Degree-preserving forests

Hajo Broersma[1], Andreas Huck[2], Ton Kloks[3], Otto Koppius[1], Dieter Kratsch[4], Haiko Müller[4] and Hilde Tuinstra[1]

[1] University of Twente
Faculty of Mathematical Sciences
P.O. Box 217
7500 AE Enschede, the Netherlands
{broersma,tuinstra}@math.utwente.nl

[2] University of Hannover
Institute of Mathematics
Welfengarten 1
30167 Hannover, Germany
huck@math.uni-hannover.de

[3] Department of Applied Mathematics
Charles University
Malostranské nám. 25
11800 Praha 1, Czech Republic
kloks@kam.ms.mff.cuni.cz

[4] Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
07740 Jena, Germany
{kratsch,hm}@minet.uni-jena.de

Abstract. We consider the degree-preserving spanning tree (DPST) problem: given a connected graph G, find a spanning tree T of G such that as many vertices of T as possible have the same degree in T as in G. This problem is a graph-theoretical translation of a problem arising in the system-theoretical context of identifiability in networks, a concept which has applications in e.g., water distribution networks and electrical networks. We show that the DPST problem is NP-complete, even when restricted to split graphs or bipartite planar graphs, but that it can be solved in polynomial time for graphs with bounded asteroidal number and for graphs with bounded treewidth. For the class of interval graphs we give a linear time algorithm. For the class of cocomparability graphs we give an $O(n^4)$ algorithm. Furthermore we present linear time approximation algorithms for planar graphs of worst case performance ratio $1 - \epsilon$ for every $\epsilon > 0$.

Keywords: graphs, algorithms, complexity, spanning tree, degree condition, planar graphs, cocomparability graphs, interval graphs, treewidth, asteroidal number, AT-free graphs, NP-completeness, approximation.
MSC: 68R10,05C05,05C85,05C90.

1 Description of the problem and its practical niche

Analysis of communication or distribution networks is often concerned with finding spanning trees (or forests) of those networks fulfilling certain criteria. Also in other contexts spanning trees show up as important tools in modeling and analyzing problems. Therefore, a myriad of problems on spanning trees have been studied in literature (see e.g. [10, 11, 16, 18]). This paper deals with a virtually unexplored problem concerning spanning trees which we call the degree-preserving spanning tree (DPST) problem: given a connected graph G, find a spanning tree T of G with a maximum number of degree-preserving vertices, i.e., with a maximum number of vertices having the same degree in T as in G.

Some closely related questions were studied before by Lewinter et al. [1, 7, 25, 26] from a purely theoretical point of view. They published a number of short notes on the subject, but we have not found any paper in which the DPST problem is studied extensively.

Our attention was initially turned to this problem through a practical application in water distribution networks (see [27]), which makes the DPST problem a nice example of theory and practice going hand-in-hand. We shortly describe the application.

Suppose that we have to determine (or control) all flows in a water distribution network by installing and using a small number of flow meters and/or pressure gauges. The network can be regarded as an undirected connected graph G and the flow through each edge of G is described by an orientation of that edge and a nonnegative flow value. Since the sum of all flow values of edges entering a vertex is always the same as the sum of all flow values of edges leaving that vertex, except for possible sources and sinks, it is not difficult to derive all flows in the network from the flows through all edges of a cotree C of G (i.e., C is obtained from G by removing the edges of a spanning tree). Hence it would suffice to install flow meters at the edges of C. However the costs of installing a flow meter are much higher than those of installing a water pressure gauge at some vertex. Alternatively, we can derive the flow through an edge from the water pressure drop between the two incident vertices. If we only use pressure gauges, and want to minimize the costs, the problem becomes that of finding a cotree whose edges are incident with a minimum number of vertices (in order to minimize the number of pressure gauges that have to be installed) or, equivalently, of finding a spanning tree T whose complement in G has as many isolated vertices as possible, i.e., T has a maximum number of degree-preserving vertices. Recently Rahal [28] independently discovered the cotree approach in his investigation of a steady state formulation for water distribution networks.

Our problem of determining all flows in the network with minimal costs of measuring (installing pressure gauges) is a so-called identifiability problem (see Walter [29]). The concrete water distribution network that we considered has 80 vertices and 98 edges, making it a very sparse network. Our network is planar and it has outerplanarity 2. Especially this latter fact enables us to solve the DPST problem in our case by a linear time algorithm, see Section 4.

Although applications of DPSTs so far have been in the area of water distribution networks, this approach generalizes immediately to all networks in which Kirchhoff's laws (which (in an electrical context) state that the sum of currents in a (nonsource, nonsink) vertex is zero and that the sum of voltages over a cycle is zero) are valid and a bijective relation exists between the flow variable and the effort variable. The most obvious example of such a system is an electrical network, with the current I being the flow variable, the voltage V being the effort variable and the bijection being Ohm's law $V = I \cdot R$, but the model can be applied to a wide variety of domains such as mechanics (flow=velocity, effort=force), thermodynamics (f=heat flow, e=temperature), acoustics (f=acoustics volume flow rate, e=acoustic pressure) and in our case the hydraulics of the water distribution network (f=hydraulic volume flow rate, e=hydraulic pressure).

In a slightly more abstract sense, the DPST problem can be thought of as finding a tree of a network in which as many vertices as possible remain 'undamaged'. This idea may have applications in for instance constraint satisfaction problems where a minimal blocking set of constraints needs to be found or a system that needs to be made free from feedback (i.e., cycles) without damaging too many vertices.

Lewinter [25] introduced the concept of degree-preserving spanning trees and he proved that the number of degree-preserving vertices interpolates on the set of spanning trees of a given connected graph G. In other words: if spanning trees exist with k and l degree-preserving vertices respectively and $k < l$, then there exists a spanning tree with exactly m degree-preserving vertices for every m with $k < m < l$. He later generalized the degree-preserving concept to that of deficiency [1, 7]: a vertex is k-deficient if its degrees in the graph and the spanning tree differ by exactly k, and a spanning tree is k-deficient if the maximum deficiency of its vertices is k.

The rest of the paper is organized as follows. We start with some terminology and preliminary results in Section 2. Section 3 deals with the complexity of the DPST problem. It is shown that the problem is NP-complete in general, and that it remains NP-complete even when restricted to split graphs or bipartite planar graphs. In Section 4 it is shown how the DPST problem can be reformulated in Monadic Second Order Logic, thereby proving that the problem is solvable in linear time for graphs with bounded treewidth. Especially this case is very interesting from a practical point of view, since water distribution networks (and other supply networks, such as telephone, data, electricity networks and, more recently, ISDN networks) tend to be of a 'tree-like structure', simply due to the high costs involved in installing and maintaining such networks. (Indeed, these costs are in most situations the bottleneck for the structure of the network to be installed.) This requirement of a tree-like structure directly imposes the study of the problem for graphs with a relatively small treewidth. In Section 5 we apply an idea of Baker [3] to establish linear time approximation algorithms for the DPST problem when restricted to planar graphs.

