Linear-time certifying recognition algorithms and forbidden induced subgraphs*

Pinar Heggernes† Dieter Kratsch‡

Abstract

We give the first linear-time certifying algorithms to recognize split graphs, threshold graphs, chain graphs, co-chain graphs and trivially perfect graphs, with sublinear certificates for negative output. In case of membership in a given graph class our algorithms provide as certificate a structure for the input graph that characterizes the class, and in case of non-membership they provide as certificate a forbidden induced subgraph of the class. The certificates of membership can be authenticated in time $O(n + m)$ and the certificates of non-membership can be authenticated in time $O(1)$.

Keywords: Algorithms and data structures, graph algorithms, recognition of graph classes, certifying algorithms.

1 Introduction

Certifying algorithms. The study of certifying algorithms is motivated by software engineering, software reliability and the insight that software is often not bug-free. Although an algorithm has been proven correct, its implementation may contain bugs. Thus it is desirable to have tools for knowing whether the output of an implementation of an algorithm is correct or returned due to a bug. Clearly there is no way to guarantee by the design and analysis of an algorithm that its implementations are bug-free. Nevertheless algorithm design may support software reliability.

We consider the following approach. A *certifying algorithm* for a decision problem is an algorithm that provides a *certificate* with each answer. A certificate is an evidence that can be used to authenticate the correctness of the answer. An *authentication algorithm* is a separate algorithm that checks the validity of the certificate; it takes the input, the output, and the certificate returned by the original algorithm, and

*The research was done while D. Kratsch was visiting the University of Bergen, supported by The Research Council of Norway.

†Department of Informatics, University of Bergen, PO box 7803, N-5020 Bergen, Norway. Email: pinar.heggernes@ii.uib.no

‡Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France. Email: kratsch@univ-metz.fr

verifies (independently of the original algorithm) whether the output is correct. Bug-free implementation of the authentication algorithm is crucial, and thus authentication should be simple. Furthermore, authentication should not require computation of the certificate from scratch. (In exceptional cases algorithms might be so simple and easy to implement that certificates and authentication are no issue.) Note that certifying algorithms allow to find out whether their outputs are correct, no matter whether their implementation has bugs or not; however the implementation of the authentication algorithm must be bug-free.

Certifying algorithms are highly desirable in practice to reduce the risk of erroneous answers caused by bugs in the implementation [18, 19]. For general discussions on result checking see also [24] and [19, section 2.14].

A certificate is *sublinear* if its authentication algorithm has a better running time than a linear one; and a certificate is *weak* if it takes the same time to authenticate as it does to solve the original problem without the certificate [18]. A familiar example is a linear-time certifying algorithm to recognize bipartite graphs, computing a 2-coloring for bipartite input graphs and an odd cycle for non-bipartite input graphs. The authentication algorithm for the membership certificate checks in time $O(n+m)$ the validity of the 2-coloring by cycling through all edges, and thus the certificate is weak. The authentication algorithm for the non-membership certificate checks in time $O(n)$ that it is a cycle, it has odd length, and that the claimed edges occur in the input graph, and thus the certificate is sublinear.

Forbidden induced subgraphs. For certifying graph recognition algorithms, certificates of non-membership that are forbidden subgraphs of the recognized class are especially desirable. Characterizations of graph classes by forbidden subgraphs are highly regarded in graph theory, and the corresponding certificates are often sublinear and indeed easy to authenticate. Small forbidden induced subgraphs as certificate of non-membership can easily be highlighted in a graphic presentation of the input graph and in this way they are a particularly convincing certificate of non-membership for the user. (No matter whether the implementation of the algorithm has bugs or not, this input graph is definitely not a member of the graph class to recognize.)

Furthermore, a forbidden subgraph for a non-member of the class is of value in its own and might be of use in other algorithms. For example, parameterized algorithms for completing an arbitrary graph into a chordal graph [2] or an interval graph [13] by adding the minimum number of edges need to identify forbidden subgraphs and get rid of these recursively as long as the graph at hand is not in the class. For the recognition of probe split graphs [3] a forbidden subgraph, in fact a $2K_2$ is needed when the graph at hand is chordal and non-split.

Previous work. A linear-time planarity test is part of the LEDA system [19, section 8.7]. It computes a planar embedding for planar input graphs and a subdivision of K_5 or $K_{3,3}$ for non-planar input graphs. Other graph classes having linear-time certifying recognition algorithms are chordal graphs [23], cographs [7], interval and permutation graphs [18], proper interval graphs [14, 21], proper interval bigraphs [14], proper circular-arc graphs, and unit circular-arc graphs [16].

The graph classes studied in this work, split, threshold, chain, co-chain, and trivially perfect graphs, are well studied graph classes with a wide range of theoretical use [11, 20]. Although linear-time recognition algorithms for split graphs and threshold graphs have been known [11], none of them provides a certificate for non-membership that can be authenticated in $O(n)$ time. Both split graphs and threshold graphs have characterizations through their degree sequences, such that membership and non-membership can be verified in $O(n)$ time when the degree sequence is given [11]. However, if the degree sequence is output as a certificate by a recognition algorithm, an authentication algorithm needs to check that the certificate is a correct degree sequence of the graph, which takes $O(n + m)$ time for both membership and non-membership. Furthermore, since authenticating the certificate involves actually computing all degrees in the graph, it is equivalent to recomputing the certificate from scratch, which is not desired. (Of course, one might feel that computation of the degree sequence is a very simple algorithm, and thus certificates are not needed.) Linear-time recognition algorithms for chain and co-chain graphs and for trivially perfect graphs are not difficult to establish. Nevertheless, prior to our work, no linear-time certifying algorithm was known for these classes of graphs.

Our Results. We present linear-time certifying algorithms to recognize split graphs, threshold graphs, chain graphs, co-chain graphs, and trivially perfect graphs. For all our algorithms, the certificate of membership can be authenticated in time $O(n + m)$. All our non-membership certificates are based on a characterization of the graph class by forbidden induced subgraphs [1, 11], meaning that each certificate of non-membership is an induced subgraph of the input graph of constant (small) size. Consequently, all our certificates of non-membership can be authenticated in time $O(n)$, and thus they are all sublinear. Even more, using ordered adjacency list representation of the input graph and a tricky way of passing information on the found constant-size induced subgraph from the recognition to the authentication algorithm, we show that authentication of a constant-size induced subgraph can in fact be done in time $O(1)$.

Each graph class that we study occupies a separate section of this paper. The linear-time certifying algorithm for split graphs either provides a partition of the vertex set into a clique and an independent set as certificate of membership, or it provides as a certificate of non-membership a subset of vertices inducing a $2K_2$, C_4, or C_5 in the input graph. The linear-time certifying algorithm for threshold graphs either provides a partition of the vertex set into a clique and an independent set and a so-called nested neighborhood ordering as certificate of membership, or it provides as a certificate of non-membership a vertex subset inducing a $2K_2$, C_4, or P_4 in the input graph. The linear-time certifying algorithm for chain graphs either provides a bipartition and a nested neighborhood ordering as certificate of membership, or it provides as a certificate of non-membership a vertex subset inducing a C_3, $2K_2$, or C_5 in the input graph. The linear-time certifying algorithm for co-chain graphs either provides a partition of the vertex set into two cliques and a nested neighborhood ordering as certificate of membership, or it provides as a certificate of non-membership a vertex subset inducing a $\overline{C_3}$, C_4 or C_5 in the input graph. Finally, we give two linear-time certifying algorithms for trivially

perfect graphs, one that provides a cotree and one that provides a so-called universal-in-a-component ordering as certificate of membership, and both of them provide as a certificate of non-membership a vertex subset inducing a P_4 or C_4 in the input graph.

