each variable of the linear program contained in set-packing inequalities is identified with a vertex of a graph, where two vertices are connected by an edge if they both appear together in some equation. The binary solutions to these inequalities correspond to the independent sets of the constructed graph. In particular, they can be aggregated to one linear equation if and only if the graph is threshold. In this context, adding and removing vertices from the graph can be seen as the addition or removal of variables, while the addition and deletion of edges corresponds to the modification of the inequalities.

Chain graphs (also known as difference graphs in the literature), introduced by Yannakakis [30] and also independently by Hammer et al. [11], are closely related to threshold graphs and in many ways form a bipartite variant of threshold graphs. In fact, a bipartite graph G with bipartition (X, Y) is a chain graph if and only if adding all edges between vertices in X yields a threshold graph [23]. Equivalently, Yannakakis [30] characterized chain graphs as bipartite graphs that do not contain an induced $2K_2$. Heggernes and Kratsch showed that both threshold and chain graphs can be recognized in linear time [13].

Our Contribution. In this paper, we present optimal fully dynamic recognition algorithms for threshold graphs and chain graphs. These algorithms are certifying, i.e., they return a certificate in form of a forbidden subgraph for the negative case. The algorithms need linear time for preprocessing and the running time of the operations are as follows: $\mathcal{O}(1)$ for deleting and inserting an edge and $\mathcal{O}(d)$ for inserting or deleting a vertex v which has d neighbors in G. For threshold graphs Shamir and Sharan [27] already introduced a fully dynamic recognition algorithm with similar running times. However, their algorithm is not certifying and uses the fully dynamic recognition algorithms for split graphs and cographs, while our implementation is self contained, using an efficient data structure for threshold graphs, and yields certificates.

Furthermore, we expand on the idea of a fully dynamic algorithm by giving efficient routines for the addition and deletion of sets of vertices or edges. While for the studied graph classes the modification of sets of *vertices* is a straightforward extension of the previous results, the modification of sets of *edges* needs a different approach which leads to a running time of $\mathcal{O}(\min\{k \log k, n + k\})$, where k is the size of the modified edge set. This approach uses special edge elimination schemes introduced by Beisegel et al. [1].

Using concepts from these recognition algorithms, we also present fully dynamic algorithms for both threshold graphs and chain graphs that decide whether a given threshold and chain graph has a Hamiltonian cycle or a Hamiltonian path. These algorithms have running times that are identical to the running times of the recognition algorithms. In the positive case, they can find a Hamiltonian cycle or path in time $\mathcal{O}(n)$. Otherwise, they can find in time $\mathcal{O}(n)$ a vertex cutset that separates the graph into many connected components. This can be used as a certificate for the non-existence of such a path or cycle. To the best of our knowledge, these are the first dynamic algorithms for the Hamiltonian path and cycle problem.

2 Preliminaries

Throughout this paper, we consider graphs $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, which are always finite, simple and undirected. Apart from possible *isolated vertices*, i.e., vertices without incident edges, the graphs are connected, that is, they have only one non-trivial component. An edge between u and v is simply denoted by uv. For a vertex $v \in V$, the *(open) neighborhood* of v is denoted by $N(v)$, i.e., $N(v) := \{u \in V \mid uv \in E\}$, the *closed neighborhood* $N[v]$ *is defined as* $N[v] := N(v) \cup \{v\}$. The *degree* of a vertex x in G is denoted by $d_G(v) := |N(v)|$.

Given a subset of vertices $U \subseteq V$, the *subgraph of* G *induced by* U is denoted by $G[U]$ and has vertex set $V(G[U]) = U$ and edge set $E(G[U]) = \{uv \in E(G) \mid u, v \in U\}$. The subgraph induced by $V(G)\backslash U$ is denoted by $G - U$ and, in the case that U contains just one element $v \in U$, we simply write $G - v$ instead of $G - \{v\}$.

Let u and v be two different vertices of $G = (V, E)$. Then, $G - uv$ denotes the graph without edge uv, i.e., the graph with vertex set V and edge set $E' = E\backslash\{uv\}$. Similarly, $G + uv$ is the graph with edge uv, that is, $E' = E \cup \{uv\}$.

In this paper, we mainly focus on two graph classes, namely threshold graphs and chain graphs. A graph is a *threshold graph* if there are vertex weights $w(v)$ for each vertex $v \in V$ and a value S (the threshold) such that $uv \in E$ if and only if $w(u) + w(v) > S$. Although this is the standard definition of threshold graphs, an equivalent definition will turn out to be more convenient for our purpose: A graph is a threshold graph if and only if it can be generated from a one-vertex graph by repeated additions of isolated vertices or *universal vertices*, i.e., vertices that are adjacent to all already existing vertices. Furthermore, threshold graphs are exactly the graphs that do not contain an induced P_4, C_4 or $2K_2$ [6], that is, there are no induced paths or cycles with four vertices and no two non-adjacent edges.

Similarly, chain graphs can be defined in various ways. A graph is a *chain graph* if there are a positive threshold T and vertex weights $w(v)$ with $|w(v)| < T$ such that $uv \in E$ if and only if $|w(u) - w(v)| > T$. Therefore, these graphs are also called *difference graphs* [11]. In fact, chain graphs are exactly the bipartite $2K_2$-free graphs [30]. The complement of a bipartite graph G is an interval graph if and only if G is a chain graph. Furthermore, chain graphs can be characterized as those bipartite graphs with vertex bipartition $B_1 \dot{\cup} B_2$, where the neighborhoods of the vertices in B_1 are totally ordered by set inclusion, i.e., for every $u, v \in B_1$, either $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$. The same applies to B_2.

There is a close relation between threshold and chain graphs. Given a threshold graph G with the vertex partition in the set U of universal vertices and the set I of independent vertices, the graph consisting only of the edges of G between U and I and the same vertex set is a chain graph. Vice versa, completing one of the sets B_i of a chain graph to a clique yields a threshold graph [30]. Note that both threshold and chain graphs have only one non-trivial component, as they are $2K_2$-free. Moreover, threshold graphs are chordal, whereas chain graphs are AT-free and co-comparability graphs [3]. A comprehensive survey on threshold and chain graphs is given in the book of Mahadev and Peled [23].

The following lemma shows that the neighborhoods of two vertices in a threshold or chain graph can be compared by simply comparing their respective degrees. This property will be used frequently throughout.

Lemma 2.1 *Let $G = (V, E)$ be a threshold graph and let $u, v \in V$ be two vertices in G such that $N_G(u) \nsubseteq N_G[v]$. Then $d_G(u) > d_G(v)$.*

Let $G = (X \cup Y, E)$ be a chain graph and let $u, v \in X$ be two vertices in the same partition set of G such that $N_G(u) \nsubseteq N_G(v)$. Then $d_G(u) > d_G(v)$.

Proof We start with the threshold graphs. Let x be a vertex in $N_G(u) \backslash N_G[v]$. Then for the weight function $w : V \to \mathbb{R}$ and the threshold S it holds: $w(u) + w(x) > S$ and $w(v) + w(x) \leq S$. Therefore, $w(u)$ must be larger than $w(v)$ and for every vertex $y \in V$ holds $w(u) + w(y) > w(v) + w(y)$. Thus, there cannot be a vertex in $N(v)$ that is not in $N[u]$ and the degree condition holds.

For the chain graph we turn X into a clique and get a threshold graph. Since the property of the neighborhoods of u and v still holds, it follows from the first part of the lemma that the degree of u is larger than the degree of v in the threshold graph. This also holds for the chain graph, as both degrees were raised by the same number. □

This shows that vertex degrees play an important role in threshold and chain graphs and motivates the following definition which aims to simplify notation throughout.

Definition 2.2 Let uv and xy be two edges connecting vertices of a graph G with $d_G(u) \leq d_G(v)$ and $d_G(x) \leq d_G(y)$. Note that it is not necessary that these edges are contained in the edge set of G. The edge xy is called *degree-lower* than edge uv in G if:

- $x \neq u$ and $d_G(x) \leq d_G(u)$, or
- $x = u$ and $d_G(y) \leq d_G(v)$.

The edge xy is called *degree-greater* than edge uv if:

- $x \neq u$ and $d_G(x) \geq d_G(u)$, or
- $x = u$ and $d_G(y) \geq d_G(v)$.

Note that an edge xy can be both degree-lower and degree-greater than an edge $uv \neq xy$ at the same time. This holds for example if $x \neq u$ and $d_G(x) = d_G(u)$ or if $x = u$ and $d_G(y) = d_G(v)$. Furthermore, in the definition of degree-lower and degree-greater, we consider both edges and non-edges of a graph. This is necessary as we will consider both the insertion and the deletion of edges in our dynamic algorithms. For a graph G and a set of edges F connecting vertices of G, but not necessarily contained in G we say that $e \in F$ is *degree-minimal in F* if it is degree-lower in G than any other edge in F. Edge e is said to be *degree-maximal in F* if it is degree-greater in G than any other edge in F.

The problem of dynamically recognizing certain graph classes can be defined as follows: Given a graph class $\mathcal{G}$ and a graph $G \in \mathcal{G}$, one looks for a representation which allows the efficient insertion or deletion of vertices and edges and a decision whether the new graph is still in $\mathcal{G}$. Algorithms for this problem are called dynamic

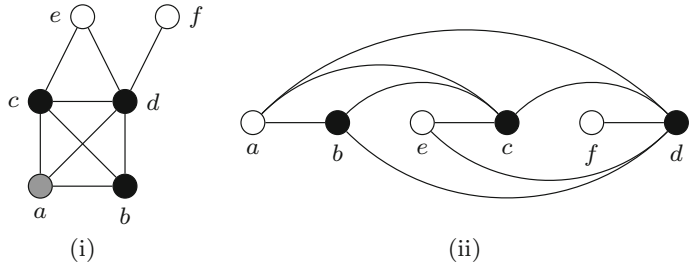


Fig. 1 A threshold graph together with its threshold partition. Black vertices belong to $\bigcup_{i=1}^{3} U_i$ and white vertices belong to the $\bigcup_{i=1}^{3} I_i$. Thus, the partition is $(I_1 = \{a\}, U_1 = \{b\}, I_2 = \{e\}, U_2 = \{c\}, I_3 = \{f\}, U_3 = \{d\})$. Usually, both a and b would be included in the clique set of the threshold graph. However, since we have an odd number of degrees, one of the vertices of degree three, i.e., vertex a, forms the set I_1 and the rest forms U_1

recognition algorithms and are categorized depending on the modification operations they support. Following the terminology used in [27], we call a dynamic recognition algorithm an *edges-only fully dynamic recognition algorithm* if it supports both edge additions and edge deletions. A *fully dynamic recognition algorithm* supports edge modifications as well as vertex additions and deletions. Note that for vertex additions one usually is also given a set of neighbors of the new vertex.