In the rest of the paper we consider the DPST problem when the input graphs are restricted to some other classes of graphs. In Section 6 we present a linear time algorithm for interval graphs. In Section 7 we obtain an $O(n^4)$ algorithm for cocomparability graphs using the fact that the maximum degree-preserving tree in a 2-edge connected cocomparability graph corresponds to a set of vertices inducing a disjoint union of paths. In Section 8 we show that the DPST problem can be solved by a polynomial time algorithm for graphs with bounded asteroidal number.

2 Preliminaries

In this paper a *graph* is a pair $G = (V, E)$ where V is a finite set (the *vertices* of G) and $E \subseteq V \times V$ is a set of two-element (unordered) subsets of V (the *edges* of G). We write $v \in e$ if a vertex $v \in V$ is *incident* with an edge $e \in E$.

Throughout let $G = (V, E)$ be a graph and let $n = |V|$ and $m = |E|$.

For a nonempty subset $S \subseteq V$ we use $G[S]$ to denote the subgraph of G induced by the vertices of S. For a subset $S \subseteq V$ we also write $G - S$ for $G[V \setminus S]$, and for a vertex x of G we write $G - x$ instead of $G - \{x\}$.

For a vertex x of G we use $N_G(x)$ to denote the set of neighbors of x in G, and we write $N_G[x] = \{x\} \cup N_G(x)$ for the closed neighborhood of x in G; the *degree* of x in G is $d_G(x) = |N_G(x)|$. A *pendant vertex* or *leaf* of G is a vertex with degree one in G. We omit the subscript G from the above expressions if it is clear which graph G we consider. For a graph $G = (V, E)$ and $W \subseteq V$ we define $N[W] = \bigcup_{w \in W} N[w]$ and $N(W) = N[W] \setminus W$.

For a graph G let $\mathsf{Comp}(G) = \{C : G[C] \text{ is a component of } G\}$. A cut vertex is a vertex x of G such that $G - x$ has more components than G. A *block* of G is a maximal subgraph of G without cut vertices, i.e., a K_1, K_2, or a maximal 2-connected subgraph.

Definition 1. A nonempty subset $S \subseteq V$ is *realizable* if there exists a spanning forest T of G such that the degree of every vertex $x \in S$ is preserved in T (i.e., if $d_T(x) = d_G(x)$ for every vertex $x \in S$). If T is such a spanning forest, then we call T an *S-preserving* forest. If, moreover, T is chosen in such a way that $|S|$ is maximum, then we call T a *maximum degree-preserving* forest, and $|S|$ the degree-preserving number (of T or G). The DPST problem is the problem to find for a given graph G a maximum degree-preserving spanning forest.

As an example, the degree-preserving number of a tree, a unicyclic graph, and a complete graph ($\neq K_2$) on n vertices are respectively n, $n-2$, and 1.

Notice that to solve the DPST problem, it is sufficient to compute a maximum (cardinality) realizable set S since, given S, an S-preserving spanning forest is then easy to find. By $p(G)$ we denote the cardinality of a maximum realizable set in G. If a graph G is disconnected, then a maximum realizable set of G is simply the union of maximum realizable sets of all components of G. If a connected graph G (or a component of G) has a bridge e, then to compute a maximum realizable set of G delete e and compute maximum realizable sets S_1 and S_2 for both components. Let T_1 be an S_1-preserving forest and T_2 be an

S_2-preserving forest. Adding e as an edge between T_1 and T_2 gives a forest T which is $S_1 \cup S_2$-preserving, and $S_1 \cup S_2$ is a maximum realizable set in G.

> Henceforth, in the rest of this paper we will assume that all graphs are 2-edge connected.

Let W be a set of vertices of a graph G. By $G[\![W]\!]$ we denote the graph with vertex set $N[W]$ containing all edges of G incident with a vertex in W. Take note of the following simple observations.

Lemma 2. *Let S be a nonempty set of vertices of a graph $G = (V, E)$. Then S is a realizable set of G if and only if $G[\![S]\!]$ is a forest.*

Proof. If S is realizable, then every edge of $G[\![S]\!]$ must be an edge of T for every S-preserving forest T of G. Conversely, if $G[\![S]\!]$ is a forest, then this forest is clearly S-preserving. □

Remark. Clearly, the above lemma implies that for any realizable set S of G, $G[S]$ is a forest.

Corollary 3. *Let S be a realizable set of vertices of G with $x \in S$. Then $G[N[x] \cap S]$ is a star (isomorphic to $K_{1,k}$ for some $k \le d_G(x)$, with center x).*

Lemma 4. *Let S be a realizable set with $x \in S$, and let $y \neq x$ be a vertex with $N(y) \subseteq N[x]$. Then $\{y\} \cup S \setminus \{x\}$ is realizable.*

Proof. Let (V, F) be an S-preserving spanning forest of $G = (V, E)$. Then for $F' = \{\{y, z\} : z \in N_G(y)\} \cup F \setminus \{\{x, z\} : z \in N_G(y)\}$ the graph (V, F') is a $(\{y\} \cup S \setminus \{x\})$-preserving spanning forest of G. □

3 Hardness results

A graph $G = (V, E)$ is called a *split graph* if V can be partitioned into an independent set I and a clique C of G. Such a split graph is also denoted by $G = (I, C, E)$. It is easy to see that split graphs are *chordal graphs*, i.e., graphs that do not contain a chordless cycle of length greater than three.

A graph $G = (V, E)$ is called *bipartite* if V can be partitioned into two independent sets X and Y of G. Such a bipartite graph is also denoted by $G = (X, Y, E)$.

Let $G = (V, E)$ be a graph. We define a split graph H with independent set V and clique $E \times \{1, 2\}$ as follows. A pair $\{v, (e, i)\}$ is an edge of H if and only if $v \in V$, $e \in E$, $i \in \{1, 2\}$ and $v \in e$. It is easy to see that a set $W \subseteq V$ is an independent set of G if and only if W is a realizable set in H. Moreover, if G has no isolated vertices (i.e., vertices with degree zero), then for every realizable set W of H with $|W| > 1$ we have $W \subseteq V$. These simple observations lead to the following theorem showing that the DPST problem restricted to split graphs is NP-complete.

Theorem 5. *For a given split graph H and a given integer k it is NP-complete to decide whether H contains a realizable set of cardinality k.*

Proof. The reduction is from the NP-complete graph problem INDEPENDENT SET. As seen before a graph G has an independent set of cardinality k if and only if the corresponding split graph H has a realizable set of cardinality k. $\square$

Next we apply the same idea to bipartite graphs. Let $G = (V, E)$ be a graph. We define a bipartite graph $B = (V \cup (E \times \{2, 4, 6, 8\}), E \times \{1, 3, 5, 7\}, F_1 \cup F_2)$, where

$$
\begin{aligned}
F_1 &= \{\{v, (e, i)\} : v \in V, e \in E, i \in \{1, 5\}, v \in e\} \\
F_2 &= \{\{(e, 1), (e, 2)\}, \{(e, 2), (e, 3)\}, \{(e, 3), (e, 4)\}, \{(e, 4), (e, 1)\}, \\
&\qquad \{(e, 5), (e, 6)\}, \{(e, 6), (e, 7)\}, \{(e, 7), (e, 8)\}, \{(e, 8), (e, 5)\} : e \in E\}.
\end{aligned}
$$

Note that for the maximum degrees $\Delta(B)$ and $\Delta(G)$ of B and G we get $\Delta(B) = \max\{4, 2 \cdot \Delta(G)\}$. Moreover, B is planar if and only if G is planar.