2 Preliminaries

All graphs in this paper are simple and undirected. For a graph $G = (V, E)$, we denote its vertex set by $V(G) = V$ and edge set by $E(G) = E$, and we let $n = |V|$ and $m = |E|$. An edge between vertices u and v is denoted by uv. If u and v are not adjacent we call uv a *non-edge*. The set of *neighbors* of a vertex $v \in V$ is the set of all vertices adjacent to v, denoted by $N_G(v)$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. The *degree* of a vertex v is $d_G(v) = |N_G(v)|$. (We omit the subscripts from these notations when there is no ambiguity.) A *clique* is a set of vertices that are all pairwise adjacent, and an *independent set* is a set of vertices that are all pairwise non-adjacent. A vertex v is called *simplicial* if $N(v)$ is a clique, *universal* if $N[v] = V$, and *isolated* if $N(v) = \emptyset$. The subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$. All subgraphs in this text are induced subgraphs. The *complement* $\overline{G}$ of G is a graph that has the vertices of G as its vertex set and the non-edges of G as its edge set.

A cycle on k vertices is denoted by C_k, and a path on k vertices is denoted by P_k. The following graph is called $2K_2$: $(\{a, b, c, d\}, \{ab, cd\})$.

Let α be an ordering $(v_1, v_2, ..., v_n)$ of V. If α is such that $d(v_1) \leq d(v_2) \leq ... \leq d(v_n)$, it is called a *non-decreasing degree ordering*. If α is such that $d(v_1) \geq d(v_2) \geq ... \geq (v_n)$, it is called a *non-increasing degree ordering*. If α has the property that v_i is simplicial in $G[\{v_i, v_{i+1}, ..., v_n\}]$, for $1 \leq i \leq n$, it is called a *perfect elimination ordering (peo)*. If α has the property that v_i is universal in a connected component of $G[\{v_i, v_{i+1}, ..., v_n\}]$, for $1 \leq i \leq n$, it is called a *universal-in-a-component ordering (uco)*. An ordering $\beta = (x_1, x_2, ..., x_k)$ of a subset $X \subseteq V$ of vertices is called a *nested neighborhood ordering* if it has the property that $(N(x_1) \setminus X) \subseteq (N(x_2) \setminus X) \subseteq ... \subseteq (N(x_k) \setminus X)$.

Observation 2.1 *Given a graph $G = (V, E)$ and a vertex subset $X \subseteq V$ which is a clique or an independent set, X has a nested neighborhood ordering if and only if any non-decreasing degree ordering of the vertices in X is a nested neighborhood ordering.*

Proof. Let X have a nested neighborhood ordering. If there are two vertices in X with the same degree, then we can conclude that they have the same neighborhood outside of X. This is because they have the same number of neighbors in X since X is an independent set or a clique, and no pair of neighborhoods outside of X are incomparable or disjoint. Thus any non-decreasing degree ordering of X is a nested neighborhood ordering. For the opposite direction, assume that X does not have a nested neighborhood ordering, then no ordering (either non-decreasing degree or not) can be a nested neighborhood ordering. ∎

The class of *chordal graphs* is the class of graphs containing no induced cycle of length longer than 3. A graph is chordal if and only if its vertex set can be ordered by a peo [9].

3 Authentication of induced subgraphs

A graph recognition algorithm is called *certifying* if it can produce a certificate such that the membership or the non-membership of the graph in the class can be checked using this certificate. The following observation is used to establish authentication algorithms for certificates of non-membership.

Observation 3.1 *Let a vertex subset $A \subset V$ of constant size be a certificate of non-membership of a certifying recognition algorithm for a graph class on input graph $G = (V, E)$, where additionally for each vertex of A a pointer to a vertex of a graph H (indicating an isomorphism from $G[A]$ to H) is part of the certificate. Then there is an $O(n)$ time algorithm to authenticate whether $G[A]$ is isomorphic to the graph H.*

Proof. The set A can be provided by the certifying algorithm in a characteristic vector of size n or as a list of size $|A|$ containing pointers to the vertex list of G. Every vertex in A has at most $n-1$ neighbors and non-neighbors, so $G[A]$ can be computed in time $O(|A| \cdot n)$, which is $O(n)$ since $|A|$ is not dependent on the size of G. The pointers can be used to authenticate in time $O(|A|^2)$ that $G[A]$ is isomorphic to H. ∎

Now we strengthen Observation 3.1 to establish that for any constant-size graph H it can be authenticated in time $O(1)$ whether H is a (induced) subgraph of the input graph G; assuming that recognition and authentication algorithm use appropriate data structures. Our approach shows the power of a clever message passing from the recognition to the authentication algorithm. (We shall need and prove this statement for induced subgraphs. It also holds and can be shown similarly for subgraphs in general.)

Theorem 3.2 *Let $G = (V, E)$, represented by ordered adjacency lists, be the input graph to a certifying recognition algorithm for a graph class, and let H be a (fixed) graph of constant size that is computed as a certificate of non-membership by the algorithm. Then there is a way to represent and pass the certificate H to an authentication algorithm such that it can be authenticated in $O(1)$ time whether H is an induced subgraph of G.*

Proof. Suppose that the recognition algorithm finds out that H is an induced subgraph of G, i.e. it computes a bijection $\sigma : V(H) \to A$, where $A \subseteq V$, such that H and $G[A]$ are isomorphic.

Clearly already under the assumptions of Observation 3.1, the vertex set A as well as all pairs $(a, \sigma(a))$ for all $a \in V(H)$ can be provided by the certifying algorithm in a characteristic vector or as a list of size $|V(H)|$ containing pointers (indicating σ) to the vertex list of G. Pointers to the image of an edge of H are also easy to provide: for

an edge ij (being image of an edge in H) in G add a pointer to j in the adjacency list $Adj(i)$ and a pointer to i in the adjacency list of $Adj(j)$. Furthermore to support the authentication of this edge, add a pointer from j in $Adj(i)$ to i in the vertex list, and a pointer from i in $Adj(j)$ to j in the vertex list.

However certification and authentication of the non edges of H seems infeasible in ordinary adjacency lists. As pointed out in [18], ordered adjacency lists seem to have important advantages over ordinary adjacency lists. Our approach is based on this. A non edge ij (being image of a non-edge in H) can now be provided as follows: add a pointer to the largest vertex k such that $k < i$ in the list $Adj(j)$, a pointer to the largest vertex k' such that $k' < j$ in the list $Adj(i)$. To support authentication of the non edge, also add a pointer from k to i in the vertex list, and a pointer from k' to j in the vertex list.