3 Threshold Graphs

Heggernes and Papadopoulos [15] introduced a vertex partition of threshold graphs which consists of an ordered partition of the clique set and an ordered partition of the independent set. Here, we introduce a similar structure that merges these two partitions into one and swaps the order of the partition sets (see Fig. 1 for an example).

Definition 3.1 Let $G = (V, E)$ be a threshold graph. A *threshold partition* of G is an ordered partition $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ of the vertex set V with $I_i, U_i \neq \emptyset$ for all $1 \leq i \leq k$ such that $xy \in E$ if and only if $x \in I_i \cup U_i$ and $y \in U_j$ or vice versa for some $i \leq j$.

Note that the set I_{k+1} contains the isolated vertices of G and can be empty. The existence of a threshold partition for every threshold graph follows from the fact that a threshold graph can be generated from a one-vertex graph by repeated additions of isolated or universal vertices. The following observation is a direct result of the definition of the threshold partition and Lemma 2.1.

Observation 3.2 *Let $G = (V, E)$ be a threshold graph with threshold partition $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$.*

- *If $x, y \in I_i$ or $x, y \in U_i$, then $d_G(x) = d_G(y)$.*
- *If $x \in I_i$ and $y \in I_j$ with $i < j$, then $d_G(x) > d_G(y)$.*
- *If $x \in U_i$ and $y \in U_j$ with $i < j$, then $d_G(x) < d_G(y)$.*
- *If $x \in I_i$ and $y \in U_j$, then $d_G(x) \leq d_G(y)$ and unless $i = j = 1$ and $|I_1| = 1$, it holds that $d_G(x) < d_G(y)$.*

Thus, apart from I_1 and U_1, the sets of the threshold partition coincide with the sets of vertices of the same degree. However, the U-sets are ordered increasingly by the degree of their vertices while the I-sets are ordered decreasingly. As we will see in the following, this unusual ordering ensures that vertices incident to edges which are safe to add or to delete lie close to each other in the threshold partition. This would not be the case if we order the partition sets strictly due to their vertex degrees. As well as leading to more intuitive proofs, this definition also has benefits with regard to the implementation of this structure.

Observation 3.2 also implies a linear-time algorithm to compute such a partition.

Theorem 3.3 *There is an algorithm that computes the threshold partition of a threshold graph in time $\mathcal{O}(n + m)$.*

Proof We sort the vertices by their degrees using bucket sort. If there are vertices of degree zero, then they form the set I_{k+1} and we delete them from the graph G. Then we consider the vertices with the smallest degree of the remaining graph. They form set I_k and their neighbors form set U_k. Note that, due to G being a threshold graph, all vertices contained in U_k are universal in G. Again, we delete both the vertices of I_k and of U_k from G. The resulting graph is still a threshold graph and the degree ordering does not change as the vertices of U_k are adjacent to all remaining vertices and the vertices of I_k are adjacent to none of them. We repeat this procedure until we have either partitioned the whole graph or we only have one bucket S left, i.e., all the remaining vertices have the same degree. The degree of these remaining vertices was not minimal in the step before, i.e., these vertices have neighbors that were not neighbors of the vertices in I_2. This implies that there are at least two vertices in S and the vertices of S are pairwise adjacent. We choose one of the vertices of S that forms I_1, the other vertices of S form U_1.

The whole procedure has a running time of $\mathcal{O}(n + m)$. The first sorting of the vertices can be found in linear time. Because there is no resorting needed, the rest of the procedure can also be done in linear time. □

In the following, we will present three lemmas that use the threshold partition to characterize edges and vertices that can be deleted from a threshold graph or added to it such that it is still a threshold graph. The proofs of these lemmas are constructive, i.e., for the positive case they provide the updated threshold partition of the new graph and for the negative case they imply a certificate for the graph not being a threshold graph in form of a forbidden induced subgraph. Both the new threshold partition as well as the forbidden subgraph will be helpful for our fully dynamic algorithm for threshold graph recognition. Additionally, we will show that the vertices of the certificate still induce a forbidden subgraph if we delete or add further edges with a special property on the degrees of their vertices. We will use these results in Sect. 5 to present a certifying dynamic algorithm for the addition and deletion of several edges at the same time.

The following data structure will be used throughout to represent threshold graphs.

Definition 3.4 *(Threshold data structure)* For a threshold graph $G = (V, E)$ a *threshold data structure* consists of the following elements:

- a threshold partition $\mathcal{P}$ of G as a doubly linked list of doubly linked lists,

- a pointer from every vertex $v \in V$ to its list L in $\mathcal{P}$ and to its element in L
- for every list in $\mathcal{P}$ the information whether it represents a U-set or an I-set

We begin with the deletion of an edge.

Lemma 3.5 *Let $G = (V, E)$ be a threshold graph with a threshold data structure containing threshold partition $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ and let $xy \in E$. There is an algorithm with running time in $\mathcal{O}(1)$ which returns a threshold data structure of $G - xy$ if one of the following conditions holds:*

1. *$|I_1| = 1$ and both x and y are elements of U_1*
2. *$x \in I_j$ and $y \in U_j$ for some $j \in \{1, \ldots, k\}$, or vice versa.*

Otherwise the algorithm returns a set of four vertices that induces a P_4 or a C_4 in G–xy and a P_4, C_4 or $2K_2$ in the graph G–xy–F where F is an arbitrary set of edges in G that are degree-greater than xy in G.

Proof In order to decide whether the first condition holds, we only have to check the size of I_1 and whether both x and y are contained in the set U_1 of the doubly linked list. For the second condition we only have to check whether the set containing one of the vertices is the successor of the set containing the other vertex in $\mathcal{P}$. These operations can be done in constant time.

Now we show that the edge xy can be removed without leaving the class of threshold graphs if one of the two conditions holds. Note that this result was already proven by Heggernes and Papadopoulos [15]. However, we need a slightly different technical argument for the fully dynamic recognition algorithm, so we present it here again. For the two cases we consider the following vertex partitions:

$$\begin{aligned}\mathcal{P}_1 &= (\{x, y\}, (U_1 \backslash \{x, y\}) \cup I_1, I_2, U_2, \ldots, I_k, U_k, I_{k+1}),\\ \mathcal{P}_2 &= (I_1, U_1, \ldots, I_{j-1}, U_{j-1}, I_j \backslash \{x\}, \{y\}, \{x\}, U_j \backslash \{y\},\\ &\qquad I_{j+1}, U_{j+1}, \ldots, I_k, U_k, I_{k+1}).\end{aligned}$$

Note that in $\mathcal{P}_2$ the vertex x becomes a member of U_{j-1} if $I_j \backslash \{x\}$ is empty and y becomes a member of I_{j+1} if $U_j \backslash \{y\}$ is empty. It follows by definition that $\mathcal{P}_1$ is again a threshold partition of the graph $G - xy$ if Condition 1 holds. Furthermore, $\mathcal{P}_2$ is also a threshold partition of $G - xy$ if Condition 2 holds. To compute theses partitions we only have to remove a constant number of elements from their corresponding lists and put them in lists that are predecessors or successors of the old lists. Therefore, these operations can be done in $\mathcal{O}(1)$.

We will show that for every edge $xy \in E$ that does not fulfill the conditions the graph $G - xy$ is not a threshold graph. To this end, we will show how we can find an induced P_4 or C_4 in $G - xy$. Without loss of generality, we may assume that $y \in U_j$ for some $j \in \{1, \ldots, k\}$.

In the following we consider three cases (see Fig. 2).

Case (i) We first consider the case that x is an element of an independent set I_ℓ. As xy is an edge of G it holds that $\ell \leq j$. Since Condition 2 fails, we have $\ell < j$. Let a be a vertex in U_ℓ and b be a vertex in I_j. Both vertices can be found in constant

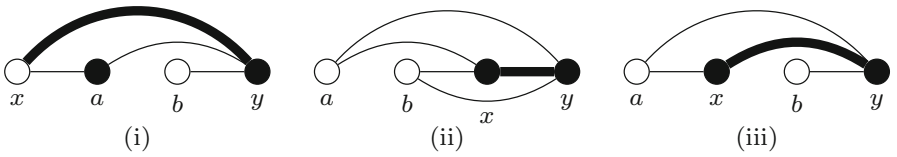


Fig. 2 The three cases of the proof of Lemma 3.5. Black vertices belong to $\bigcup_{i=1}^{k} U_i$ and white vertices belong to $\bigcup_{i=1}^{k} I_i$. The thick edge is the edge we want to delete. The order of the vertices respects the order of their corresponding sets in the threshold partition

time. The set $\{x, a, y, b\}$ induces the following edges: xa, ay, by. Thus, (x, a, y, b) is an induced P_4 in $G - xy$, since x is not adjacent to b, as both x and b are part of an independent set, and a is not adjacent to b because $\ell < j$. Due to Observation 3.2, both xa and by are not degree-greater than xy in G. If we remove arbitrary edges from $G - xy$ which are degree-greater than xy, then the vertices a, b, x and y induce a P_4 or a $2K_2$, depending on whether the edge ay was removed.

For cases (ii) and (iii) we assume that $x \in U_i$ and $i \leq j$, as otherwise we can swap the roles of x and y.

Case (ii) Let $i = j$. If $i = 1$, then I_1 contains at least two vertices because Condition 1 does not hold. Otherwise, there are at least two vertices in $I_1 \cup I_2$. In both cases we have at least two non-adjacent vertices a and b which are adjacent both to x and to y. Thus, the set $\{x, y, a, b\}$ induces a C_4 in $G - xy$. Both a and b can be found in constant time. Furthermore, all the edges contained in this cycle are not degree-greater than edge xy in G, due to Observation 3.2. Therefore, the C_4 is also contained in any graph $G - F$ where F is a set of edges that are degree-greater than xy in G.

Case (iii) Let $i < j$. We pick vertices $a \in I_1$ and $b \in I_j$ in constant time. The edges ax, ay and by are contained in G. Note that x is not adjacent to b since $j > i$ and that a is not adjacent to b since both belong to independent sets. Now (x, a, y, b) is an induced P_4 in $G - xy$. Again, all the edges contained in this path are not degree-greater than edge xy in G. □

For the addition of an edge there is a similar characterization.

Lemma 3.6 *Let $G = (V, E)$ be a threshold graph with a threshold data structure containing threshold partition $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ and $xy \notin E$. There is an algorithm with running time in $\mathcal{O}(1)$ which returns a threshold data structure of $G + xy$ if one of the following conditions hold:*

1. *$x \in I_1$ and $y \in I_1$;*
2. *$I_1 = \{x\}$ and $y \in I_2$, or vice versa;*
3. *$x \in U_j$ and $y \in I_{j+1}$ for some $j \in \{1, \ldots, k\}$, or vice versa.*

Otherwise the algorithm returns a set of four vertices that induces a P_4 or a $2K_2$ in $G + xy$ and a P_4, $2K_2$ or C_4 in the graph $G + xy + F$ where F is an arbitrary set of non-edges of G that are degree-lower than xy in G.