We observe that for every edge $e \in E$ and every realizable set S of B, $|S \cap (\{e\} \times \{1, 2, 3, 4\})| \leq 2$. In what follows we may assume $S \subseteq V \cup (E \times \{2, 3, 6, 7\})$ for all realizable sets S of B, since for every other realizable set T the set $T' = (T \cap V) \cup (E \times \{2, 3, 6, 7\})$ is also realizable and fulfills $|T| \leq |T'|$.

Next observe that $W \subseteq V$ is an independent set of G if and only if W is a realizable set of B. This leads to the following theorem showing that the DPST problem restricted to bipartite planar graphs is NP-complete.

Theorem 6. *For a given bipartite planar graph B of maximum degree six and a given integer k, it is NP-complete to decide whether B contains a realizable set of cardinality k.*

Proof. The reduction is from the INDEPENDENT SET problem restricted to cubic (i.e., 3-regular) planar graphs [18]. Let (G, k) be an instance of this NP-complete problem where $G = (V, E)$ with $|E| = m$. As seen before a planar graph G has an independent set of cardinality k if and only if the corresponding bipartite planar graph B has a realizable set of cardinality $k + 4m$. $\square$

Our problem remains NP-complete even when restricted to bipartite planar graphs of maximum degree four or three [15].

The INDEPENDENT SET problem is not only NP-complete, it is also hard to approximate. More precisely, for every $\epsilon > 0$, there is no polynomial time approximation algorithm for the MAXIMUM INDEPENDENT SET problem with worst case ratio $n^{1/4-\epsilon}$ unless P=NP [4], and there is no polynomial time approximation algorithm with worst case ratio $n^{1-\epsilon}$ unless co-NP=NP [20]. By the reduction used in the proof of Theorem 5, approximating an optimal solution to the DPST problem is as hard as approximating MAXIMUM INDEPENDENT SET, even when DPST is restricted to split graphs.

Theorem 7. *For every $\epsilon > 0$, there is no polynomial time algorithm to approximate a maximum realizable set of a given split graph with worst case ratio $n^{1/4-\epsilon}$ unless P=NP (respectively with worst case ratio $n^{1-\epsilon}$ unless co-NP=NP).*

However, we cannot conclude the same for the restriction to planar graphs, since the INDEPENDENT SET problem admits a polynomial time approximation scheme for planar graphs [3]. In fact, as we will show in Section 5, the idea of Baker [3] can be applied to establish linear time approximation algorithms for the DPST problem when restricted to planar graphs.

4 Graphs of bounded treewidth

In this section we will prove that the problem of finding a maximum realizable set is solvable in linear time for graphs which have bounded treewidth. From a practical viewpoint this is of great interest, since the graphs under consideration are often of a 'tree-like' structure. Well-known examples of graph classes with bounded treewidth are forests, series-parallel graphs, Halin graphs, almost trees, k-outerplanar graphs, graphs with bounded bandwidth and cutwidth, etc. (see e.g., [22]).

The definition of the treewidth of a graph is usually given in terms of a tree-decomposition (see e.g., [17, 21]). For our purpose it is more convenient to take as a starting point the observation made in [22] that a graph has treewidth at most k if and only if it is a subgraph of a chordal graph with clique number at most $k+1$. If some *constant upper bound* is placed on the treewidth of the graphs under consideration, we say that the class of graphs has *bounded treewidth*. So, for example, the class of Halin graphs is a class of bounded treewidth graphs, since it can be shown that Halin graphs have treewidth at most three (see e.g., [22]).

Many optimization problems can be solved 'efficiently' for graphs of bounded treewidth by formulating the problem in a logical language, called *Monadic Second Order Logic* (abbr. MSOL). It is known that problems which can be expressed in this way can be solved in linear time for graphs with bounded treewidth [2].

The description below of the logical language and Definition 8 below are taken from [17]. For graphs $G = (V, E)$ the previously mentioned MSOL consists of a language in which predicates can be built with the following constituents:

- the logic connectives $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and $\Leftrightarrow$ (with their usual meanings),
- individual variables which may be vertex variables (with domain V), edge variables (with domain E), vertex set variables (with domain $\mathcal{P}(V)$, the power set of V) and edge set variables (with domain $\mathcal{P}(E)$),
- the existential and universal quantifiers ranging over variables ($\exists$ and $\forall$ respectively) and
- the following binary relations:
  - $v \in W$ (where v is a vertex variable and W a vertex set variable),
  - $e \in F$ (where e is an edge variable and F an edge set variable),

- 'v and w are adjacent in G' (where v and w are vertex variables),
- 'v is incident with e in G' (where v is a vertex variable and e an edge variable) and
- equality for variables.

It can easily be seen that if A and B are edge set variables and $\mathcal{H}$ is a predicate in this language, then predicates like $\forall A \subseteq B\ \mathcal{H}$ and $\exists A \subseteq B\ \mathcal{H}$ are also admissible.

Definition 8. We define which graph properties and optimization problems are *MS-definable* as follows.

- An *extended graph property* is a function Q for which there are $D_1, \ldots, D_t$ ($t \geq 0$), such that for each graph G and each $X_i \in D_i$, $1 \leq i \leq t$, $Q(G, X_1, \ldots, X_t)$ is mapped to the value `true` or `false`.
- An extended graph property Q is said to be *MS-definable* if there is a predicate $R(Y_1, \ldots, Y_t)$ that is defined in MSOL for graphs, with free variables $Y_1, \ldots, Y_t$, such that for each graph G and every $X_1, \ldots, X_t$ with $X_i \in D_i$ for each i, $Q(G, X_1, \ldots, X_t)$ has the value `true` if and only if G satisfies $R(X_1, \ldots, X_t)$.
- An optimization problem is MS-definable if there is an MS-definable extended graph property $Q(G, X_1, \ldots, X_t)$ and constants $\alpha_1, \ldots, \alpha_t$ such that the problem is to find for a given graph G the maximum value of $\alpha_1|X_1| + \ldots + \alpha_t|X_t|$ for which $Q(G, X_1, \ldots, X_t)$ evaluates to `true`.

Now we consider the optimization problem P: Given a graph $G = (V, E)$, find a realizable set $S \subseteq V$ of maximum cardinality. We will show that the problem P is MS-definable. For that purpose we define the following predicates, for $v \in V$, $e \in E$, $S \in \mathcal{P}(V)$ and $E', F \in \mathcal{P}(E)$.