Let us mention that any algorithm having found that H is an induced subgraph of G by computing A and σ can provide the above described additional information for the certificate in time $O(n)$ by passing through the adjacency list of the vertices of A in G.

Obtaining as input the graph $G = (V, E)$ and the presentation of the certificate (by a collection of pointers), authentication can be done in time $O(|V(H)|^2) = O(1)$ by verifying for each pair of vertices $\{a, b\}$ in H whether $ab \in E(H)$ if and only if $\sigma(a)\sigma(b) \in E$.

It should be pointed out that, for the the authentication to be reliable, the certifying algorithm should not be able to change the data structure of the input graph other than adding pointers. In particular it is important that the order of the adjacency lists are never changed by the certifying algorithm. This is to ensure that authentication refers to the copy of the data structure that the user has, and thus is completely reliable. To achieve this, the adjacency list data structure can be passed to the certifying algorithm as a read-only parameter. The pointers that are needed to be added within the data structure have nothing to do with the certificate, but only with non-adjacency information, and can be added before passing the structure to the certifying algorithm. Pointers that are needed for the certificate are only added from the images of the vertices in H to the data structure. ∎

4 Split graphs

A graph is a *split graph* if its vertex set can be partitioned into a clique K and an independent set I, in which case we call (K, I) a *split partition*. Hence the split graph property is preserved under the graph complement operation. In fact, the class of split graphs consists of exactly those graphs that are both chordal and co-chordal [8].

Theorem 4.1 [8] *A graph is split if and only if it contains no vertex subset that induces $2K_2$, C_4, or C_5.*

Lemma 4.2 *For a split graph, every non-decreasing degree ordering is a perfect elimination ordering.*

Proof. Let $G = (V, E)$ be a split graph, (K, I) a split partition of G, and α a non-decreasing degree ordering of G. Note that any vertex of I is simplicial in G. Any vertex v with $d(v) > |K|$ belongs to K, and any vertex v with $d(v) < |K|$ belongs to I and is thus simplicial. Hence, if there are no vertices of degree $|K|$ then α is trivially a peo. Assume that G has vertices of degree $|K|$. If all of these are simplicial in G, then α must be a peo. Let x be a non-simplicial vertex of degree $|K|$ with the smallest index in α. Thus all vertices that appear before x in α are simplicial in G. Since x is not simplicial, x belongs to K and has a neighbor w in I which is not adjacent to every vertex of K. Since x is in K and has degree $|K|$, w is the only neighbor of x in I. Furthermore, $d(w) < |K|$ since w is not adjacent to every vertex of K. Thus w is ordered before x in α. After the elimination of w, x becomes simplicial. This argument can be repeated for all non-simplicial vertices of degree $|K|$, and consequently α is a peo. ∎

Based on these properties of split graphs, we give in Figure 1 a certifying algorithm for recognizing split graphs, called Algorithm Certifying-Split.[1]

Lemma 4.3 *Algorithm Certifying-Split is correct.*

Proof. If the algorithm returns YES then the graph is split, since every induced subgraph of each iteration is split, evidenced by the computed sets K and I. Notice in particular that all isolated vertices are placed in I, except when G consists only of isolated vertices, in which case exactly one vertex is placed in K and the rest in I. If the graph is split, then a non-decreasing degree sequence is a peo by Lemma 4.2 and the algorithm places the last k vertices of this ordering in the returned clique K, so the NO cases will never occur, and the algorithm will give a YES reply with a correct split partition. Thus the algorithm outputs the correct YES or NO reply.

We need to argue that in the case of a NO reply, the output vertex subset indeed induces a forbidden subgraph of G. The algorithm outputs a NO and returns in three cases. These are numbered (1), (2), and (3) in the algorithm, and we will consider each case separately.

(1) In this case, there exists a vertex v_i with neighbors v_j and v_k such that $v_jv_k \notin E$, and $i < j < k$. Since the degrees of v_j and v_k are at least as high as that of v_i, both v_j and v_k must have at least one neighbor other than v_i. For any common neighbor z of v_j and v_k, either edge $v_iz \in E$, or G has a chordless 4-cycle induced by $\{v_i, v_j, z, v_k\}$. Assume that every common neighbor of v_j and v_k is also a neighbor of v_i. In this case each of v_j and v_k must have a neighbor other than v_i that is not adjacent to the other or to v_i, because of the above degree argument. Thus there are edges v_jx and v_ky such that $x \neq y \neq v_i$, and neither x nor y is adjacent to v_i. Now, either $xy \in E$ and we have a C_5 induced by $\{v_i, v_j, x, y, v_k\}$, or $xy \notin E$ and we have a $2K_2$ induced by $\{v_j, x, v_k, y\}$.

For the rest of the proof we know that α is a peo, since the algorithm did not terminate at (1). Therefore, we also know that G is chordal, and the largest size k of a clique of G can be computed in time $O(n + m)$ (see e.g. [11]).

[1] In the algorithm, by **return** we mean that the algorithm terminates at that point, and the subsequent lines are not executed.

Algorithm Certifying-Split
Input: A graph $G = (V, E)$.
Output: A reply YES if G is split together with a partition of V into a clique K and an independent set I, or a reply NO if G is not split together with a vertex subset inducing $2K_2$, C_4, or C_5.

Compute a non-decreasing degree ordering $\alpha = (v_1, v_2, ..., v_n)$ of G;
if α is not a peo of G **then**
  Find a vertex v_i with two neighbors v_j and v_k such that $v_jv_k \notin E$ and $i < j < k$;
  if v_j and v_k have a common neighbor z with $v_iz \notin E$ **then**
    $\{v_i, v_j, z, v_k\}$ induces a C_4;
  else
    Find two vertices x and y such that $v_jx, v_ky \in E$ and $v_jy, v_kx, v_ix, v_iy \notin E$;
    if $xy \in E$ **then** $\{v_i, v_j, x, v_k, y\}$ induces a C_5;
    else $\{v_j, x, v_k, y\}$ induces a $2K_2$;
    end-if
  end-if
  Output NO and the vertex set inducing C_4, C_5 or $2K_2$, found above;
  return; **(1)**
end-if
Compute the largest size of a clique in G, and denote it k;
$K = \emptyset$; $I = \emptyset$; $i = n$;
while $|K| \leq k - 1$ **do**
  $A = N(v_i) \cap K$;
  if $|A| = |K|$ **then**
    $K = K \cup \{v_i\}$;
  else
    Find a neighbor $x \notin K$ and a non-neighbor $y \in K$ of v_i;
    Find a neighbor z of y such that $v_iz, xz \notin E$;
    Output NO and $\{v_i, x, y, z\}$ which induces a $2K_2$;
    return; **(2)**
  end-if
  $i = i - 1$;
end-while
while $i \geq 1$ **do**
  $A = N(v_i) \cap (K \cup I)$;
  if $A \subseteq K$ **then**
    $I = I \cup \{v_i\}$;
  else
    Denote by x the single vertex in $A \cap I$;
    Find a vertex $y \in K$ such that $xy, v_iy \notin E$;
    Find a neighbor z of y with $xz \notin E$;
    Output NO and $\{v_i, x, y, z\}$ which induces a $2K_2$;
    return; **(3)**
  end-if
  $i = i - 1$;
end-while
Output YES and K and I;