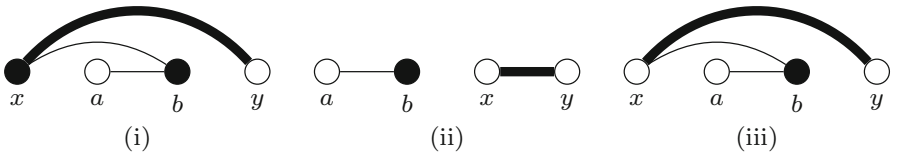


Fig. 3 The cases of the proof of Lemma 3.6. Black vertices belong to $\bigcup_{i=1}^{k} U_i$ and white vertices belong to $\bigcup_{i=1}^{k} I_i$. The thick edge is the edge we want to add. The order of the vertices respects the order of their corresponding sets in the threshold partition

Proof All three conditions can be checked in constant time. Assume first that the edge xy fulfills one of the three conditions. Then consider the following threshold partitions:

$$\mathcal{P}_1 = (\{x\}, \{y\}, I_1 \backslash \{x, y\}, U_1, \ldots, I_k, U_k, I_{k+1})$$
$$\mathcal{P}_2 = (\{z\}, U_1 \backslash \{z\}, \{y\}, \{x\}, I_2 \backslash \{y\}, U_2, \ldots, I_k, U_k, I_{k+1}) \text{ with } z \in U_1$$
$$\mathcal{P}_3 = (I_1, U_1, \ldots, I_j, U_j \backslash \{x\}, \{y\}, \{x\}, I_{j+1} \backslash \{y\}, U_{j+1}, \ldots, I_k, U_k, I_{k+1})$$

If $I_1 \backslash \{x, y\}$ in $\mathcal{P}_1$ is empty, then y becomes an element of U_1. If $U_1 \backslash \{z\}$ is empty in $\mathcal{P}_2$, then we join the sets $\{z\}$ and $\{y\}$. If $I_2 \backslash \{y\}$ in $\mathcal{P}_2$ is empty, x becomes a member of U_2. If $U_j \backslash \{x\}$ is empty in $\mathcal{P}_3$, then $\{y\}$ becomes a member of I_j and if $I_{j+1} \backslash \{y\}$ is empty, then x is added to U_{j+1}. It is not difficult to see that these three partitions form threshold partitions of $G + xy$ for the three different cases, respectively. Just as in the argument in the proof of Lemma 3.5, they can be computed in constant time.

Now assume that xy does not fulfill any of these three conditions. First assume that one of the vertices, say x, is in a set U_i. Since $xy \notin E$ and Condition 3 does not hold, vertex y must be in a set I_j with $j > i + 1$. Let a be a vertex in I_{i+1} and b be a vertex in U_{i+1}. Then, (a, b, x, y) is an induced P_4 in $G + xy$ (see Fig. 3(i)). Note that we can find a and b in constant time since we only have to find the successor of U_i and its successor in the threshold partition $\mathcal{P}$. Furthermore, the only non-edge of the P_4 that is degree-lower than xy in G is the edge ay, due to Observation 3.2. Therefore, in any graph $G + xy + F$ where F contains only edges that are degree-lower than xy in G the vertices a, b, x and y induce a P_4 or C_4, depending on whether ay is in F.

Thus, we can assume that both x and y are elements of I-sets. First assume that both x and y are elements of I_i. Since Condition 1 does not hold, i must be greater than 1. Let $a \in I_1$ and $b \in U_1$. Then the vertices a, b, x, and y induce a $2K_2$ in $G + xy$ (see Fig. 3(ii)) Again, we can find a and b in constant time and all non-edges of the $2K_2$ are not degree-lower than xy in G.

Now consider the case that x and y are in different I-sets. Without loss of generality, we assume that $x \in I_i$ and $y \in I_j$ with $i < j$. Assume first that $i = 1$. If $j = 2$, then, by Condition 2, there is a vertex $a \in I_1 \backslash \{x\}$. In this case let $b \in U_1$. If $j > 2$, than let $a \in I_2$ and $b \in U_2$. In both cases (a, b, x, y) is an induced P_4 in $G + xy$ (see Fig. 3(iii)) and we find a and b in time $\mathcal{O}(1)$. Similar to the first case, ay is the only non-edge of this path that is degree-lower than xy and the insertion of ay turns the P_4 into a C_4 in any $G + xy + F$.

Now assume that $i \neq 1$ and let $a \in I_1$ and $b \in U_1$. The vertices a, b, x and y induce a $2K_2$ in $G + xy$ (see Fig. 3(ii)), which is also a $2K_2$ in any $G + xy + F$ with suitable F. □

Since threshold graphs are hereditary, the deletion of a vertex in a threshold graph always leads to a threshold graph. Therefore, we only have to characterize the case in which a vertex with a given set of neighbors can be added to G such that G is still threshold.

Lemma 3.7 *Let $G = (V, E)$ be a threshold graph with a threshold data structure containing threshold partition $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ and let $U = \bigcup_{j=1}^{k} U_j$. For a vertex $z \notin V$ and a set $N_z \subseteq V$ of vertices in G we consider the graph $G' = (V \cup \{z\}, E \cup \{zv \mid v \in N_z\})$.*

There is an algorithm with running time in $\mathcal{O}(|N_z|)$ which returns a threshold data structure of G' if one of the following conditions hold.

1. *$N_z \subseteq U$ and there is an $1 \leq i \leq k$ such that $\bigcup_{j=i+1}^{k} U_j \subseteq N_z$ and $U_\ell \cap N_z = \emptyset$ for all $1 \leq \ell \leq i - 1$;*
2. *$|I_1| = 1$ and $N_z \subseteq U \cup I_1$ and $\bigcup_{j=2}^{k} U_j \cup I_1 \subseteq N_z$;*
3. *$U \subseteq N_z$ and there is an $1 \leq i \leq k + 1$ such that $\bigcup_{j=1}^{i-1} I_j \subseteq N_z$ and $I_\ell \cap N_z = \emptyset$ for all $i + 1 \leq \ell \leq k + 1$.*

Otherwise the algorithm returns a set of four vertices that induces a P_4, C_4 or $2K_2$ in G'.

Proof We first describe how to check in time $\mathcal{O}(|N_z|)$ whether one of the three conditions hold. We iterate through N_z, mark the contained vertices and count the number of elements of N_z in each set U_i and I_i as well as the total number of elements of N_z in U-sets and I-sets, respectively. Furthermore, when considering an element of N_z we remove it from its list and reinsert it at the front of this list. Therefore, in every list the elements of N_z are at the beginning.

If all elements of N_z are in U-sets, then we have to check whether Condition 1 holds. We iterate through the U-sets starting in U_k and check whether the number of elements of N_z in the respective set is equal to the size of the set. Furthermore, we count how many elements of N_z we have already found in the considered sets. When we have found the first set U_i for which $|U_i| \neq |U_i \cap N_z|$, we check whether we have already found all elements of N_z.

If there is only one vertex in I_1 and this vertex is the only element of an I-set in N_z, then we check Condition 2 in the same way that we have checked Condition 1. Otherwise, we have to check whether Condition 3 holds. We first examine whether for all U-sets their size is equal to the number of elements of N_z in it. Then we iterate through the I-sets starting in I_1 and check the same property. When the first I-set for which this does not hold is found, we check whether we have discovered all elements of N_z so far.

Now we present the threshold partitions of the graph G' belonging to each of the three conditions. For the first condition let $A = U_i \setminus N_z$ and $B = U_i \cap N_z$. The following partition is a threshold partition of G':

$$\mathcal{P}_1 = (I_1, U_1, \ldots, I_{i-1}, U_{i-1}, I_i, A, \{z\}, B, I_{i+1}, U_{i+1}, \ldots, I_k, U_k, I_{k+1}).$$

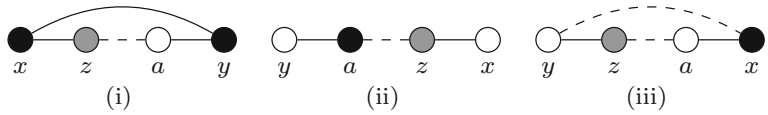


Fig. 4 The three cases of the proof of Lemma 3.7. Black vertices belong to $\bigcup_{i=1}^{k} U_i$ and white vertices belong to $\bigcup_{i=1}^{k} I_i$, the added vertex z is gray. In (a) and (b) the order of the vertices respects the order of their corresponding sets in the threshold partition (apart from the position of z). Dashed edges could be in the graph or not, missing edges are not in the graph

For the second condition let $A = U_1 \setminus N_z$ and $B = U_1 \cap N_z$ and let a be an arbitrary vertex in A. Then the following partition is a threshold partition of G':

$$\mathcal{P}_2 = (\{a\}, A\setminus\{a\}, \{z\}, B \cup I_1, I_2, U_2 \ldots, I_k, U_k, I_{k+1}).$$

If A is empty, then we omit the sets $\{a\}$ and $A\setminus\{a\}$.
For the third condition let $A = I_i \setminus N_z$ and $B = I_i \cap N_z$. In this case

$$\mathcal{P}_3 = (I_1, U_1, \ldots, I_{i-1}, U_{i-1}, B, \{z\}, A, U_i, I_{i+1}, U_{i+1}, \ldots, I_k, U_k, I_{k+1})$$

is a threshold partition of G'. If in any of the three cases the set preceding (or succeeding) $\{z\}$ is empty, then we remove this empty set and put z into the set preceding (or succeeding) this empty set. Due to our sorting of the lists in $\mathcal{P}$, we can find A and B in time $\mathcal{O}(|N_z|)$. This is done by simply iterating through U_i or I_i, removing the elements from U_i or I_i and put them into the new list B as long as they are in N_z. The remaining list is A.

Now, we show how to find the forbidden induced subgraphs in the case that none of the three conditions hold. Assume first that there is a vertex $x \in U_i \cap N_z$ and a vertex $y \in U_j$ with $j > i$ and $y \notin N_z$. Vertex y is found at the end of the first set for which $|U_j| \neq |U_j \cap N_z|$. To find the vertex x we mark each set U_i while iterating through the U-sets. If Condition 1 fails, then we iterate through N_z and find x by checking whether the U-sets of the vertices are marked or not. Additionally, we consider a vertex $a \in I_j$. The set $\{a, x, y, z\}$ induces a P_4 or a C_4 in G' depending on whether a is in N_z or not (see Fig. 4(i)).