1. The boolean expression 'vertex v is incident with edge e' is denoted as: $v \in e$
2. Vertex v is not incident with an edge e from an edge set E':
   $d_0(v, E') = \neg\,(\exists e \in E'\ v \in e)$
3. Vertex v is incident with at least two edges in E':
   $d_{\geq 2}(v, E') = \exists e_1 \in E'\ \exists e_2 \in E'\ \ (\neg(e_1 = e_2)\ \wedge\ v \in e_1\ \wedge\ v \in e_2)$
4. A set F of edges contains a cycle:
   $\mathtt{cycle}(F) = \exists F' \subseteq F\ \ (\exists e' \in F'\ \wedge\ \forall v \in V\ \ (\neg d_0(v, F') \Rightarrow d_{\geq 2}(v, F')))$
5. A set $F \subseteq E$ of edges is the edge set of the graph $G[\![S]\!]$:
   $\mathtt{weakly}(F, S) = \forall e \in E\ \ (e \in F \Leftrightarrow \exists v \in S\ v \in e)$
6. The set of edges of $G[\![S]\!]$ contains no cycle:
   $R(S) = \forall F \subseteq E\ \ (\mathtt{weakly}(F, S) \Rightarrow \neg\mathtt{cycle}(F))$

Lemma 9. *$R(S)$ evaluates to* `true` *if and only if $S \subseteq V$ is a realizable set.*

Proof. This follows from Lemma 2 and the observation that a graph contains a cycle if and only if it has a (nonempty) subgraph in which all vertices have degree at least 2. □

Corollary 10. *Let Q be the extended graph property defined by: $Q(G, S)$ is* `true` *if and only if $S \subseteq V$ is a realizable set of G. Then Q is MS-definable.*

Proof. This follows directly from Definition 8 and Lemma 9. ☐

Corollary 11. *The optimization problem P is MS-definable.*

Proof. This follows from Definition 8 and Corollary 10: take $\alpha_1 = 1$ and the extended graph property Q, then P is equivalent to finding the maximum value of $\alpha_1|S|$ for which $Q(G, S)$ evaluates to `true`. ☐

Arnborg et al. [2] have shown that MS-definable optimization problems can be solved in linear time, given a tree decomposition of bounded width of the input graph. Bodlaender [6] proved that for any fixed constant $k \geq 1$ there exists a linear time algorithm that tests whether a given graph has treewidth at most k and, if so, outputs a tree decomposition of the graph with treewidth at most k. A linear time algorithm for the DPST problem can now be described as follows. First use the algorithm by Bodlaender to find a tree-decomposition of minimum width in linear time. Use this tree-decomposition to find a solution for the DPST problem in linear time using the dynamic programming method described in [2].

Theorem 12. *The DPST problem is solvable in linear time for graphs of bounded treewidth.*

It is important to notice that the graphs arising from applications such as water distribution networks are very sparse and are likely to be k-outerplanar for a very small k, and hence have small treewidth. However for larger values of k (say 4 or 5) the linear time algorithm described above is mainly of theoretical interest. In those cases it is not very practical because of the enormous constants involved.

5 Approximation for planar graphs

In this section we apply an idea of Baker [3] to establish linear time approximation algorithms for the DPST problem when restricted to planar graphs. We will prove the following theorem.

Theorem 13. *For every $\epsilon > 0$ there is a linear time approximation algorithm of worst case performance ratio $1 - \epsilon$ for the DPST problem restricted to planar graphs.*

Let $W \subseteq V$ be a set of (forbidden) vertices. A realizable set R of G is called a *maximum W-avoiding realizable set* if $R \cap W = \emptyset$ and $|R| \geq |R'|$ for every realizable set R' of G with $R' \cap W = \emptyset$.

Let $G = (V, E)$ be a planar graph given with a fixed embedding in the plane. We partition V into levels $L_1, L_2, \ldots, L_d$. The level L_1 contains all vertices on the outer face of G. For $i > 1$, the level L_i contains all vertices on the outer face of $G - \bigcup_{j=1}^{i-1} L_j$. Let d be the largest index such that $L_d \neq \emptyset$. For technical reasons set $L_i = \emptyset$ for $i > d$ or $i < 1$. A planar graph is *k-outerplanar* if and only if it has an embedding defining at most k nonempty levels. We remark that, given a planar graph, a k-outerplanar embedding for which k is minimal can be found in polynomial time [5].

We decompose the planar graph G into k-outerplanar graphs. Each k-outerplanar graph consists of k consecutive levels of G. More precisely, let k and r be integers with $1 \leq r \leq k$. For $i = 0, 1, \ldots, q$ with $q = \lceil \frac{d-r}{k} \rceil$ we define

$$G_{k,r,i} = G\big[\bigcup_{j=(i-1)k+r+1}^{ik+r} L_j\big] \quad \text{and} \quad W_{k,r,i} = L_{(i-1)k+r+1} \cup L_{ik+r} \,.$$

Note that $W_{k,r,i}$ contains all vertices in the outer and inner level of $G_{k,r,i}$.

Lemma 14. *For $i = 0, 1, \ldots, q$ let $R_{k,r,i}$ be a $W_{k,r,i}$-avoiding realizable set of $G_{k,r,i}$. Then $\bigcup_{i=0}^{q} R_{k,r,i}$ is a realizable set of G.*

Proof. For all i the set $W_{k,r,i}$ contains the vertices on the outer and the inner level of the k-outerplanar graph $G_{k,r,i}$. Hence the endvertices of an arbitrary edge of $G[\![R_{k,r}]\!]$ belong to the same k-outerplanar graph. □

Lemma 15. *Let R be a maximum realizable set of G. For every $k \geq 1$ there is an index $r(k)$ with $1 \leq r(k) \leq k$ such that*

$$|R \setminus \bigcup_{i=0}^{q} W_{k,r(k),i}| \geq \tfrac{k-2}{k} p(G) \,.$$

Proof. Let R be a maximum realizable set of G and let $W_{k,r} = \bigcup_{i=0}^{q} W_{k,r,i}$. For every level L_j, $j = 1, 2, \ldots, d$, of G there exist at most two $r \in \{1, 2, \ldots, k\}$ with $L_j \subset W_{k,r}$. Hence $\sum_{r=1}^{k} |R \cap W_{k,r}| \leq 2|R|$, which implies that there is an $r = r(k)$ such that $|R \cap W_{k,r(k)}| \leq \frac{2}{k}|R|$. □

Let $k \geq 1$. For every $r = 1, 2, \ldots, k$ and every $i = 1, 2, \ldots, q$ let $R_{k,r,i}$ be a maximum $W_{k,r,i}$-avoiding realizable set of $G_{k,r,i}$. By Lemma 14, $R_{k,r} = \bigcup_{i=0}^{q} R_{k,r,i}$ is a realizable set of G. Consequently,

$$\max\{|R_{k,r}| : 1 \leq r \leq k\} \geq \tfrac{k-2}{k} p(G) \,.$$

For every k we are able to develop an exact linear time algorithm computing a maximum W-avoiding realizable set for k-outerplanar graphs. Using a variant of the method of Section 4, using labels to indicate the vertices of W, it can be shown that a linear time algorithm exists [2]. Notice that the treewidth of a k-outerplanar graph is at most $3k - 1$ (see [22]). Consequently, for every fixed k we can obtain a linear time approximation algorithm of worst case performance ratio $\frac{k-2}{k}$.