Figure 1: Algorithm Certifying-Split

(2) In this case, $|K| \leq k-1$. We know that there are k vertices of degree at least $k-1$ in the graph, and since v_i is picked before some of these, $d(v_i) \geq k-1$. Thus, since v_i has at least one non-neighbor $y \in K$, it has at least one neighbor $x \notin K$. Notice that no neighbor x of v_i outside of K can be adjacent to y. This is because every such neighbor has an earlier position in α, and its higher numbered neighbors in $K \cup \{v_i\}$ induce a clique, since α is a peo. Thus $v_iy, xy \notin E$. Since y has degree at least as high as that of v_i, $d(y) \geq k-1$, and since there are at most $k-2$ vertices in K in addition to y, vertex y must have a neighbor z outside of K. The positions of x and z in α are earlier than v_i and y. Therefore xz cannot be an edge in G, otherwise v_iy would be an edge in G by the property of a peo. Thus $\{v_i, x, y, z\}$ induces a $2K_2$.

(3) In this case, v_i has exactly one neighbor x in I, since I is an independent set, and the higher numbered neighbors of v_i induce a clique by the peo property of α. Observe that no non-neighbor of x in K can be a neighbor of v_i, since x has a higher position in α, which is a peo. So the set of neighbors of v_i in K is a subset of neighbors of x in K. In addition, x has at most $k-1$ neighbors in K, because otherwise we get a clique of size $k+1$. Thus there is a vertex y in K such that $xy, v_iy \notin E$. Now we will argue that y has a neighbor z which is not adjacent to x or v_i. Assume first that y has a neighbor z outside of K. Vertex z cannot be adjacent to x or v_i since either z is already safely placed in I, or it has a smaller position in α and this would violate the peo property of α. Assume for the other case that y has no neighbor outside of K. This means that $d(y) = k-1$. Since y has higher position in α than x, we can conclude that $d(x) \leq k-1$. Since v_i is a neighbor of x, x has at most $k-2$ neighbors in K, and thus there are at least two vertices in K that are both non-adjacent to x, and consequently to v_i. Therefore, in this case, y has a neighbor z in K which is not adjacent to x or v_i. In either case, we get the desired $2K_2$ induced by $\{v_i, x, y, z\}$. ∎

Lemma 4.4 *The running time of Algorithm Certifying-Split is $O(n+m)$.*

Proof. Golumbic [11] describes a linear-time procedure that checks whether a given ordering is a peo, and if not, provides the vertices v_i, v_j, v_k that are used in the first part of Algorithm Certifying-Split. Furthermore, in [11] it is explained how the size of a largest clique in a chordal graph can be computed in linear time.

In the rest of the algorithm, for each v_i, the time required is linear in the number of edges between v_i and the already processed vertices, as long as v_i is placed in K or I. Thus the total time required by all steps that place a vertex in K or I is $O(n+m)$. Every time we start looking for a forbidden induced subgraph we are already sure that the output is NO. Thus we search for a forbidden subgraph at most once during an execution of the algorithm. Finding such a forbidden subgraph requires $O(n)$ time, since it only concerns checking the neighborhoods of at most 4 vertices. ∎

Lemma 4.5 *The certificates returned by Algorithm Certifying-Split can be authenticated in $O(n+m)$ time for input graphs that are split, and $O(1)$ time for input graphs that are not split.*

Proof. When the algorithm concludes that the graph is split, it returns K and I. It can be checked in $O(n+m)$ time that K is indeed a clique and I is indeed an independent set, and that there are no other vertices in the graph. When the algorithm concludes that the graph is not split, a vertex subset of 4 or 5 vertices is returned as a certificate. By Theorem 3.2, it can be checked in $O(1)$ time that the vertex subset indeed induces a forbidden subgraph. ∎

By the above lemmas, we have proved the following result.

Theorem 4.6 *There is a linear-time certifying algorithm for recognizing split graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

5 Threshold graphs

A graph $G = (V, E)$ is a *threshold graph* if there exists an assignment of non-negative integers to the vertices of G and an integer threshold t such that for every vertex subset $X \subseteq V$ the following holds: X is an independent set if and only if the sum of the labels of the vertices in X is at most t.

Some important characterizations of threshold graphs that will be used for our certifying algorithm are gathered in the following theorem.

Theorem 5.1 [5, 11, 20] *For a graph $G = (V, E)$, the following are equivalent:*

- *G is a threshold graph.*
- *G contains no vertex subset that induces $2K_2$, P_4, or C_4.*
- *G is split and the vertices of the independent set (in any split partition) have a nested neighborhood ordering.*
- *When we repeatedly delete isolated or universal vertices until no such vertices remain, the resulting graph is empty.*

From the above, it is obvious that a disconnected graph is threshold if and only if deleting all isolated vertices results in a connected threshold graph. Based on these characterizations, we give a certifying algorithm for recognizing threshold graphs in Figure 2. In particular, the last point of Theorem 5.1 [11, Exercise 10-4] constitutes the recognition part of our algorithm, which we call Algorithm Certifying-Threshold.

Lemma 5.2 *Algorithm Certifying-Threshold is correct.*

Proof. If the input graph is not split the algorithm returns a forbidden subgraph. If it is split, then we have a partition of its vertices into K and I.

If there are any universal vertices in a graph then any vertex of highest degree must be universal. Note that the non-decreasing degree order of the vertices does not

Algorithm Certifying-Threshold;
Input: A graph $G = (V, E)$.
Output: A reply YES if G is threshold together with a nested neighborhood ordering β of the vertices in the independent set, or a reply NO if G is not threshold together with a vertex set inducing $2K_2$, C_4, or P_4.

Run Algorithm Certifying-Split to recognize whether G is split;
if G is not split **then**
    Output NO, and the vertex set inducing $2K_2$, C_4, or P_4 (contained in C_5) that is returned by the above call to Algorithm Certifying-Split;
else
    Let K and I be as returned by the above call to Algorithm Certifying-Split;
    Let $\alpha = (v_1, v_2, ..., v_n)$ be a non-decreasing degree ordering of G;
    $\beta = (v_1, v_2, ..., v_{|I|})$;
    Delete all isolated vertices;
    threshold =TRUE; $i = n$;
    while v_i is undeleted **do**
        if v_i is universal **then**
            delete v_i;
            for all neighbors x of v_i **do**
                $N(x) = N(x) \setminus \{v_i\}$;
                if $d(x) = 0$ **then**
                    delete x;
                end-if
            end-for
        else
            threshold = FALSE;
            continue; (abort the while-loop)
        end-if
        $i = i - 1$;
    end-while
    if *threshold* **then**
        Output YES, and β;
    else
        repeat
            Delete all remaining vertices of K that have no neighbors in I;
            Delete all remaining vertices of I that are adjacent to all vertices of K;
        until no such vertices remain;
        Choose a vertex v of I of highest degree;
        Find a non-neighbor y of v in K;
        Find a neighbor z of y in I;
        Find a vertex $w \in K$ that is a neighbor v and a non-neighbor of z;
        Output NO and $\{v, w, y, z\}$ which induces a P_4;
    end-if
end-if

Figure 2: Algorithm Certifying-Threshold

change by deleting universal or isolated vertices. Thus at every step of the while-loop, v_i is universal if and only if the remaining graph contains universal vertices. Note also that, as soon as we reach the first deleted vertex v_i, all vertices $v_1, v_2, \ldots, v_{i-1}$ are already deleted, since they have smaller degree. Hence at the end of the while-loop we can conclude that G is threshold if and only if the value of *threshold* is TRUE, by the last point of Theorem 5.1. The ordering β that is output is a non-decreasing degree ordering of the vertices in I. Since we already know that these vertices have a nested neighborhood ordering, then by Observation 2.1, β is a nested neighborhood ordering.