Assume that $x \in I_i \cap N_z$ and $y \in I_j \setminus N_z$ with $j < i$. We find both vertices in a similar way as we found x and y in the case above while iterating through the I-sets in order to check Condition 3. For some $a \in U_j$ the set $\{a, x, y, z\}$ induces a P_4 or a $2K_2$ depending on whether a is in N_z or not (see Fig. 4(ii)).

If none of the two cases above occurs, then there must be a vertex $x \in U \setminus N_z$ and a vertex y in N_z which is an element of an I-set. Since Condition 2 does not hold, there must be another vertex a in an I-set with $a \neq y$ and $xa \in E$. We find x by scanning the U-sets starting in U_k and consider the set U_i with largest index that contains a vertex $x \notin N_z$. We choose an arbitrary vertex $y \in N_z \setminus U$ and another vertex a in I_i or I_{i-1}. We know that both yz and ax are edges of G' and neither ay nor zx are edges in G'. Depending on which of the edges xy and az are in G', the set $\{a, x, y, z\}$ induces a P_4 or a C_4 or a $2K_2$ in G' (see Fig. 4(iii)). □

Using the results above, we now present an optimal fully dynamic algorithm for threshold graphs that provides a certificate for the negative case.

Theorem 3.8 *There is a fully dynamic algorithm for the recognition of threshold graphs which provides a set of four vertices inducing a forbidden induced subgraph as certificate in the negative case and has the following costs:*

- *Initialization:* $\mathcal{O}(n + m)$
- *Adding and deleting an edge:* $\mathcal{O}(1)$
- *Adding and deleting a vertex with d neighbors:* $\mathcal{O}(d)$

Proof It follows from Lemmas 3.5, 3.6 and 3.7 that we have an algorithm which allows addition and deletion of edges and the addition of a vertex with d neighbors. It remains to show that this is also possible for the deletion of a vertex. Since threshold graphs are hereditary, we do not have to check anything if we delete a vertex z. The new threshold partition is constructed by simply deleting z from its set. If the set is empty afterwards, then we remove it from the threshold partition and merge its predecessor set with its successor set. If z was in a set U_i we move all vertices from I_i into I_{i+1}. If z was in a set I_i, then we move all vertices from U_i into U_{i-1}. Therefore, only neighbors of z change their set and the whole procedure can be done in $\mathcal{O}(|N(z)|)$. □

4 Chain Graphs

We begin with the definition of a special vertex partition of chain graphs that we will use for this algorithm.

Definition 4.1 Let $G = (A \cup B, E)$ be a chain graph. A *chain partition* of G is an ordered partition $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$ of V with $A_i, B_i \neq \emptyset$ for all $1 \leq i \leq k$ such that $xy \in E$ if and only if $x \in A_i$ and $y \in B_j$ or vice versa with $i \leq j$. In particular, the set I is the set of isolated vertices and can be empty.

In order to be more convenient in this context, this definition has been adapted from a similar concept introduced by Heggernes and Papadopoulos [15]. Note that a graph with a chain partition does not contain a $2K_2$, as for any two edges the vertex that appears first in the ordered partition must be adjacent to both vertices in the other set of the bipartition. Thus, a graph with a chain partition is, in fact, a chain graph, as it is also bipartite by definition. For the existence and computation of a chain partition we use the following result by Hammer et al. [11].

Lemma 4.2 [11] *A bipartite graph is a chain graph if and only if every induced subgraph without isolated vertices has a dominating vertex on each side of the bipartition, that is, a vertex adjacent to all the vertices on the other side of the bipartition.*

For a chain graph we first remove all isolated vertices and insert them into I. For the remaining graph we compute a bipartition (A, B). Due to Lemma 4.2, there must be at least one vertex in B that is adjacent to all vertices in A. These vertices form the elements of B_k. After the deletion of B_k there are isolated vertices in A which form

the set A_k. These are then also removed from the graph. Repeating this procedure until the graph is empty leads to a chain partition. For our fully dynamic algorithm we use the following data structure.

Definition 4.3 *(Chain data structure)* For a chain graph $G = (V, E)$ a *chain data structure* consists of the following elements:

- a chain partition $\mathcal{P}$ of G as a doubly linked list of doubly linked lists
- a pointer from every vertex $v \in V$ to its list L in $\mathcal{P}$ and to its corresponding element in L
- for every list in $\mathcal{P}$ the information whether it stands for an A-set or a B-set or the set I

Similar to the results for threshold graphs, we now present algorithms for the deletion and addition of edges as well as the addition of a vertex with a set of neighbors. These algorithms will either return a new chain data structure or they will return an induced $2K_2$ or an odd cycle contained in the new graph, showing that it is not a chain graph.

Lemma 4.4 *Let $G = (V, E)$ be a chain graph with a chain data structure containing chain partition $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$ and let $xy \in E$. There is an algorithm with running time in $\mathcal{O}(1)$ which returns a chain data structure of $G - xy$ if $x \in A_i$ and $y \in B_i$ or vice versa.*

Otherwise, the algorithm returns a set of four vertices that induces a $2K_2$ in $G - xy$ and any graph $G - xy - F$ where F is an arbitrary set of edges in G that are degree-greater than xy in G.

Proof In order to check the condition we only have to compare the successor of the set of x to the set of y in the chain partition and vice versa. This can be done in constant time. If $x \in A_i$ and $y \in B_i$, then consider the following partition:

$$\mathcal{P}' = (A_1, B_1, \ldots, A_{i-1}, B_{i-1}, A_i \backslash \{x\}, \{y\}, \{x\}, B_i \backslash \{y\}, A_{i+1}, B_{i+1}, \ldots, A_k, B_k, I)$$

If $A_i \backslash \{x\}$ is empty, then y becomes a member of B_{i-1} and if $B_i \backslash \{y\}$ is empty, then x becomes a member of A_{i+1} (or I if $i = k$). It is not difficult to see that $\mathcal{P}'$ is a chain partition of $G - xy$ and can be constructed in time $\mathcal{O}(1)$.

On the other hand, assume that xy does not fulfill the condition, i.e., without loss of generality we may assume that $x \in A_i$ and $y \in B_j$ with $j > i$. Consider vertices $w \in B_i$ and $z \in A_j$. Then the set $\{w, x, y, z\}$ induces a $2K_2$ in $G - xy$. Since $d_G(z) < d_G(x)$ and $d_G(w) < d_G(y)$, this $2K_2$ is contained in any graph $G - xy - F$ where F only contains edges that are degree-greater than xy. □

Lemma 4.5 *Let $G = (V, E)$ be a chain graph with a chain data structure containing chain partition $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$ and let $xy \notin E$. There is an algorithm with running time in $\mathcal{O}(1)$ which returns a chain data structure of $G + xy$ if one of the following conditions holds:*

1. $x \in A_i$ *and* $y \in B_{i-1}$ *or vice versa*
2. $x \in A_1$ *and* $y \in I$ *or vice versa*
3. $x \in I$ *and* $y \in B_k$ *or vice versa*
4. $I = V$

Otherwise, the algorithm returns an induced $2K_2$ *or a* C_3 *contained in* $G + xy$. *In this case let* F *be an arbitrary set of non-edges of* G *that are degree-lower than* xy *in* G. *Then the algorithm can find an induced* $2K_2$ *or a* C_3 *contained in* $G' = G + xy + F$ *in time* $\mathcal{O}(|F|)$.

Proof One can check whether one of the conditions holds in $\mathcal{O}(1)$ using the information given by the chain data structure. Consider the following partitions:

$$\begin{aligned}
\mathcal{P}_1 &= (A_1, B_1, \ldots, A_{i-1}, B_{i-1}\backslash\{y\}, \{x\}, \{y\}, A_i\backslash\{x\}, B_i,\\
&\qquad A_{i+1}, B_{i+1}, \ldots, A_k, B_k, I);\\
\mathcal{P}_2 &= (\{x\}, \{y\}, A_1\backslash\{x\}, B_1, A_2, B_2, \ldots, A_k, B_k, I\backslash\{y\});\\
\mathcal{P}_3 &= (A_1, B_1, \ldots, A_k, B_k\backslash\{y\}, \{x\}, \{y\}, I\backslash\{x\});\\
\mathcal{P}_4 &= (\{x\}, \{y\}, I\backslash\{x, y\}).
\end{aligned}$$

Similar to the proofs of Lemmas 3.5–3.7, we remove empty sets from these partitions and add x and y to the preceding or succeeding set. It follows from the structure of the chain graphs, as given above, that partition $\mathcal{P}_i$ is a chain partition of graph $G + xy$ if Condition i holds. Furthermore, all partitions can be constructed in time $\mathcal{O}(1)$.

For the reverse, first consider the case that $x, y \in I$ but $I \neq V$. Then let $w \in A_1$ and $z \in B_1$. The set $\{w, x, y, z\}$ induces a $2K_2$ in $G + xy$ and any missing edge in this $2K_2$ is not degree-lower than xy in G. If both x and y are elements of A-sets, then we choose a vertex $z \in B_k$. If both x and y are elements of B-sets, then we choose a vertex $z \in A_1$. In both cases the set $\{x, y, z\}$ induces a C_3 in $G + xy$ and any graph containing $G + xy$, showing that they are not bipartite and, thus, not chain graphs.

If $x \in A_i$ and $y \in B_j$ with $j < i - 1$, then x and y together with vertices $w \in A_{i-1}$ and $z \in B_{i-1}$ induce a $2K_2$ in $G + xy$. Assume $x \in I$. If $y \in B_i$ with $i < k$, then a $2K_2$ is induced by x and y and the vertices $w \in A_k$ and $z \in B_k$. If $y \in A_i$ with $i > 1$, then the $2K_2$ is induced by x, y and the vertices $w \in A_1$ and $z \in B_1$. All these vertices can easily be found in constant time. The only missing edges of the $2K_2$ that could be degree-lower than the edge xy in G are the edges yz and wx. However, these edges connect two vertices of A or B, respectively. Therefore, if we find such an edge in the set F, then we can find a C_3 contained in $G + xy + F$ using the steps described above. □

Lemma 4.6 *Let* $G = (V, E)$ *be a chain graph with a chain data structure containing chain partition* $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$. *For a vertex* $z \notin V$ *and a set* $N_z \subseteq V$ *of vertices in* G *we consider the graph* $G' = (V \cup \{z\}, E \cup \{zv \mid v \in N_z\})$.