Remark. A detailed analysis of the dynamic programming algorithm for the DPST problem on k-outerplanar graphs could give a polynomial time approximation algorithm with better worst case performance ratio for the DPST problem on planar graphs.

6 Interval graphs

Definition 16. A graph is *chordal* if it contains no induced cycle of length more than three.

There are many characterizations of chordal graphs, for example using perfect elimination schemes, intersection models of subtrees of a tree, the existence of simplicial vertices etc. For an introduction into this graph class we refer to [19].

Notice that for chordal graphs in general, the problem of finding a maximum realizable set is NP-complete, since the class of split graphs is a proper subclass of the class of chordal graphs. However, for the class of interval graphs, which is another important subclass of the class of chordal graphs, we can give a fast algorithm.

Our first result shows that for chordal graphs we can restrict our search for realizable sets to independent sets.

Theorem 17. *If G is a 2-edge connected chordal graph, then any realizable set S of G is an independent set of G.*

Proof. Let $G = (V, E)$ be a 2-edge connected chordal graph and assume $\{x, y\} \in E$ for two distinct vertices $x, y \in S$. Since G is 2-edge connected, $\{x, y\}$ is contained in a cycle of G, and, since G is chordal this implies $\{x, y\}$ is contained in some triangle of G. This contradicts Lemma 2. □

Remark. Notice that the condition that S is independent is in general not sufficient. A counterexample is the diamond (i.e., $K_4 - e$). It has an independent set with two vertices, but clearly this set is *not* realizable.

We will use the above observations and the following properties of 2-edge connected interval graphs.

Definition 18. An interval graph is a graph for which one can associate with each vertex an interval on the real line such that two vertices are adjacent if and only if their corresponding intervals have a nonempty intersection.

Interval graphs can be recognized in linear time, and, given an interval graph, an interval model for it can be found in linear time [8, 19]. In the following we assume that an interval model of the graph is given, and we identify the vertices of the graph with the corresponding intervals. Without loss of generality we may assume that no two intervals have an endpoint in common.

Definition 19. An interval and its corresponding vertex are called *minimal* if the interval is minimal with respect to inclusion, i.e., if it does not contain any other interval.

Lemma 20. *Let G be a 2-edge connected interval graph. Then there exists a maximum realizable set S of G such that for every vertex $p \in S$ the corresponding interval is minimal.*

Proof. Let S be a maximum realizable set containing a vertex x which is not minimal. Then there exists an interval y contained in the interval x. By Theorem 17 we know that a realizable set can contain only one of x and y and hence $y \notin S$. Now $N(y) \subseteq N[x]$, and hence, by Lemma 4 there exists a maximum realizable set $S' = \{y\} \cup S \setminus \{x\}$. Repeating the arguments we can prove the assertion of the lemma. $\square$

Consider the ordering of the minimal intervals defined by the left endpoints.

Lemma 21. *Let G be a 2-edge connected interval graph with corresponding interval model and let x be the first minimal interval (i.e., with the leftmost left endpoint). There exists a maximum realizable set S of G with $x \in S$.*

Proof. Consider a maximum realizable set S of G containing only minimal intervals. If $x \in S$ there is nothing to prove. Otherwise, let y be the first interval in S. The other intervals of S lie totally to the right of y because S is an independent set by Theorem 17. The right endpoint of y must be to the right of the right endpoint of x since the interval x is minimal. It follows that $S' = \{y\} \cup S \setminus \{x\}$ is also realizable, since x lies totally left of $S \setminus \{y\}$ and $N(z) \cap N(x) \subseteq N(z) \cap N(y)$ for all $z \in S \setminus \{y\}$. $\square$

Theorem 22. *There is a linear time algorithm to compute a maximum realizable set S for a given interval graph G.*

Proof. Locate the set of bridges B in G and compute maximum cardinality realizable sets for each component of $G - B$. This can be done as follows.

Consider an interval model for a 2-edge connected component. First mark the minimal intervals. Take the minimal interval with the leftmost left endpoint as the first element of S. Consider the endpoints one by one, from left to right. We keep track of the last minimal interval in S which is totally left of the current position. We also keep a counter for the number of intervals that have one endpoint to the left of the current position and that overlap with the last interval in S. If we encounter a left endpoint of a minimal interval which starts to the right of the last interval in S so far, and if there is at most one interval overlapping the current position and the last interval of S, then we put this new minimal interval in S.

Let S' be a maximum realizable set such that $S \neq S'$. By the previous lemmas we may assume that S' contains minimal intervals only and that S and S' have a common first interval. Suppose y is the first interval of S' which is not in S, and that $x_1, x_2, \dots, x_p$ are common intervals of S and S' and $x_{p+1} \neq y$ is the next interval of S chosen by the above procedure. We complete the proof by showing that y in S' can be replaced by x_{p+1}. This follows by the same arguments as in the proof of Lemma 21 and the following observations. By the choice of $x_1, x_2, \dots, x_p$, for all $i, j \in \{1, \dots, p\}$ with $i \neq j$, x_i and x_j have at most one common neighbor and $N(x_{p+1}) \cap N(x_i) \subseteq N(x_{p+1}) \cap N(x_{i+1})$ $(i = 1, \dots, p-1)$. If the addition of x_{p+1} to $\{x_1, \dots, x_p\}$ would cause a cycle in $G[\![\{x_1, x_2, \dots, x_p, x_{p+1}\}]\!]$, then such a cycle would already exist in $G[\![\{x_1, \dots, x_p\}]\!]$, a contradiction to the choice of $x_1, x_2, \dots, x_p$. $\square$

7 Cocomparability graphs

Definition 23. A graph $G = (V, E)$ is a *cocomparability graph* if and only if there is an ordering $v_1, v_2, \ldots, v_n$ of V such that $i < j < k$ and $\{v_i, v_k\} \in E$ implies either $\{v_i, v_j\} \in E$ or $\{v_j, v_k\} \in E$. Hence $N(v_j) \cap \{v_i, v_k\} \neq \emptyset$ for all j with $i < j < k$. Such an ordering is called a *cocomparability ordering*.

For $w, w' \in V$ we shall write $w < w'$ if w is on the left of w' in the ordering, i.e. $w = v_i$, $w' = v_j$ and $i < j$.

Given a cocomparability graph $G = (V, E)$, a cocomparability ordering can be computed in linear time [12]. In this section we consider a cocomparability graph $G = (V, E)$ with a fixed cocomparability ordering.