Since we know that G is split, it does not contain a $2K_2$ or a C_4. Thus, if we have concluded that G is not threshold, we know that the remaining graph must contain a P_4. Note that any remaining vertex of K that has no neighbors outside of K cannot belong to a P_4. Similarly, any remaining vertex of I that is adjacent to all remaining vertices of K cannot belong to a P_4. Therefore we delete all such vertices. The remaining graph cannot be complete, because otherwise the repeat-loop would not have stopped. It cannot be empty, since this would mean that there was an undeleted universal vertex after the while-loop. There cannot be any isolated vertices either, since we never delete any neighbor of a vertex in I in the repeat-loop. Thus, in the remaining graph after the repeat-loop, vertex v of highest degree in I has at least one non-neighbor y in K, and y has at least one neighbor z in I. Since the degree of v is at least the degree of z, v must have a neighbor w in K which is not adjacent to z. Hence we have a set $\{v, w, y, z\}$ which induces a P_4. ∎

Lemma 5.3 *The running time of Algorithm Certifying-Threshold is $O(n + m)$.*

Proof. Trivially, the while-loop requires $O(n + m)$ time. As argued above, the non-decreasing degree order never changes. Thus in the repeat-loop it is enough to check the vertices of I starting from the highest degree and the vertices of K starting with the lowest degree. The last steps require linear time. ∎

Lemma 5.4 *The certificates returned by Algorithm Certifying-Threshold can be authenticated in $O(n + m)$ time for input graphs that are threshold, and $O(1)$ time for input graphs that are not threshold.*

Proof. A certificate of membership for input graph G consists of K, I and β. It can be checked in linear time whether β is a nested neighborhood ordering of I as follows. The first vertex of β can mark its neighbors, and initialize a variable that keeps the number of marked vertices. Starting from the second vertex of β, each vertex can first check if it is adjacent to all marked vertices by counting the number of marked neighbors and comparing to the counter, and then mark its neighbors and update the counter. From β, I and K are implicit, and we can check in linear time that I is an independent set and K is a clique. Thus the membership certificate can be authenticated in $O(n + m)$ time. By Theorem 3.2, it can be checked in $O(1)$ time that the vertex subset indeed induces a forbidden subgraph. ∎

With the above three lemmas, we have thus proved the following result.

Theorem 5.5 *There is a linear-time certifying algorithm for recognizing threshold graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

6 Chain and co-chain graphs

A graph $G = (V, E)$ is a *difference graph* if there exist real numbers a_v for each $v \in V$ and a real number t, such that $|a_v| < t$ for $v \in V$ and $uv \in E$ if and only if $|a_u - a_v| \geq t$. A graph whose vertex set can be partitioned into two independent sets is called a *bipartite graph*. This partition of the vertex set into two independent sets is called *bipartition*, and it is unique if and only if the graph is connected. A graph is bipartite if and only if contains no induced cycle of odd length [17]. It follows from the definition of difference graphs that they are bipartite.

Yannakakis gave difference graphs another name. He called them *chain graphs*, and defined a bipartite graph to be a chain graph if one of the sides of the bipartition has nested neighborhood property. He also showed that one side has this property if and only if both sides have the property [25].

Theorem 6.1 [20] *A graph $G = (V, E)$ is a chain graph if and only if there is a bipartition of V into independent sets X and Y such that adding all possible edges with both endpoints in X yields a threshold graph.*

We can see from Theorem 6.1 that chain graphs are bipartite, and vertices on each side of the bipartition have nested neighborhoods. A nice consequence of Theorem 6.1, which is also easy to show independently, is that one side of the bipartition has a nested neighborhood ordering if and only if the other side also has a nested neighborhood ordering.

Theorem 6.2 [20] *A bipartite graph is a chain graph if and only if every induced subgraph without isolated vertices has on each side of the bipartition a vertex adjacent to all vertices of the other side of the bipartition.*

Theorem 6.3 [20] *A graph is a chain graph if and only if it contains no vertex subset that induces $2K_2$, C_3, or C_5.*

Based on the above results, we can obtain a linear-time certifying algorithm for chain graphs.

Theorem 6.4 *There is a linear-time certifying algorithm for recognizing chain graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

Proof. First we check whether the input graph is bipartite. It is well known that simple modifications of breadth-first search can be used to find an odd cycle in a graph or give a bipartition of it into two independent sets in linear time. Thus, after this test,

if the graph is not bipartite, we get a certificate in the form of an odd cycle. Hence, we output a vertex set inducing C_3, C_5, or $2K_2$ (contained in a larger odd cycle) as a certificate.

If the reply is yes, we get a bipartition. The bipartition is unique if the graph is connected. If the graph is not connected then either at most one connected component contains an edge, or the graph contains a vertex set inducing $2K_2$ and thus it is not chain. We place all isolated vertices on one (the same) side of the bipartition. We proceed to compute a non-decreasing degree ordering of each side of the bipartition. By Theorems 5.1, 6.1 and Observation 2.1 the ordering of each side is a nested neighborhood ordering if and only if the input graph is a chain graph. We check this in linear time as described in the proof of Lemma 5.4. If we conclude that the graph is a chain graph, we output YES and the certificates for membership: the bipartition and the ordering of each side, which can be authenticated in $O(n+m)$ time.

If we conclude that the input graph is bipartite but not chain, then we know that it contains a $2K_2$. To find a vertex set inducing $2K_2$, we repeatedly delete from either side a vertex that is adjacent to all vertices of the other side and the resulting isolated vertices, since no such vertex can be a part of a $2K_2$. After this, we take a vertex v of highest degree. We know that v has a non-neighbor y on the other side. Since y is not isolated, it has a neighbor z on the same side as v. Since the degree of v is at least the degree of z, v has at least one neighbor x which z is not adjacent to. We can return $\{v, x, y, x\}$ which induces a $2K_2$. By Theorem 3.2, the certificates for non-membership can be authenticated in $O(1)$ time.

Furthermore, breadth-first search is linear and all remaining steps require $O(n+m)$ time, thus the algorithm runs in time $O(n+m)$. ∎

A *co-bipartite graph* is a graph whose complement is bipartite. A bipartition into two independent sets in a bipartite graph is a bipartition into two cliques in its complement. A graph is a *co-chain* graph if it is the complement of a chain graph. It is straight forward to verify that co-chain graphs are exactly the graphs that are co-bipartite and have the nested neighborhood property on each side of the bipartition. In the following, we denote an edgeless graph on 3 vertices ($\overline{C_3}$) by I_3.