There is an algorithm with running time in $\mathcal{O}(|N_z|)$ *which returns a chain data structure of* G' *if one of the following conditions holds.*

1. $N_z = \emptyset$;
2. $\bigcup_{j=1}^{k} A_j \subseteq N_z$ *and* $B_j \cap N_z = \emptyset$ *for all* $1 \leq j \leq k$;

3. $\bigcup_{j=1}^{k} B_j \subseteq N_z$ *and* $A_j \cap N_z = \emptyset$ *for all* $1 \leq j \leq k$*;*
4. *there is an* $1 \leq i \leq k$ *such that* $\bigcup_{j=1}^{i-1} A_j \subseteq N_z$ *and* $A_\ell \cap N_z = \emptyset$ *for all* $k \geq \ell > i$*,* $B_j \cap N_z = \emptyset$ *for all* $1 \leq j \leq k$ *and* $I \cap N_z = \emptyset$*;*
5. *there is an* $1 \leq i \leq k$ *such that* $\bigcup_{j=i+1}^{k} B_j \subseteq N_z$ *and* $B_\ell \cap N_z = \emptyset$ *for all* $1 \leq \ell < i$*,* $A_j \cap N_z = \emptyset$ *for all* $1 \leq j \leq k$ *and* $I \cap N_z = \emptyset$*.*

Otherwise the algorithm returns an induced C_3 *or* $2K_2$ *contained in* G' *or a set of five vertices inducing a subgraph of* G' *that contains a* C_5*.*

Proof To check these conditions, we iterate through N_z and count the number of elements that it has in each set in the chain partition. Furthermore, when considering an element of N_z we remove it from its list and reinsert it at the front of this list. Therefore, in every list the elements of N_z are at the beginning. If there are vertices from both A- and B-sets in N_z, then none of the conditions hold. If there are only vertices of A-sets (and maybe vertices of I), then we iterate through the A-sets starting in A_1 and search for the first set where the size of the set is not equal to the number of vertices of N_z in this set. At this point we must have found all elements of N_z, as otherwise none of the conditions hold. If no such set exists, then Condition 2 holds. For the case that N_z only contains elements of B-sets (and maybe I) we do the same, with the difference that we start in B_k. All these steps can be done in $\mathcal{O}(|N_z|)$ since the number of sets that we have to consider is bounded by $\mathcal{O}(|N_z|)$.

Assume that z fulfills one of the conditions of the lemma. If $N_z = \emptyset$, then we simply insert z into I. For the other conditions we consider the following partitions:

$$\begin{aligned}
\mathcal{P}_2 &= (A_1, B_1, \ldots, A_k, B_k, I \cap N_z, \{z\}, I \backslash N_z),\\
\mathcal{P}_3 &= (\{z\}, I \cap N_z, A_1, B_1, \ldots, A_k, B_k, I \backslash N_z),\\
\mathcal{P}_4 &= (A_1, B_1, \ldots, A_{i-1}, B_{i-1}, N_z \cap A_i, \{z\}, A_i \backslash N_z, B_i,\\
&\qquad A_{i+1}, B_{i+1}, \ldots, A_k, B_k, I),\\
\mathcal{P}_5 &= (A_1, B_1, \ldots, A_{i-1}, B_{i-1}, A_i, B_i \backslash N_z, \{z\}, B_i \cap N_z,\\
&\qquad A_{i+1}, B_{i+1}, \ldots, A_k, B_k, I).
\end{aligned}$$

In the partition $\mathcal{P}_4$ vertex z is added to B_{i-1} if $N_z \cap A_i$ is empty and to B_i if $A_i \backslash N_z$ is empty. In the partition $\mathcal{P}_5$ vertex z is added to A_i if $B_i \backslash N_z$ is empty and to A_{i+1} if $B_i \cap N_z$ is empty. The partition $\mathcal{P}_i$ is a chain partition of G' if Condition i holds. Furthermore, we can compute these partitions in time $\mathcal{O}(|N_z|)$.

Now assume that z does not fulfill any of the conditions. If there are vertices of N_z in both A-sets and B-sets, then we consider different cases. If there is a vertex $x \in A_1 \cap N_z$, then we choose an arbitrary vertex y in N_z which is an element of a B-set and $\{x, y, z\}$ induces a C_3 in G'. If there is a vertex in $x \in B_k \cap N_z$, then this vertex x together with z and an arbitrary vertex y in $A_i \cap N_z$ induces a C_3 in G'. If none of these two cases occurs, then we choose arbitrary vertices in $x \in N_z \cap A_i$ and $y \in N_z \cap B_j$. Additionally, we choose a vertex $a \in A_1$ and $b \in B_k$. The set $\{a, b, x, y, z\}$ induces a subgraph of G' that contains a C_5.

Now assume that there are vertices $a \in A_i \cap N_z$ (or $a \in I \cap N_z$) and $b \in A_j \backslash N_z$ with $j < i$ (or $\leq k$). We find b while iterating through the A-sets and afterwards

a while iterating through N_z and check whether the sets of the vertices are already marked. Let c be an arbitrary vertex in B_j. Then the set $\{a, b, c, z\}$ induces a $2K_2$ in G'. Analogously, we find a $2K_2$ if $a \in B_i \cap N_z$ (or $a \in I \cap N_z$) and $b \in B_j \setminus N_z$ with $j > i$ (or $\leq k$) by considering a vertex $c \in A_j$. □

Using these results, we can state a fully dynamic recognition algorithm for chain graphs.

Theorem 4.7 *There is a fully dynamic algorithm for the recognition of chain graphs which provides a set of at most five vertices inducing a forbidden subgraph as certificate in the negative case and has the following costs:*

- *Initialization:* $\mathcal{O}(n + m)$
- *Adding and deleting an edge:* $\mathcal{O}(1)$
- *Adding and deleting a vertex with d neighbors:* $\mathcal{O}(d)$

Proof Due to Lemmas 4.4, 4.5 and 4.6, it only remains to show that our algorithm also works for the deletion of a vertex. Since chain graphs are hereditary, we do not have to check anything if we delete a vertex z and the new chain partition is constructed by simply deleting z from its set. If the set of z is empty afterwards, then we remove this set from the chain partition and merge its predecessor set with its successor set. Similar to the proof of Theorem 3.8, we do this in such a way that only neighbors of z are moved. Therefore, this procedure can be done in time $\mathcal{O}(|N(z)|)$. □

5 Adding and Deleting Multiple Vertices and Edges

In some applications it is important to not only delete or add one edge (vertex) at a time but rather sets of edges (vertices). This problem cannot always be solved by regular dynamic algorithms. For example, when dealing with the vertex connectivity problem we cannot delete vertices in an arbitrary order while maintaining connectivity. However, if we have a set of vertices after whose deletion the graph is still connected, then we can find an order of deletions of these vertices that maintains connectivity. On the other hand, if we consider edge deletion in Eulerian graphs, then it is obvious that no *one* edge can be deleted without leaving the class of Eulerian graphs. For the graph classes considered in this paper, i.e., threshold and chain graphs, we will see that it is always possible to add and delete sets of edges (vertices) one at a time by finding a suitable order.

Since both threshold graphs and chain graphs are hereditary, the deletion of a set of vertices can be done iteratively in any order. The addition of vertices together with incident edges is also simple. If the graph that results from this addition is still in the respective graph class, then we can delete the vertices from this larger graph in an arbitrary order and all intermediate graphs are in the respective graph class. The reverse of this deletion scheme is an addition scheme of the vertices. Therefore, any ordering of the vertices to add is sufficient. Furthermore, a forbidden induced subgraph found by the algorithms described in Lemmas 3.7 and 4.6 remains an induced subgraph, even if we add further vertices together with edges incident to at least one of these vertices.

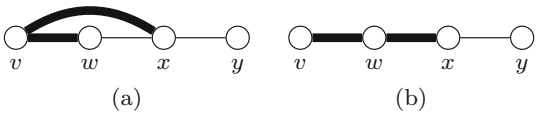


Fig. 5 In (a) the graphs with and without the thick edges are threshold graphs. However, if we delete the edge vx from the larger graph or add the edge vw to the smaller graph, then the result is not a threshold graph. In (b) both graphs are chain graphs. The deletion of wx from the larger graph or the addition of vw to the smaller graph results in a graph that is not a chain graph

Therefore, we can easily adapt our certifying dynamic algorithms to the addition and deletion of whole sets of vertices (and incident edges).

For sets of edges, the problem is more difficult. In Fig. 5we present two examples which show that we cannot choose an arbitrary ordering in which we add or delete the edges iteratively since there are orderings where the intermediate graphs are not threshold or chain graphs anymore. However, in both examples there is an ordering where the intermediate graphs are in the respective class. This motivates the following definition: A graph class $\mathcal{C}$ is called *sandwich monotone* if for any graph $G = (V, E)$ with $G \in \mathcal{C}$ and any set $F \subseteq E$ with $G - F \in \mathcal{C}$ it holds that there is an ordering $(e_1, \ldots, e_k)$ of the edges in F such that for any $i \in \{1, \ldots, k\}$ the graph $G - \{e_1, \ldots, e_i\}$ is in $\mathcal{C}$. Equivalently, we can say that F must contain an edge $e \in F$ such that $G - e \in \mathcal{C}$. From this definition it follows directly that the same holds for the addition of edges, i.e., if $G \in \mathcal{C}$ and $G + F \in \mathcal{C}$, then there is an edge $e \in F$ for which $G + e$ is in $\mathcal{C}$.

This property was already studied in 1976 by Rose et al. [26], who showed that chordal graphs fulfill the property. The term sandwich monotonicity was introduced in 2007 by Heggernes and Papadopoulos [14]. The same authors also proved that both threshold and chain graphs are sandwich monotone.

Theorem 5.1 [15] *Threshold and chain graphs are sandwich monotone.*

This result leads to a dynamic algorithm that decides whether a set of edges can be added to or deleted from a threshold graph (chain graph) such that the graph is still a threshold graph (chain graph). We iterate through the edges that are not already added or deleted and check with the algorithms of Lemmas 3.5, 3.6, 4.4 and 4.5 whether they are a safe choice. If we do not find such an edge, then the graph that results from the addition or deletion of all edges of the set is not a threshold or chain graph. This algorithm needs time $\mathcal{O}(k^2)$ to delete or add k edges. In the following, we will present a more sophisticated approach that only needs time $\mathcal{O}(\min\{k \log k, n + k\})$. In the negative case, the presented algorithms return a forbidden induced subgraph of the graph that results from the addition or deletion of all edges.

Beisegel et al. [1] showed that for any threshold graph (chain graph) and a special partition $(E_1, \ldots, E_k)$ of the edges of the graph one can compute an elimination ordering of the edges of the graph in linear time such that the edges within a set E_i are consecutive and the graph never leaves its respective graph class during the elimination process.

The difference to the problem discussed here is that all edges of the graph are deleted and, as a result, the running time must be dependent on $n + m$. Here, we only delete k edges and ask whether this can be done faster than $n + k$. Furthermore, we want

to give a certificate in the negative case which was not considered in [1]. However, it will be possible to follow a similar approach using so-called degree-minimal edge elimination schemes.