Lemma 24. *Let P be a path with endvertices v_i and v_k, $i < k$, in a cocomparability graph G. Then $i < j < k$ implies that v_j has a neighbor in P.*

Proof. Assume $i < j < k$ and v_j does not belong to P. Then P contains an edge $\{v_h, v_l\}$ such that $h < j < l$. Hence v_h or v_l is adjacent to v_j. □

We will use chordless paths of a cocomparability graph in our algorithm to solve the DPST problem.

Lemma 25 [24]. *Let $P = (p_1, p_2, \ldots, p_k)$, $k \geq 1$, be a chordless path in a cocomparability graph G with $p_1 < p_k$. Then $p_i < p_{i+2}$ for all i with $1 \leq i \leq k-2$.*

Consider a chordless path $P = (p_1, p_2, ..., p_k)$ with $p_1 < p_k$ and traverse P from p_1 to p_k. Then each traversed edge is either a forward edge, i.e., the next vertex is further to the right than any previous vertex, or a backward edge, i.e., the next vertex is to the left of the previous vertex but to the right of all other previous vertices. By Lemma 25, there cannot be two consecutive backward edges.

Let S be a subset of vertices such that each component of $G[S]$ is a chordless path. An *S-path* is either the vertex set or the corresponding chordless path of a component of $G[S]$, depending on the context. We say that a vertex w is a common neighbor of two different S-paths S' and S'' if $w \notin S' \cup S''$ and w is adjacent to a vertex $s' \in S'$ and to a vertex $s'' \in S''$.

Our algorithm is based on the following characterization of realizable sets.

Theorem 26. *Let $G = (V, E)$ be a 2-edge connected cocomparability graph. A set $S \subseteq V$ is realizable if and only if*

1. *$G[S]$ is a union of chordless paths,*
2. *two vertices of an S-path have no common neighbor outside S, and*
3. *different S-paths have at most one common neighbor.*

Proof. Assume S is realizable. First consider condition 1. Suppose, on the contrary, that the vertices $c, x, y, z \in S$ induce a claw in G with central vertex c. There is a vertex $x' \in N(x) \setminus \{c\}$ since G is 2-edge connected. Moreover x'

is not adjacent to c, y or z since S is realizable. Similarly there exist vertices $y' \in N(y) \setminus N(\{c, x, z\})$ and $z' \in N(z) \setminus N(\{c, x, y\})$. If $\{x', y', z'\}$ is an independent set of G, then these three vertices form a so-called asteroidal triple (see also Section 8), which is impossible in cocomparability graphs. Hence we may assume $\{x', y'\} \in E$. But now x', x, c, y and y' induce a chordless 5-cycle in G, which is also impossible. Consequently, for 2-edge connected cocomparability graphs condition 1 holds. It is easy to check that a realizable set S must satisfy conditions 2 and 3.

Assume a set S satisfies the three conditions. Consider a shortest cycle C in $G[\![S]\!]$. By conditions 2 and 3, C contains vertices from at least three S-paths. Since C is a shortest cycle each vertex of C that does not belong to S has exactly two neighbors in S which belong to C by condition 2. Hence all three vertices of C in different S-paths form an asteroidal triple. This proves the theorem since cocomparability graphs do not contain asteroidal triples. □

Combining Lemma 24 and Theorem 26 we obtain

Lemma 27. *Let S be a realizable set of a 2-connected cocomparability graph G with $v_i, v_j, v_k \in S$ and $i < j < k$. If v_i and v_k belong to one S-path, then v_j belongs to the same S-path.*

Our dynamic programming algorithm is based on Theorem 26. We still assume that the input graph is 2-edge connected. The algorithm constructs a set S such that $G[S]$ is a union of chordless paths. Thus the algorithm can be considered as a procedure to construct a particular collection of chordless paths of G. For two chordless paths T' and T'' of G, we define $T' < T''$ if $t' < t''$ for all $t' \in T'$ and all $t'' \in T''$. Notice that $S_i < S_j$ or $S_j < S_i$ for any two different S-paths of a realizable set S by Lemma 27.

Lemma 28. *Let T', T'' and $\widetilde{T}$ be chordless paths of a cocomparability graph G such that $T' < \widetilde{T} < T''$ and there is no edge between either T' or T'' and $\widetilde{T}$. Then $t' \in T'$ and $t'' \in T''$ imply $\{t', t''\} \notin E$. Furthermore if T' and T'' have a common neighbor, then every $\tilde{t} \in \widetilde{T}$ is adjacent to every common neighbor of T' and T''.*

Proof. Let $\tilde{t} \in \widetilde{T}$, $t' \in T'$ and $t'' \in T''$. Then by the definition of a cocomparability ordering, $t' < \tilde{t} < t''$ and $\{t', t''\} \in E$ imply either $\{\tilde{t}, t'\} \in E$ or $\{\tilde{t}, t''\} \in E$, contradicting the choice of $\widetilde{T}$.

Now let w be a common neighbor of T' and T''. Then there are $t' \in T'$ and $t'' \in T''$ such that $\{w, t'\} \in E$ and $\{w, t''\} \in E$. Hence (t', w, t'') is a path in G. Since $t' < \tilde{t} < t''$ for all $\tilde{t} \in \widetilde{T}$, Lemma 24 implies $\{w, \tilde{t}\} \in E$. □

Our algorithm constructs a maximum realizable set S of a given cocomparability graph $G = (V, E)$ with cocomparability ordering $v_1, v_2, \ldots, v_n$. It uses a dynamic programming approach with a linear scan through the cocomparability ordering. This technique has been used in previous algorithms for cocomparability graphs (see, e.g., [24]).

A subsolution constructed by the algorithm is a realizable set S. Subsolutions are stored as states $Z \in \{0, 1, \ldots, n\}^4$ with $Z = [z_1, z_2, z_3, z_4]$ such that z_4, z_3 and z_2 are the indices of the last, second last and third last vertex of S, respectively, in the order of the path or, in case of different S-paths, according to the cocomparability ordering. ($z_j = 0$ if the corresponding vertex of S does not exist.) Finally z_1 indicates the maximum number of vertices in a realizable set with last vertices z_2, z_3, and z_4.

The algorithm starts with a preprocessing in which it computes A^2 in time $O(n^{2.376})$ by matrix multiplication [14], where A is the adjacency matrix of G for which $A(i,j) = 1$ if $i \neq j$ and $\{v_i, v_j\} \in E$, and $A(i,j) = 0$ otherwise. Consequently during the dynamic programming part of the algorithm, the number of common neighbors of two vertices v_i and v_j can be computed in constant time.

The dynamic programming algorithm starts with the subsolution $S = \emptyset$ and state $Z = [0, 0, 0, 0]$. It works in rounds $j = 0, 1, \ldots, n-1$, such that in round j the successors of all existing states Z with $j = \max(z_3, z_4)$ are computed. As typical for the dynamic programming approach, the algorithm maintains the following invariant. If Z is a state computed by the algorithm, then there is a realizable set S corresponding to Z, i.e. $|S| = z_1$ and z_2, z_3, z_4 are the last vertices of S. Furthermore for any realizable set S of G the algorithm computes a state Z such that S corresponds to Z with the possible exception of $z_1 > |S|$.