Theorem 6.5 [1] *A graph is a co-chain graph if and only if it contains no vertex subset that induces I_3, C_4, or C_5.*

Now we want to show that there is a linear-time certifying algorithm for recognizing co-chain graphs.

Theorem 6.6 *There is a linear-time certifying algorithm for recognizing co-chain graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

Proof. First we study the complement $\overline{G}$ of G. It is well known that breadth-first search on $\overline{G}$, and related operations like computing connected components of $\overline{G}$, can

be performed in $O(n+m)$ time using the adjacencies in G, without actually computing $\overline{G}$ explicitly [4, 15]. In case $\overline{G}$ is disconnected with at least two connected components containing at least one edge each, then we can easily find a set of vertices inducing $2K_2$ in $\overline{G}$, which induces C_4 in G, and hence G cannot be co-chain. In this case we return a reply NO and the set of vertices inducing C_4 as a certificate. In the opposite case, we simply check whether $\overline{G}$ is bipartite. If $\overline{G}$ is not bipartite, then we find a set of vertices inducing C_3, C_5, or $2K_2$ (contained in a larger odd cycle) in $\overline{G}$. These sets of vertices induce I_3, C_5, or C_4, respectively, in G. Again, we reply NO, and return the found set of vertices. Until this point, we have only used breadth-first search on $\overline{G}$, and thus spent $O(n+m)$ time.

After the above steps, if $\overline{G}$ is bipartite, then the bipartition is unique (except any isolated vertices), and gives a bipartition of G into two cliques, and we know that G is co-bipartite. The remaining work is done on G. The isolated vertices of $\overline{G}$ are universal in G, and can safely be placed on either side of the bipartition.

Now we remove each edge of G with both endpoints belonging to the same side of the bipartition. Definitely, the resulting graph G' is bipartite. Next, we check that each side of the bipartition has a nested neighborhood ordering, as described in the proof of Lemma 5.4. If we conclude that G' is not bipartite chain, then we get a set of vertices inducing a $2K_2$ in G', and we return this set as a certificate for a NO reply, since this set induces a C_4 in G. If we conclude that G' is bipartite chain, then we return a reply YES and the ordering of the vertices of each side.

The work described in the above paragraph requires $O(n+m)$ time. The certificates can be authenticated in $O(n+m)$ time for membership, and in $O(1)$ time for non-membership. ∎

7 Trivially perfect graphs

A graph G is a *trivially perfect graph* if for each induced subgraph H of G, the number of maximal cliques of H is equal to the maximum size of an independent set of H [10]. The following is a characterization of trivially perfect graphs through their forbidden subgraphs.

Theorem 7.1 [10] *A graph is trivially perfect if and only if it contains no vertex subset that induces P_4 or C_4.*

A *cograph* is a graph without a vertex subset that induces a P_4. Hence trivially perfect graphs form a subclass of cographs. In fact, it follows immediately that trivially perfect graphs are exactly chordal cographs [1]. Since both chordal graphs and cographs have linear time certifying algorithms [23, 7, 12], obtaining a forbidden induced subgraph as a certificate of non-membership can be done by using the previous algorithms. However, the challenge is to give a certificate of membership that can be checked in $O(n+m)$ time. We will here give two linear-time certifying algorithms for trivially perfect graphs that both output as certificates of non-membership a vertex

subset that induces a P_4 or a C_4. The first algorithm outputs as certificate of membership a universal-in-a-component ordering, and the second outputs a restricted cotree (a representation of cographs, defined below).

7.1 Vertex orderings as certificates of membership

Each connected trivially perfect graph G has a universal vertex [26, 27], and consequently each connected induced subgraph of a trivially perfect graph has a universal vertex [11]. Hence for a trivially perfect graph G, we can find an ordering of its vertices $\alpha = (v_1, v_2, ..., v_n)$ such that v_i is universal in the connected component of $G[\{v_i, v_{i+1}, ..., v_n\}]$ that v_i belongs to, for $1 \leq i \leq n$. Thus trivially perfect graphs have universal-in-a-component orderings (uco). We first prove that such orderings characterize trivially perfect graphs.

Theorem 7.2 *A graph is trivially perfect if and only if it has a uco.*

Proof. If a graph is trivially perfect, it has a uco by the results mentioned above. For the opposite direction, let G be a graph that has a uco α and assume for a contradiction that G is not trivially perfect. Thus G has a set of vertices $\{w, x, y, z\}$ that induces a $C4$ or a $P4$. Let x be the vertex among these four that has the earliest occurrence in α. Thus $\{w, x, y, z\}$ is a connected subgraph in the remaining graph when all vertices of α prior to x are removed from G. Then x should be universal in a connected induced subgraph of G that contains $\{w, x, y, z\}$, but this is not possible since there is no vertex in a $C4$ or a $P4$ that is adjacent to all three other vertices. Thus α is not a uco. ∎

Lemma 7.3 *A graph is trivially perfect if and only if every non-increasing degree ordering is a uco.*

Proof. Let $G = (V, E)$ be a graph, and let $\alpha = (v_1, v_2, ..., v_n)$ be a non-increasing degree ordering of G, such that $d(v_1) \geq d(v_2) \geq ... \geq d(v_n)$. If α is a uco, then by Theorem 7.2 G is trivially perfect. In the other direction, assume that G is trivially perfect, and assume for a contradiction that v_i is not universal in the connected component of $G[\{v_i, v_{i+1}, ...v_n\}]$ that it belongs to, for some i. Thus there is a vertex v_j with $j > i$ such that $v_i v_j \notin E$, and there is a path from v_i to v_j containing only vertices from $\{v_i, v_{i+1}, ...v_n\}$. Observe that there must be a vertex v_k on this path that is adjacent to both v_i and v_j because otherwise we have a P_4 with endpoints v_i and v_j. Since v_k is adjacent to v_j, and v_i is not adjacent to v_j, and the degree of v_i is at least the degree of v_k, v_i must have a neighbor x in G that is not adjacent to v_k (x might belong to $\{v_{i+1}, ...v_n\}$ or not). But, if x is adjacent to v_j, then $\{x, v_i, v_k, v_j\}$ induces a C_4, and if x is not adjacent to v_j, this set induces a P_4, contradicting that G is trivially perfect. ∎

Before we can conclude with a linear-time certifying recognition algorithm for trivially perfect graphs, we also have to show the following.