Definition 5.2 Let $G = (V, E)$ be a graph and $F \subseteq E$ be a set of edges of G. A *degree-minimal edge elimination scheme of F from G* is an ordering $(e_1, \ldots, e_k)$ of the edges in F such that for any $i \in \{1, \ldots, k\}$ it holds that the edge e_i is a degree-minimal edge of $F \setminus \{e_1, \ldots, e_{i-1}\}$ for the graph $G - \{e_1, \ldots, e_{i-1}\}$.

In [1], Beisegel et al. showed the following theorem.

Theorem 5.3 [1] *Let $G = (V, E)$ be a threshold graph (chain graph). Let $F \subseteq E$ be a subset of the edges of G such that $G - F$ is a threshold graph (chain graph). Then for any degree-minimal edge e in F it holds that $G - e$ is a threshold graph (chain graph).*

This immediately implies the following corollary.

Corollary 5.4 *Let $G = (V, E)$ be a threshold graph (chain graph) and $F \subseteq E$. Then, $G - F$ is a threshold graph (chain graph) if and only if for any degree-minimal edge elimination scheme $(e_1, \ldots, e_k)$ of F from G it holds that $G - \{e_1, \ldots, e_i\}$ is a threshold graph (chain graph) for any $i \in \{1, \ldots, k\}$.*

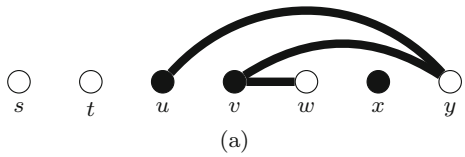
Beisegel et al. [1] showed that for any graph we can compute a degree-minimal elimination scheme of all the edges of the graph in time $\mathcal{O}(n + m)$.

Theorem 5.5 [1] *Given a graph $G = (V, E)$, there is an algorithm that computes a degree-minimal edge elimination scheme of E from G in time $\mathcal{O}(n + m)$.*

An important step in this algorithm is the sorting of up to m objects by a degree value in time $\mathcal{O}(n + m)$ using counting sort. As here it is only necessary to sort up to k objects we can sort these objects in $\mathcal{O}(k \log k)$. This allows us to reach an overall time bound of $\mathcal{O}(\min\{k \log k, n + k\})$. We will give a short overview of the steps of the algorithm. For an exhaustive analysis we refer to [1].

First we identify all vertices that are incident to an edge in F. Then we sort these vertices due their degrees in G. This takes $\mathcal{O}(\min\{k \log k, n + k\})$ many steps. Now, we remove the edges of F incident to the first vertex of our ordering and update the ordering of the vertices in time linear in the number of deleted edges. Iterating this process leads to an edge elimination scheme. In order to turn this elimination scheme into a degree-minimal edge elimination scheme, we have to sort the edges that are deleted at the same vertex by the degree of their second vertex. This can be done in $\mathcal{O}(\min\{k \log k, n + k\})$ when sorting all edges together.

Theorem 5.6 *Given a threshold graph (chain graph) $G = (V, E)$ with a threshold (chain) data structure and a set $F \subseteq E$. There is an algorithm that returns either a threshold (chain) data structure of $G - F$ or returns a constant number of vertices of G that induces a forbidden subgraph in $G - F$. This algorithm needs time $\mathcal{O}(\min\{k \log k, n + k\})$.*



(a)

Fig. 6 This example shows why the usage of the reverse of a degree-minimal edge elimination scheme as an edge addition scheme is a problem. Black vertices belong to $\bigcup_{i=1}^{k} U_i$ and white vertices belong to $\bigcup_{i=1}^{k} I_i$. The order of the vertices respects the order of their corresponding sets in the threshold partition. The edges of the graph are not depicted but given implicitly by the threshold partition. The thick edges are elements of the set F, which we want to add to the graph

Proof As described above, we first compute a degree-minimal edge elimination scheme of F from G in time $\mathcal{O}(\min\{k \log k, n+k\})$. Then we delete the edges following this scheme using the algorithms given by Lemmas 3.5 and 4.4. If the graph $G - F$ is still a threshold (chain) graph, then this procedure results in a threshold (chain) data structure of $G - F$.

It remains to show that otherwise the returned vertices induce a forbidden subgraph in $G - F$. If the algorithms described in Lemmas 3.5 and 4.4 fail to remove the edge e, then all edges in F that are not already deleted are degree-greater than e in the currently considered graph. Therefore, it follows from Lemmas 3.5 and 4.4 that the returned vertices also induce a forbidden subgraph in $G - F$. □

For the addition of the edges contained in a set F but not in the graph G we could use the following algorithm: We compute a degree-minimal edge elimination scheme of the set F from the graph $G + F$. If $G + F$ is a threshold graph (chain graph), then all intermediate graphs of this elimination scheme must be threshold graphs (chain graphs). Therefore, we can use the reverse ordering as an edge addition scheme of the set F to the graph G. If $G + F$ is not threshold (not chain), then in the reverse ordering there must be an edge which cannot be added to the graph such that it remains a threshold graph (chain graph). However, the forbidden subgraph returned by the algorithm proposed in Lemmas 3.6 and 4.5 does not have to be an induced subgraph of $G + F$. An example for threshold graphs is given in Fig. 6. The threshold graph G is given by its threshold partition. The edge set F is marked with thick lines. The graph $G + F$ is not threshold. A possible degree-minimal edge elimination scheme of F from $G + F$ is (vw, uy, vy). Therefore, our addition scheme would add vy to G first. The algorithm of Lemma 3.6 would reject this addition and would return the path (w, x, v, y) which is an induced P_4 in $G + vy$. However, the four vertices do not induce a forbidden subgraph in $G + F$ since the edge vw is in F.

To solve this problem we consider degree-maximal addition schemes.

Definition 5.7 Let $G = (V, E)$ be a graph and F be a set of non-edges of G. A *degree-maximal edge addition scheme of F to G* is an ordering $(e_1, \ldots, e_k)$ of the edges in F such that for any $i \in \{1, \ldots, k\}$ it holds that the edge e_i is a degree-maximal edge of $F \setminus \{e_1, \ldots, e_{i-1}\}$ for the graph $G + \{e_1, \ldots, e_{i-1}\}$.

We will now show that in such an addition scheme every intermediate graph is a threshold or chain graph if the initial and the final graph are threshold or chain graphs.

The following observation for degree-maximal edges will be helpful for this. It follows directly from the definition of degree-maximal edges.

Observation 5.8 *Let G be a graph, F be a set of edges of G, and xy a degree-maximal edge in F. Then, for every edge $xz \in F$, we have $d_G(z) \leq d_G(y)$.*

In the following two propositions we will show that a degree-maximal edge in F can be added to a threshold or chain graph without leaving the respective graph class.

Proposition 5.9 *Let $G = (V, E)$ be a threshold graph and F be a set of non-edges of G such that $G + F$ is a threshold graph. Then for any degree-maximal edge $e \in F$ it holds that $G + e$ is a threshold graph.*

Proof Let $e = xy$ and assume for contradiction that $G + e$ would not be a threshold graph. Then there must be an induced P_4, $2K_2$ or C_4 in $G + e$. If the addition of e leads to an induced C_4 in G, then G must contain an induced P_4, which is a contradiction to the fact that G is a threshold graph. Thus, we can assume that $G + e$ contains a P_4 or $2K_2$. First we consider the case of a P_4. Without loss of generality, we may assume that the path has the form (a, b, x, y) since G does not contain a $2K_2$. Since $G + F$ is a threshold graph, F must contain ax or by. Due to Lemma 2.1, it holds that $d_G(b)$ is larger than both $d_G(a)$ and $d_G(x)$ since b is adjacent to both a and x in G, whereas a and x are not adjacent. Furthermore, $d_G(y)$ is smaller than $\min\{d_G(a), d_G(x)\}$ since y is not adjacent to b. Thus, xy cannot be degree-maximal in F, due to Observation 5.8.

Now assume that $G + e$ contains an induced $2K_2$ containing the edges ab and xy. Due to Lemma 2.1, both $d_G(a)$ and $d_G(b)$ are greater than $\max\{d_G(x), d_G(y)\}$. Since $G + F$ is a threshold graph, F contains an edge with one endpoint in $\{a, b\}$ and one endpoint in $\{x, y\}$. Thus, the edge xy is not degree-maximal in F, due to Observation 5.8. □

Proposition 5.10 *Let $G = (A \cup B, E)$ be a chain graph and F be a set of non-edges of G such that $G + F$ is a chain graph. Then for any degree-maximal edge $e \in F$ it holds that $G + e$ is a chain graph.*

Proof Let $e = xy$ and assume for contradiction that $G + e$ is not a chain graph. If $G + e$ contains an odd cycle, then there is also an odd cycle in $G + F$; a contradiction. Therefore, we can assume that $G + e$ contains a $2K_2$ containing the edges ab and xy with $a, x \in A$ and $b, y \in B$. Since $G + F$ does not contain an induced $2K_2$, either xb or ay must be part of F. Due to Lemma 2.1, it holds that $d_G(x) < d_G(a)$ and $d_G(y) < d_G(b)$. Due to Observation 5.8, xy was not degree-maximal in F; a contradiction. □

In order to compute a degree-maximal edge addition scheme using only $\mathcal{O}(\min\{k \log k, n + k\})$ many steps, we can use a modified version of the algorithm that computes the degree-minimal edge elimination scheme. The following theorem follows from Lemmas 3.6 and 4.5 as well as Propositions 5.9 and 5.10.

Theorem 5.11 *Let $G = (V, E)$ be a threshold graph (chain graph) with a threshold (chain) data structure and let F be a set of non-edges of G. There is an algorithm*

that returns either a threshold (chain) data structure of $G + F$ or returns a constant number of vertices of G that induce a forbidden subgraph in $G + F$. This algorithm runs in time $\mathcal{O}(\min\{k \log k, n + k\})$.

Proof Similar to Theorem 5.6, we compute a degree-maximal addition scheme of F to G and then add the edges of F one by one following this scheme using the algorithms given by Lemma 3.6 and 4.5. Following from Propositions 5.9 and 5.10, the algorithms return a threshold (chain) data structure of $G + F$ if and only if $G + F$ is a threshold (chain) graph.

If the respective algorithm fails to add an edge e, then $G + F$ is not a threshold (chain) graph. If G is a threshold graph, then, due to Lemma 3.6, the returned set of vertices induces a forbidden subgraph in $G + F$ since the edges following in the addition scheme are degree-lower than edge e. If G is a chain graph, then by Lemma 4.5 the algorithm can find a set of vertices, that induced a forbidden subgraph in $G + F$ needing time $\mathcal{O}(|F|)$. □

6 Fully Dynamic Algorithms for Hamiltonian Cycles and Paths

In this section we present fully dynamic algorithms that decide for a given threshold graph or a given chain graph whether it has a Hamiltonian cycle or a Hamiltonian path. Similar to the algorithms in the former sections these algorithms can handle updates until the graph leaves the corresponding graph class. However, the graph is not forced to stay Hamiltonian during the process, i.e., some of the intermediate graphs can have a Hamiltonian cycle or path while others have no such cycle or path.