Now Z' is a successor of the state $Z = [z_1, z_2, z_3, z_4]$ if Z' is a state corresponding to a realizable set $S \cup \{v_k\}$, where S is a realizable set corresponding to Z and v_k is added by a backward or a forward step. This means, v_k is a vertex with $z_3 < k < z_4$ in a backward step and $\max(z_3, z_4) < k$ in a forward step.

Any round j of our algorithm consists of two phases. In the first phase all successors via a backward step of previously computed states Z with $j = \max(z_3, z_4)$ are computed. In the second phase all successors via a forward step of previously computed states Z with $j = \max(z_3, z_4)$ are computed. Notice that this implies that in the second phase all successors of states obtained during the first phase are computed. To justify the correctness of our algorithm we show that it is enough to know the last three vertices of any realizable set S for deciding whether $S \cup \{v_k\}$ is realizable or not.

Lemma 29. *Let $P = (p_1, p_2, \ldots, p_k)$, $k \geq 4$, be a chordless path and $p_1 < p_k$. Let u be a vertex with $p_{k-1} < u$. Assume $\widetilde{P} = (p_{k-2}, p_{k-1}, p_k, u)$ is a chordless path. Then $P' = (p_1, p_2, \ldots, p_k, u)$ is also a chordless path. Furthermore if no vertex of $\widetilde{P}$ has a common neighbor with u outside $\widetilde{P}$ then no vertex of P' has a common neighbor with u outside P'*

Proof. Suppose $\widetilde{P} = (p_{k-2}, p_{k-1}, p_k, u)$ is chordless but P' is not. Let l be the largest index $l < k$ for which $\{u, p_l\} \in E$. By our assumption $l < k-2$. Thus $(p_l, p_{l+1}, \ldots, p_k, u, p_l)$ is a chordless cycle of length at least 5 in G. This is a contradiction since a chordless cycle of a cocomparability graph has length at most 4. Suppose w is a common neighbor of u and a vertex in P', while the only neighbor of w in $\widetilde{P}$ is u. Let p_l be the rightmost neighbor of w in P'. Hence

$l < k-2$ and $(p_l, p_{l+1}, \dots, p_k, u, w, p_l)$ is a chordless cycle of length at least 6 in G, a contradiction. $\square$

Lemma 30. *Let $P = (p_1, p_2, \dots, p_k)$ be a chordless path and $p_1 < p_k$. Let $L(P) = \{p_{k-1}, p_k\}$ if $k \geq 2$, and $L(P) = \{p_k\}$ if $k = 1$. Let u be a vertex with $\max(p_{k-1}, p_k) < u$. If u is nonadjacent to the vertices of $L(P)$ then u has no neighbor in P. Furthermore let w be a common neighbor of u and a vertex of P. Then u has a common neighbor with a vertex of $L(P)$, unless either $|L(P)| = 2$ and u is adjacent to both vertices of $L(P)$, or w has at least two neighbors in P.*

Proof. Let $p_i \notin L(P)$ be a vertex of the path P with $\{u, p_i\} \in E$. Then $k \geq 3$, $i \leq k-2$ and $p_i < p_k$. Hence $p_i < p_k < u$ and $\{u, p_i\} \in E$ implies $\{u, p_k\} \in E$, since $\{p_i, p_k\} \notin E$ by the choice of P. Hence u has a neighbor in $L(P)$.

Let w be a common neighbor of u and a vertex $p_i \notin L(P)$ of the path P. Then $k \geq 3$, $i \leq k-2$ and $p_i < p_k$. If $\max(p_{k-1}, p_k) < w$ then w has a neighbor in $L(P)$ as shown above.

Now assume that u is not adjacent to both vertices of $L(P)$ and that w has exactly one neighbor in P, implying that w has no neighbor in $L(P)$. If $w < \min(p_{k-1}, p_k)$ then $\{w, u\} \in E$ implies that both p_{k-1} and p_k are adjacent either to u or to w. By our assumption u is not adjacent to p_{k-1} or p_k. Thus w has two neighbors in P, a contradiction. Finally w cannot be between p_{k-1} and p_k in the cocomparability ordering since $\{p_{k-1}, p_k\} \in E$ implies that w is adjacent to a vertex of $L(P)$, a contradiction. $\square$

Clearly, if u and all vertices of P are contained in a realizable set S of G, then neither u nor w can be adjacent to two vertices of P.

Summarizing we obtain

Proposition 31. *Let S be a realizable set of a cocomparability graph G. Let $L_3(S)$ be the set of the last three vertices of S, if $|S| \geq 3$. Otherwise, let $L_3(S) = S$. Let u be a vertex such that either $s < u$ for all $s \in S$ (forward edge or nonedge) or $s < u$ for all but one $s \in S$ (backward edge). Then $S \cup \{u\}$ is realizable in G if $L_3(S) \cup \{u\}$ is realizable.*

Proof. By Lemma 28, when we add vertex u to a realizable set S, we only have to consider the component of $G[S \cup \{u\}]$ containing u and the previous S-path in the cocomparability ordering. Then by Lemmas 29 and 30, checking $L_3(S) \cup \{u\}$ is sufficient if u creates a new component or is added to an isolated vertex of $G[S]$. By Lemma 30, when adding u to an S-path of more than two vertices the previous component need not be checked. Hence by Lemma 29 checking $L_3(S) \cup \{u\}$ suffices. $\square$

The proposition immediately guarantees the correctness of our dynamic programming algorithm: In a backward step $Z' = [z_1 + 1, z_3, z_4, k]$ is a successor of a state $Z = [z_1, z_2, z_3, z_4]$ if $z_3 < k < z_4$, $\{v_{z_4}, v_k\} \in E$ and $\{v_{z_2}, v_{z_3}, v_{z_4}, v_k\}$

is realizable. In a forward step $Z' = [z_1 + 1, z_3, z_4, k]$ is a successor of a state $Z = [z_1, z_2, z_3, z_4]$ if $\max(z_3, z_4) < k$ and $\{v_{z_2}, v_{z_3}, v_{z_4}, v_k\}$ is realizable.

Consider the running time. The test whether a set of up to 4 vertices is realizable can be done in constant time, since by Theorem 26 it requires only adjacency tests and the computation of the number of common neighbors for vertices in the set. States are maintained as follows. There is a three-dimensional array $B[0..n, 0..n, 0..n]$ initialized to be zero. Whenever a new state $Z' = [z'_1, z'_2, z'_3, z'_4]$ has been computed as a successor, then z'_1 is stored in $B(z'_2, z'_3, z'_4)$, if z'_1 is larger than the current value of $B(z'_2, z'_3, z'_4)$ (which means that we found a better subsolution). Hence during the algorithm the $O(n)$ successors of $O(n^3)$ different states are computed. The algorithm uses a variable **max** to maintain the largest first entry of any state computed. Hence the value of **max** upon termination is the degree-preserving number of the input graph. Consequently the running time of the algorithm is $O(n^4)$.