Lemma 7.4 *Given a graph G and an ordering α of the vertices of G, it can be checked in $O(n + m)$ time whether α is a uco of G.*

Proof. We process the vertices of G one by one in the order given by α. At the beginning all vertices are labeled with 0. At each step i, we check whether v_i and all its neighbors have the same label. If so, we change the labels of all neighbors of v_i to i, and delete v_i from the graph. If, at some step, there are several labels among the neighbors of v_i or if their labels are different than the label of v_i, then let k be the largest such label. This means that vertex v_k was not a universal vertex in the connected subgraph that it belonged to in the subgraph $G[\{v_k, v_{k+1}, ..., v_n\}]$, and thus α is not a uco. We claim that if we manage to iterate through all vertices without discovering such an anomaly, then α is a uco. To see this, observe that if α is not a uco then a vertex v_i must exist so that v_i does not label all vertices in the connected component that it belongs to in $G[\{v_i, ..., v_n\}]$ by i. Thus in this subgraph there is a vertex that is labeled by i that has a neighbor that is labeled with something smaller than i. Either the labels of these vertices remain different until the turn comes to one of them and then we detect the anomaly, or some other vertex changes both their labels to become the same. But since that other vertex cannot have the same label as both of them, again the anomaly will be detected. The running time is clearly $O(n+m)$ since we only check the neighborhood of each vertex once. ∎

Now we are ready to give the main result on trivially perfect graphs.

Theorem 7.5 *There is a linear-time certifying algorithm for recognizing trivially perfect graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

Proof. Given the input graph $G = (V, E)$, we start by computing a non-increasing degree order in $O(n+m)$ time. Then we check as described in the proof of Lemma 7.4 whether this order is a uco. If it is, we output YES and this ordering as a certificate of membership. If it is not, then whenever we detect an anomaly, the largest label k among the labels of neighbors of v_i and the label of v_i itself, gives a vertex v_k that was not universal in the connected component of $G[\{v_k, v_{k+1}, ..., v_n\}]$. Thus we can find one vertex in this component that is not adjacent to v_k, and this will give us the desired vertex set that induces a P_4 or a C_4 as described in the proof of Lemma 7.3, in $O(n+m)$ time. Then we output NO and this vertex set as a certificate of non-membership. The certificate of membership can be authenticated in $O(n+m)$ time by Lemma 7.4. The certificate for non-membership can be authenticated in $O(1)$ time by Theorem 3.2. ∎

7.2 Cotrees as certificates of membership

We now give an alternative certifying algorithm for recognizing trivially perfect graphs that exploits structural and algorithmic properties of cographs. If G is a cograph then either G is disconnected, or its complement $\overline{G}$ is disconnected, or G consists of a single vertex. Using the corresponding decomposition rules, one obtains the modular decomposition tree of a cograph which is called a cotree. A *cotree* T of a cograph G is a rooted tree with two types of interior nodes: 0-nodes and 1-nodes. The vertices of G are assigned to the leaves of T in a one-to-one manner. Finally two vertices u and v

are adjacent in G if and only if the lowest common ancestor of the leaves u and v in T is a 1-node [6]. The cotree of a cograph is uniquely determined. Note that due to the above mentioned decomposition rules, we may assume that each interior node of a cotree has at least two children, that no 0-node is a child of a 0-node, and that no 1-node is a child of a 1-node.

Combining Theorem 7.1 with the definition of cotrees, we establish the following result.

Lemma 7.6 *A cograph G is a trivially perfect graph if and only if, in the cotree T of G, every 1-node has at most one child that is a 0-node.*

Proof. Let a be a 1-node of T with two 0-node children, say b and c. Then each of the subtrees rooted at b and rooted at c contains at least two leaves. These four leaves induce a C_4 in G. Thus G is not trivially perfect.

Assume that a cograph G is not trivially perfect. Since it is a cograph, it does not contain P_4 as an induced subgraph. By Theorem 7.1 there is a subset $\{w, x, y, z\}$ that induces a C_4 in G. Let a be the root of the smallest possible rooted subtree of T containing the leaves assigned to w, x, y, z. Since $G[\{w, x, y, z\}]$ is connected, a is a 1-node. If a has two 0-nodes as children, the proof is completed. Thus we may assume that a has exactly one 0-node as a child, say b. By the definition of cotrees, all other children of a are leaves of T. No vertex of $\{w, x, y, z\}$ could be assigned to a child of a, since each of these vertices is non-adjacent to one vertex of $\{w, x, y, z\}$. Thus all four leaves of T corresponding to these vertices are contained in the subtree rooted at b. This contradicts the choice of a. Consequently, a has at least two 0-node children. ■

Thus a certifying algorithm for recognizing trivially perfect graphs can output such a cotree as a certificate of membership. However, then we have to argue that there is an $O(n+m)$ time authentication algorithm for checking this certificate, which also involves checking that the cotree output as certificate is indeed the cotree of G. Typically, certifying cograph recognition algorithms output a cotree if the input is a cograph, and a vertex subset that induces a P_4 if the input is not a cograph [7]. Despite the many cograph recognition algorithms we could not find an $O(n + m)$ time algorithm for checking whether a given cotree is the cotree of a given graph. Therefore we present here a cotree authentication algorithm, which is also the main motivation for giving an additional certifying algorithm for trivially perfect graphs.

Lemma 7.7 *Given a graph G and a tree T, it can be checked in $O(n+m)$ time whether T is the cotree of G.*

Proof. It is easy to check that T is indeed a cotree, i.e. has 0-nodes, 1-nodes and leaves to which the vertices of G are assigned. To check whether T is indeed the cotree of $G = (V, E)$ we verify in a bottom-up fashion the following property which is an immediate consequence of the definition of a cotree.

- If u and v are leaves and have the same 1-node of the cotree T as their parent then $N[u] = N[v]$, and thus $uv \in E$.

- If u and v are leaves and have the same 0-node of the cotree T as their parent then $N(u) = N(v)$, and thus $uv \notin E$.

We now present an authentication algorithm that does exactly this. The adjacency lists are sorted for efficient comparison of the adjacencies.

1. For each vertex v, add v to the adjacency lists it belongs to. Then sort all adjacency lists of G such that vertices appear in the same order as that of the leaves in the cotree T.

2. In a bottom-up fashion check the interior nodes of T thereby updating the graph and the cotree. The current cotree is denoted by T', the current graph is denoted by $G' = G[V']$, where V' is the set of vertices not yet processed. Consequently, $N'[v] = N_{G'}[v] = N[v] \cap V'$ and $N'(v) = N_{G'}(v) = N(v) \cap V'$. Clearly T can only be accepted after a successful check of its root. We start with $G' := G$, $V' := V$ and $T' := T$. Interior nodes have to be checked as follows.

   (R0) Let a be a 0-node of T' all of whose children are leaves, and the corresponding vertices of G' are $u_1, u_2, \ldots, u_t$. Now compare the adjacency lists $N'[u_1], N'[u_2], \ldots, N'[u_t]$. Conclude that T is not the cotree of G and terminate, if any list contains more than one vertex of $\{u_1, u_2, \ldots, u_t\}$, or if the lists $N'(u_1), N'(u_2), \ldots, N'(u_t)$ are not all equal. The comparison can be done in a standard fashion by passing simultaneously through all lists in time proportional to the total length of the inspected lists. If the algorithm has not terminated, conclude that T is the cotree of G if a is the root of T. Otherwise, remove the vertices $u_2, \ldots, u_t$ from G' and V' (updating the adjacency lists of G' accordingly), and the corresponding leaves from T'. Remove the 0-node a from T' and put the leaf u_1 at its place.