We will first characterize when a threshold graph or a chain graph has such a cycle or path using the threshold partition and chain partition. Afterwards, we will explain how we can use these characterizations algorithmically. Note that threshold graphs with Hamiltonian cycles have already been characterized by Harary and Peled [12].

The following observation holds for all graphs and is well known.

Observation 6.1 *Let $G = (V, E)$ be a graph and $\emptyset \subseteq S \subset V$. If G contains a Hamiltonian cycle, then $G - S$ has at most $\max\{1, |S|\}$ components. If G contains a Hamiltonian path, then $G - S$ has at most $|S| + 1$ components.*

The following four lemmas characterize threshold and chain graphs that contain a Hamiltonian cycle or path. The proofs of the lemmas are constructive, i.e., we will explain how to find the cycle or path in the positive case and we will give a cut set contradicting Observation 6.1 for the negative case. For both cases we will show that the respective certificate can be found in time $\mathcal{O}(n)$. We start by characterizing those threshold graphs which contain a Hamiltonian path.

Lemma 6.2 *Let $G=(V, E)$ be a threshold graph and let $\mathcal{P}=(I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ be the corresponding threshold partition. The graph G has a Hamiltonian path if and only if the following conditions hold:*

1. $I_{k+1} = \emptyset$;
2. $\sum_{j=i}^{k} |I_j| \leq \sum_{j=i}^{k} |U_j| \; \forall i \in \{2, \ldots, k\}$;

3. $(\sum_{j=1}^{k} |I_j|) - 1 \leq \sum_{j=1}^{k} |U_j|$.

There is an algorithm that finds a Hamiltonian path or a set $S \subset V$ where $G - S$ has more than $|S| + 1$ components in time $\mathcal{O}(n)$.

Proof Assume that the three statements hold. We construct the Hamiltonian path in the following way: We start in a vertex of I_k and follow a path that alternates between the vertices in I_k and U_k. Since $|I_k| \leq |U_k|$, this path visits all vertices in I_k and ends in a vertex of U_k. Using Conditions 2 and 3 we can visit all vertices in I-sets in that way. If there are still unvisited vertices in U-sets, we can visit them in an arbitrary order. Otherwise, the Hamiltonian path ends in the last vertex of I_1. This procedure can easily be implemented in time $\mathcal{O}(n)$ using two pointers that iterate through all I-sets and U-sets, respectively.

Now assume that one of the three statements does not hold. If $I_{k+1} \neq \emptyset$, then G is not connected and cannot have a Hamiltonian path. In this case, we return $S = \emptyset$. If the second statement does not hold, then there is an $i \in \{2, \ldots, k\}$ with $\sum_{j=i}^{k} |I_j| > \sum_{j=i}^{k} |U_j|$. Let $U^* = \bigcup_{j=i}^{k} U_j$ and $I^* = \bigcup_{j=i}^{k} I_j$. Since $N(I^*) = U^*$ and I^* induces an independent set, the graph $G - U^*$ has at least $|I^*| + 1$ components (every vertex in I^* and the component containing I_1). Therefore, $S = U^*$ shows the non-existence of a Hamiltonian path, due to Observation 6.1. If the third statement does not hold we choose $S = \bigcup_{j=1}^{k} U_j$. The graph $G - S$ contains at least $|S| + 2$ components, since $|\bigcup_{j=1}^{k} I_j| - 1 > |S|$. The construction of S can straightforwardly be done in time $\mathcal{O}(n)$. □

There is a similar characterization for Hamiltonian cycles in threshold graphs.

Lemma 6.3 *Let $G = (V, E)$ be a threshold graph and let $\mathcal{P} = (I_1, U_1, \ldots, I_k, U_k, I_{k+1})$ be the corresponding threshold partition. The graph G has a Hamiltonian cycle if and only if the following conditions hold:*

1. $I_{k+1} = \emptyset$;
2. $(\sum_{j=i}^{k} |I_j|) + 1 \leq \sum_{j=i}^{k} |U_j| \; \forall i \in \{2, \ldots, k\}$;
3. $\sum_{j=1}^{k} |I_j| \leq \sum_{j=1}^{k} |U_j|$.

There is an algorithm that finds a Hamiltonian cycle or a set $S \subset V$ where $G - S$ has more than $\max\{1, |S|\}$ *components in time $\mathcal{O}(n)$.*

Proof Assume that the three statements hold. We construct a Hamiltonian cycle as follows: We start in a vertex s of U_k. Now we follow a path which alternates between the vertices in I_k and U_k. Since $|I_k| + 1 \leq |U_k|$, this path visits all vertices in I_k and ends in a vertex in U_k. Now we visit a vertex in I_{k-1} next and alternate between vertices in I_{k-1} and unvisited vertices in $U_{k-1} \cup U_k$. Similar to the argument before, we can visit all elements in I_{k-1} and end in a vertex in $U_{k-1} \cup U_k$. Due to Conditions 2 and 3, it is possible to continue this procedure until we have visited the last vertex of I_1. If there are still unvisited vertices in U-sets, then we can visit them in an arbitrary order. Afterwards we go back to the starting vertex s.

Now assume that one of the three conditions does not hold. If $I_{k+1} \neq \emptyset$, the graph is not connected and we can return $S = \emptyset$. If the second statement does not

hold for $i \in \{2, \ldots, k\}$, we choose set $S = \bigcup_{j=i}^{k} U_j$. In $G - S$ every element of $I^* = \bigcup_{j=i}^{k} I_j$ is isolated. Since $|I^*| + 1 > |S|$ and the elements of I_1 and U_1 form their own component, $G - S$ has more than $|S|$ components. Due to Observation 6.1, G does not contain a Hamiltonian cycle. If the third statement does not hold, we choose $S = \bigcup_{j=1}^{k} U_j$. Then $G - S$ contains more than $|S|$ components.

Similar to the proof of Lemma 6.2, both the construction of the Hamiltonian cycle and of the set S can be implemented in time $\mathcal{O}(n)$. □

Due to the close relation between threshold and chain graphs, there are similar characterizations for chain graphs which have a Hamiltonian cycle or a Hamiltonian path.

Lemma 6.4 *Let $G = (V, E)$ be a chain graph and let $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$ be the corresponding chain partition. The graph G has a Hamiltonian path if and only if the following conditions hold:*

1. $I = \emptyset$;
2. $\sum_{j=i}^{k} |A_j| \leq \sum_{j=i}^{k} |B_j| \; \forall i \in \{2, \ldots, k\}$;
3. $\sum_{j=1}^{i} |A_j| \geq \sum_{j=1}^{i} |B_j| \; \forall i \in \{1, \ldots, k-1\}$;
4. $(\sum_{j=1}^{k} |A_j|) - 1 \leq \sum_{j=1}^{k} |B_j| \leq (\sum_{j=1}^{k} |A_j|) + 1$.

There is an algorithm that finds a Hamiltonian path or a set $S \subset V$ where $G - S$ has more than $|S| + 1$ components in time $\mathcal{O}(n)$.

Proof Assume that all four statements hold. First we consider the case that $\sum_{j=1}^{k} |B_j| \leq \sum_{j=1}^{k} |A_j|$. We start in a vertex of A_k. Due to Condition 2, we can visit all vertices in $\bigcup_{j=2}^{k} A_j$ by alternating between these vertices and their neighbors in the sets $B_2, \ldots, B_k$ and end in a vertex of B_2. The number of visited vertices in A-sets and B-sets is identical at this point. Due to Condition 4, the number of unvisited vertices in B-sets is either $|A_1| - 1$ or $|A_1|$. Therefore, we can visit all vertices in A_1 and all remaining vertices in B.

It remains to show that for $\sum_{j=1}^{k} |B_j| = \sum_{j=1}^{k} |A_j| + 1$ there is also a Hamiltonian path. In this case, it holds that $|B_k| > |A_k|$ due to Condition 3 for $i = k - 1$. Therefore, we can start in a vertex of B_k and visit all vertices of A_k. Following the same argument as in the first case, we find a Hamiltonian path.

Now assume that G does not fulfill one of the statements. If $I \neq \emptyset$, then G is connected and we can return $S = \emptyset$. If the second statement does not hold for $i \in \{2, \ldots, k\}$, then we choose $S = \bigcup_{j=i}^{k} B_j$. If the third statement does not hold, then we choose $S = \bigcup_{j=1}^{i} A_j$. If the fourth statement does not hold, we put all vertices of the smaller partition set into the set S. In all cases, the graph $G - S$ contains more than $|S| + 1$ components.

Similar to the proof of Lemma 6.2, both the construction of the Hamiltonian path and of the set S can be implemented in time $\mathcal{O}(n)$. □

Lemma 6.5 *Let $G = (V, E)$ be a chain graph and let $\mathcal{P} = (A_1, B_1, \ldots, A_k, B_k, I)$ be the corresponding chain partition. The graph G has a Hamiltonian cycle if and only if the following conditions hold:*

1. $I = \emptyset$;
2. $(\sum_{j=i}^{k} |A_j|) + 1 \leq \sum_{j=i}^{k} |B_j| \; \forall i \in \{2, \ldots, k\}$;
3. $\sum_{j=1}^{k} |A_j| = \sum_{j=1}^{k} |B_j|$.

There is an algorithm that finds a Hamiltonian path or a set $S \subset V$ where $G - S$ has more than $\max\{1, |S|\}$ *components in time $\mathcal{O}(n)$.*

Proof The proof follows a similar chain of arguments as the proof of Lemma 6.3. If the three conditions hold, then we start in a vertex of B_k and visit all vertices in A_k first, alternating with vertices from B_k. Iterating this procedure also for the other A-sets and using Conditions 2 and 3 we can visit all vertices in A and B. After visiting the last vertex in A_1, we go back to our starting vertex in B_k. Since Condition 3 holds with equality, all vertices are contained in the cycle.

If the first statement does not hold, then we set $S = \emptyset$. If the second statement does not hold for $i \in \{2, \ldots, k\}$, then we set $S = \bigcup_{j=i}^{k} B_j$. If the last statement does not hold, we add all vertices of the smaller partition set to the set S. In all three cases, the graph $G - S$ contains more than $\max\{1, |S|\}$ components.

Similar to the proof of Lemma 6.2, both the construction of the Hamiltonian cycle and of the set S can be implemented in time $\mathcal{O}(n)$. □

Using the above characterizations and the results of Sect. 3, we can formulate a fully dynamic algorithm on threshold graphs and chain graphs for both Hamiltonian paths and cycles.

Theorem 6.6 *There are fully dynamic algorithms for the Hamiltonian cycle problem and the Hamiltonian path problem on threshold graphs and chain graphs which have the following costs:*

- *Initialization: $\mathcal{O}(n + m)$*
- *Adding and deleting an edge: $\mathcal{O}(1)$*
- *Adding and deleting a set of k edges: $\mathcal{O}(\min\{k \log k, n + k\})$*
- *Adding and deleting a vertex with d neighbors: $\mathcal{O}(d)$*
- *Computing a Hamiltonian cycle or path, or a vertex set contradicting Observation 6.1 contained in G: $\mathcal{O}(n)$*

Proof We will prove this theorem for threshold graphs; the algorithms for chain graphs work analogously and their correctness and running time can be shown in the same way.

We use the threshold data structure defined in Definition 3.4. Let $\mathcal{P} = (I_1, U_1, \ldots, U_k, I_{k+1})$ be the threshold partition of the threshold graph G. For every set U_i we introduce a variable α_i storing the number $\sum_{j=1}^{i} |I_j|$ and for every set I_i we have a variable β_i storing the number $\sum_{j=i}^{k} |U_j|$. This ensures that every set only counts vertices that are adjacent to the elements of the set. Furthermore, we store the total size γ of the U-sets and the total size δ of the I-sets. In order to check whether G has a Hamiltonian cycle or a Hamiltonian path, we have to compare γ and δ. Furthermore, for every set I_i and U_i we have to compare β_i and $\gamma - \alpha_i + |I_i|$ as stated in Lemmas 6.2 and 6.3. Since we also want to handle threshold graphs without a Hamiltonian cycle or a Hamiltonian path, we introduce a variable ζ which counts the

number of pairs U_i and I_i whose values α_i and β_i do not fulfill the given constraint. Whenever we update α- or β-values we check whether the status of the corresponding constraint has changed. If this is the case we increase or decrease ζ. Therefore, the graph has a Hamiltonian cycle or Hamiltonian path if and only if $\zeta = 0$, I_{k+1} is empty and the constraint on γ and δ is fulfilled.

If we delete an edge from G or add an edge to G, we apply the operations described in the proofs of the Lemmas 3.5 and 3.6. As one can see there is only a constant number of variables α_i and β_i which have to be updated. Furthermore, we have to update γ and δ. For the addition and deletion of whole sets of edges we use the algorithms described in Theorems 5.6 and 5.11.

If we delete a vertex z of a U-set, we only have to update the variables of I-sets that are to the left of it in P. All elements of these sets are neighbors of z and, therefore, it only uses $\mathcal{O}(|N(z)|)$ operations. If we delete an element z of an I-set, we only have to update the variables of U-sets to the right of it and their elements are neighbors of z. Thus, the number of operations is also in $\mathcal{O}(|N(z)|)$.

For the addition of a vertex we consider the operations in the proof of Lemma 3.7. Here, we only have to update sets whose elements are neighbors of the new vertex z. Therefore, these operation can also be done in time $\mathcal{O}(d)$, where d is the number of neighbors of z. □

7 Conclusion

In this paper we have presented optimal fully dynamic recognition algorithms for the classes of threshold and chain graphs, which additionally yield a constant size certificate in the negative case. Furthermore, we expand on the idea of a fully dynamic algorithm by giving efficient routines for the addition and deletion of sets of vertices or edges. Using the techniques developed for these algorithms, it was also possible to state fully dynamic algorithms for the Hamiltonian path and cycle problems for both of these classes. These algorithms have the same running times as the recognition algorithms and return certificates in the negative case which are of size $\mathcal{O}(n)$ and can also be checked in time $\mathcal{O}(n)$.

Some open questions remain. While our algorithms are able to decide whether a modified graph has a Hamiltonian path or cycle, it is not yet possible to give a fully dynamic update on the path or cycle itself. Therefore, it is an interesting question whether there exists a data structure to keep track of the cycle or the path efficiently on these graph classes while adding or removing vertices and edges.

To the best of our knowledge the problem of dynamically solving the Hamiltonian cycle and path problems has not been considered so far. Thus, a natural question is whether one can find efficient dynamic algorithms for these problems on other graph classes. Clearly, only graph classes for which these problems can be solved efficiently in a non-dynamic way are candidates. Examples for such graph classes are cographs and interval graphs.

Similarly, the addition or deletion of whole sets of edges does not seem to have been considered so far. It is an interesting question whether similar approaches can be

implemented for other graph classes, especially graph classes that are not sandwich monotone.

In the light of these results, one can also ask whether further graph problems can be solved efficiently in a dynamic way on threshold or chain graphs. A possible candidate could be a dynamic algorithm for the max cut problem on threshold graphs.

Acknowledgements We thank Nina Chiarelli, Matjaž Krnc, Martin Milanič, and Nevena Pivač for fruitful discussions. Furthermore, we thank the German Academic Exchange Service for its financial support (BI-DE/17-19-18 and BI-DE/19-20-007).

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Beisegel, J., Chiarelli, N., Köhler, E., Krnc, M., Milanič, M., Pivač, N., Scheffler, R., Strehler, M.: Edge elimination and weighted graph classes. In: Adler, I., Müller, H. (eds.) Graph-Theoretic Concepts in Computer Science, LNCS, vol. 12301, pp. 134–147. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60440-0_11
2. Bhattacharya, S., Henzinger, M., Nanongkai, D.: New deterministic approximation algorithms for fully dynamic matching. In: Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, pp. 398–411 (2016). https://doi.org/10.1145/2897518.2897568
3. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM (1999). https://doi.org/10.1137/1.9780898719796
4. Calamoneri, T., Monti, A., Petreschi, R.: Fully dynamically maintaining minimal integral separator for threshold and difference graphs. In: WALCOM: Algorithms and Computation, LNCS, vol. 9627, pp. 313–324. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30139-6_25
5. Chvátal, V., Hammer, P.L.: Set-packing and threshold graphs. Tech. Rep. CORR 73-21, University of Waterloo (1973)
6. Chvátal, V., Hammer, P.L.: Aggregations of inequalities in integer programming. Stud. Integ. Program. Ann. Discrete Math. **1**, 145–162 (1977)
7. Cozzens, M.B., Leibowitz, R.: Threshold dimension of graphs. SIAM J. Algebr. Discrete Methods **5**, 579–595 (1984). https://doi.org/10.1137/0605055
8. Cozzens, M.B., Leibowitz, R.: Multidimensional scaling and threshold graphs. J. Math. Psychol. **31**, 179–191 (1987). https://doi.org/10.1016/0022-2496(87)90014-9
9. Crespelle, C., Paul, C.: Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. Algorithmica **58**(2), 405–432 (2010). https://doi.org/10.1007/s00453-008-9273-0
10. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic shortest paths in digraphs with arbitrary arc weights. J. Algorithms **49**(1), 86–113 (2003). https://doi.org/10.1016/S0196-6774(03)00082-8

11. Hammer, P.L., Peled, U.N., Sun, X.: Difference graphs. Discrete Appl. Math. **28**(1), 35–44 (1990). https://doi.org/10.1016/0166-218X(90)90092-Q
12. Harary, F., Peled, U.: Hamiltonian threshold graphs. Discrete Appl. Math. **16**, 11–15 (1987). https://doi.org/10.1016/0166-218X(87)90050-3
13. Heggernes, P., Kratsch, D.: Linear-time certifying recognition algorithms and forbidden induced subgraphs. Nordic J. Comput. **14**(1–2), 87–108 (2007)
14. Heggernes, P., Papadopoulos, C.: Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. In: International Computing and Combinatorics Conference, pp. 406–416. Springer (2007). https://doi.org/10.1007/978-3-540-73545-8_40
15. Heggernes, P., Papadopoulos, C.: Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. Theor. Comput. Sci. **410**(1), 1–15 (2009). https://doi.org/10.1016/j.tcs.2008.07.020
16. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM J. Comput. **31**(1), 289–305 (2002). https://doi.org/10.1137/S0097539700372216
17. Henderson, P.B., Zalcstein, Y.: A graph-theoretic characterization of the PV_{chunk} class of synchronizing primitives. SIAM J. Comput. **6**(1), 88–108 (1977). https://doi.org/10.1137/0206008
18. Henzinger, M.R.: Fully dynamic biconnectivity in graphs. Algorithmica **13**(6), 503–538 (1995). https://doi.org/10.1007/BF01189067
19. Henzinger, M.R., Fredman, M.L.: Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998). https://doi.org/10.1007/PL00009228
20. Ibarra, L.: Fully dynamic algorithms for chordal graphs and split graphs. ACM Trans. Algorithms **4**(4), 1–40 (2008). https://doi.org/10.1145/1383369.1383371
21. Klein, P.N., Subramanian, S.: A fully dynamic approximation scheme for shortest paths in planar graphs. Algorithmica **22**(3), 235–249 (1998). https://doi.org/10.1007/PL00009223
22. Koop, G.J.: Cyclic scheduling of offweekends. Oper. Res. Lett. **4**, 259–263 (1986). https://doi.org/10.1016/0167-6377(86)90026-X
23. Mahadev, N.V.R., Peled, U.N.: Threshold Graphs and Related Topics. Annals of Discrete Mathematics, vol. 56. North-Holland Publishing Co., Amsterdam (1995)
24. Ordman, E.T.: Threshold coverings and resource allocation. In: Proceedings of the 16th Southeastern Conference on Combinatorics, Graph Theory, and Computing, pp. 99–113. Utilitas Mathematica Pub., Winnipeg (1985)
25. Ordman, E.T.: Minimal threshold separators and memory requirements for synchronization. SIAM J. Comput. **18**(1), 152–165 (1989). https://doi.org/10.1137/0218010
26. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**(2), 266–283 (1976). https://doi.org/10.1137/0205021
27. Shamir, R., Sharan, R.: A fully dynamic algorithm for modular decomposition and recognition of cographs. Discrete Appl. Math. **136**(2–3), 329–340 (2004). https://doi.org/10.1016/S0166-218X(03)00448-7
28. Soulignac, F.J.: Fully dynamic recognition of proper circular-arc graphs. Algorithmica **71**(4), 904–968 (2015). https://doi.org/10.1007/s00453-013-9835-7
29. Thorup, M.: Near-optimal fully-dynamic graph connectivity. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, pp. 343–350 (2000). https://doi.org/10.1145/335305.335345
30. Yannakakis, M.: The complexity of the partial order dimension problem. SIAM J. Algebr. Discrete Methods **3**(3), 351–358 (1982). https://doi.org/10.1137/0603036