   (R1) Let b be a 1-node whose children are all leaves in T', and the corresponding vertices of G' are $v_1, v_2, \ldots, v_s$. Now compare the adjacency lists $N'[v_1], N'[v_2]$, $\ldots, N'[v_s]$ in a standard fashion by passing simultaneously through all lists. If the lists are not all equal, conclude that T is not the cotree of G and terminate. Otherwise, if b is the root of T then conclude that T is the cotree of G and stop. If b is not the root of T, remove all vertices $v_2, \ldots, v_s$ from G' and V', and the corresponding leaves from T', remove the 1-node b from T' and put the leaf v_1 at its place.

Correctness follows immediately from the above mentioned properties.

To establish running time of $O(n + m)$, efficient deletion of vertices need to be supported. One standard approach is to add pointers between the two occurrences of an edge uv, i.e. from u in $N[v]$ to v in $N[u]$ and vice versa, when sorting the lists. Deleting a vertex v takes hence $O(d(v))$ time for removing the whole adjacency list of v and removing v from all adjacency lists of its neighbors. When we compare adjacency lists $N'[v_1], N'[v_2], \ldots, N'[v_s]$, we either find an anomaly and stop, or we delete all of these vertices but v_1. We might have to rescan $N'[v_1]$ and other "survivor" vertices

like v_1. However, notice that at least one vertex is deleted after each adjacency list comparison, and at most one vertex remains undeleted among the compared vertices. Hence we can charge each rescanning of an adjacency list to a unique deleted vertex, and thus each adjacency list is scanned amortized a constant number of times. The total $O(n+m)$ running time follows. ∎

Our second certifying algorithm for recognizing trivially perfect graphs is heavily based on cograph recognition.

Theorem 7.8 *There is a linear-time certifying algorithm for recognizing trivially perfect graphs that outputs certificates which can be authenticated in $O(n+m)$ time for membership and in $O(1)$ time for non-membership.*

Proof. First we present the algorithm.

1. Run a linear time recognition algorithm for cographs that outputs a cotree in case of membership and that outputs a P_4 in case of non-membership.

2. If the cograph recognition algorithm rejects G and outputs a set of vertices inducing a P_4, then conclude that G is not trivially perfect; output NO and the set of vertices inducing a P_4.

3. Otherwise check that in the cotree T each 1-node has at most one 0-node child. If this is the case, then conclude that G is trivially perfect; output YES and T.

4. If none of the above, then T has a 1-node with two 0-node children, say a and b. Each of them has at least two children. Let u and v be vertices assigned to leaves of subtrees rooted at different children of a, let w and x be vertices assigned to leaves of the subtrees rooted at different children of b. Then output NO, and the vertex subset $\{u,v,w,x\}$ which induces a C_4.

An authentication algorithm can check in $O(n+m)$ time that T is the cotree of G by Lemma 7.7. To verify that each 1-node of T has at most one child that is a 0-node can easily be done within the same time limit. The certificate for non-membership can be authenticated in $O(1)$ time by Theorem 3.2. ∎

8 Concluding remarks

Some researchers are mislead to believe that all algorithms for recognizing graph classes are naturally certifying. This is not true in general, and usually different insight is needed to produce useful certificates. The algorithms presented in this paper are mostly completely different from the already known recognition algorithms for the studied classes, and rely on deeper ideas to be able to produce certificates for all outcomes.

We would like to conclude with some open problems. The most interesting problem to continue this line of research is to study circular-arc graphs. Is there a linear-time certifying algorithm for circular-arc graphs? Attacking this problem by forbidden

induced subgraphs as non-membership certificates requires to provide a characterization of this graph class by its forbidden induced subgraphs, which is not yet known. Another interesting graph class to study for certifying recognition algorithms are triangle-free graphs. What could be a good certificate of membership in this case? Finally, we mention graphs of bounded treewidth. Is there a linear-time certifying algorithm for recognizing graphs of treewidth at most k for fixed k?

Acknowledgment

We would like to thank Fedor V. Fomin for inviting Dieter Kratsch to Bergen.

References

[1] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes : A survey*. Philadelphia, SIAM, 1999.

[2] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[3] M. S. Chang, T. Kloks, D. Kratsch, J. Liu, and S. L. Peng. On the Recognition of Probe Graphs of Some Self-Complementary Classes of Perfect Graphs. *Proceedings of COCOON 2005*, Springer LNCS 3595:808–817.

[4] K. W. Chong, S. D. Nikolopoulos, and L. Palios. An optimal parallel co-connectivity algorithm. *Theory of Computing Systems*, 37:527–546, 2004.

[5] V. Chvátal and P. L. Hammer. Set-packing and threshold graphs. Univ. Waterloo Res. Report, CORR 73-21, 1973.

[6] D. G. Corneil, H. Lerchs, and L. Stewart-Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3:163–174, 1981.

[7] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 14:926–934, 1985.

[8] S. Földes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19:311–315, 1977.

[9] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Math.*, 15:835–855, 1965.

[10] M.C. Golumbic. Trivially perfect graphs. *Discrete Math.* 24:105–107, 1978.

[11] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Second edition. Annals of Discrete Mathematics 57. Elsevier, 2004.

[12] M. Habib and C. Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145:183–197, 2005.

[13] P. Heggernes, C. Paul, J. A. Telle, and Y. Villanger. Interval completion with few edges. *Proceedings of STOC 2007*, 374-381, ACM.

[14] P. Hell and J. Huang. Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs. *SIAM J. Discrete Math.*, 18:554–570, 2004.

[15] H. Ito and M. Yokoyama. Linear time algorithms for graph search and connectivity determination on complement graphs. *Inf. Process. Lett.*, 66:209-213, 1998.

[16] H. Kaplan and Y. Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Proceedings of WG 2006*, Springer LNCS 4271:289–300.

[17] D. König. *Theorie der endlichen und unendlichen Graphen*, Akademische Verlagsgesellschaft, Leipzig, 1936.

[18] D. Kratsch, R. M. McConnell, K. Mehlhorn and J. P. Spinrad. Certifying algorithms to recognize interval and permutation graphs. *SIAM J. Computing*, 36:326-353, 2006.

[19] K. Mehlhorn and S. Näher. *LEDA : A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, 1999.

[20] N. V. R. Mahadev and U. N. Peled. *Threshold Graphs and Related Topics.* Annals of Discrete Mathematics 56. North-Holland, 1995.

[21] D. Meister. Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. *Discrete Appl. Math.*, 146:193–218, 2005.

[22] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.

[23] R. E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Computing*, 14:254–255, 1985.

[24] H. Wassermann and M. Blum. Software reliability via run-time result-checking. *Journal of the ACM*, 44:826–849, 1997.

[25] M. Yannakakis. Node deletion problems on bipartite graphs. *SIAM J. Comput.*, 10:310-327, 1981.

[26] E. S. Wolk. The comparability graph of a tree. *Proc. Amer. Math. Soc.*, 13:789–795, 1962.

[27] E. S. Wolk. A note on "The comparability graph of a tree". *Proc. Amer. Math. Soc.*, 16:17–20, 1965.