Using a standard pointer structure the algorithm can be implemented to compute within the same time a maximum realizable set and this can easily be transformed into a maximum degree-preserving forest. Hence we may conclude

Theorem 32. *There is an algorithm to compute a maximum degree-preserving forest of a cocomparability graph in time* $O(n^4)$.

8 Graphs with bounded asteroidal number

We remind the reader that we still assume that G is 2-edge connected.

Definition 33. A set $A \subseteq V$ is called an *asteroidal set* if for every vertex $a \in A$, the set $A \setminus \{a\}$ is contained in a component of $G - N[a]$. The *asteroidal number* of a graph G, denoted by $\mathsf{an}(G)$, is the maximum cardinality of an asteroidal set in G.

Clearly every asteroidal set of G is an independent set of G. On the other hand, every independent set of cardinality at most two is asteroidal in G. An asteroidal set of cardinality three is called *asteroidal triple*, AT for short. The class of AT-free graphs contains all graphs G with $\mathsf{an}(G) \leq 2$. This class, studied in detail in [13], contains all cocomparability graphs. The intersection with the class of chordal graphs gives exactly all interval graphs [23].

In this section we consider graphs with bounded asteroidal number.

For a vertex $w \in W$ a neighbor $u \in N(w)$ is called a *W-private neighbor* of w if $u \notin N[W \setminus \{w\}]$. We define $M(c) = \{v \in V : N[v] \subseteq N[c]\}$. Note that if S is a realizable set containing c then $S \cap M(c) = \emptyset$ since G is 2-edge connected.

The next lemma bounds $|S \cap N(c)|$.

Lemma 34. *Let c be a vertex in a realizable set S of a graph G with* $\mathsf{an}(G) \leq k$. *Then* $|S \cap N(c)| \leq k^2$.

Clearly S contains no elements of $M(c)$. To prove the lemma we need the following claim.
Claim. Let G be a graph on n vertices such that every independent set of G contains at most a vertices and every block of G contains at most c vertices. Then $n \leq a \cdot c$.

Proof. For fixed values a and c we choose a graph $G = (V, E)$ such that it has maximum number of vertices and among those one with maximum number of edges. This implies that every block of G is complete.

Let Z be the set of cut vertices of G and $N = V \setminus Z$.

We consider a block $G[B]$ of G with $B \cap N \neq \emptyset$. By maximality we have $|B| = c$. Every maximal independent set of G contains exactly one vertex of B since $G[B]$ is a complete graph. By the induction hypothesis applied to $G - B$ we know $n - c \leq (a - 1)c$. Consequently, $n \leq ac$.

It remains to show that G has a block containing a vertex in N. Therefore we consider a graph $(\mathcal{B} \cup Z, F)$ where $\mathcal{B} = \{B : G[B] \text{ is a block of } G\}$ and $F = \{\{B, x\} : B \in \mathcal{B} \text{ and } x \in B \cap Z\}$. It is well-known that this graph is a forest. Every leaf of this forest corresponds with a block containing a vertex in N. $\square$

Now we are able to prove the lemma.

Proof. We define $X = S \cap N(c)$ and choose a set $Y \subseteq N(X) \setminus N[c]$ such that every vertex in X has exactly one X-private neighbor in Y. Notice that $|X| = |Y|$ and every vertex of Y has exactly one neighbor in X. Then every independent set of $G[Y]$ is an asteroidal set of G, and for every block $G[T]$ of $G[Y]$ the set $N(T) \cap X$ is an asteroidal set of G. Thus $\mathsf{an}(G) \leq k$ and our claim imply $|X| = |Y| \leq k^2$. $\square$

Definition 35. Let A be an asteroidal set of $G = (V, E)$ and $C \in \mathsf{Comp}(G - N[A]) \cup \{\emptyset\}$. The pair (A, C) is called a *lump* of G if either

- $A = \emptyset$ and $C \in \mathsf{Comp}(G)$ or
- $A = \{a\}$ and $C \in \mathsf{Comp}(G - N[a]) \setminus \mathsf{Comp}(G)$ or
- $|A| \geq 2$ and $C = \bigcap_{a \in A} C_a$ where C_a is the set in $\mathsf{Comp}(G - N[a])$ with $A \setminus \{a\} \subseteq C_a$.

We give some examples. For every asteroidal set A of $G = (V, E)$ with $|A| > 1$ there is exactly one set $C \subseteq V$ such that (A, C) is a lump of G.

If $\mathsf{an}(G) = 1$, then G is complete and $(\emptyset, V)$ is the unique lump of G.

For all $C \in \mathsf{Comp}(G)$ every lump of $G[C]$ is a lump of G. All other lumps of G are of the type $(\{x, y\}, \emptyset)$ for two vertices x and y in different components of G.

If G is isomorphic to $K_{n,m}$, $n \geq 1$ and $m \geq 2$, and (A, C) is a lump of G with $A \neq \emptyset$ or $C \neq V$, then there exist two different nonadjacent vertices x and y of G such that $A = \{x\}$ and $C = \{y\}$ or $A = \{x, y\}$ and $C = \emptyset$.

Lemma 36. *Every graph G on n vertices with $\mathsf{an}(G) = k \neq 2$ has at most n^k lumps. Every AT-free graph G on n vertices has at most $\frac{3}{2}n^2$ lumps.*

Proof. The number of lumps (A, C) with $A = \emptyset$ is bounded by n. The number of lumps (A, C) with $|A| = 1$ is bounded by $n(n-1)$, since for every vertex a at most $n-1$ components of $G - N[a]$ exist. The number of lumps (A, C) with $|A| = i$ is bounded by $\binom{n}{i}$ for $2 \leq i \leq k$, since in this case C is uniquely determined by A. For $k > 2$ this sums up to at most n^k since $n \geq 2k$. $\square$

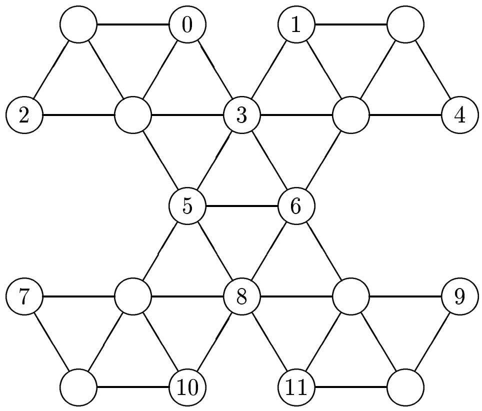

Fig. 1. The set $A = \{2, 4, 7, 9\}$ is an asteroidal set in this graph. Since $|A| \geq 2$ there is a unique set C such that (A, C) is a lump. Here $C = \{0, 1, 3, 5, 6, 8, 10, 11\}$. The vertex $3 \in C$ decomposes (A, C) into three smaller lumps, namely $(\{2, 3\}, \emptyset)$, $(\{3, 4, \}, \emptyset)$, and $(\{3, 7, 9\}, \{8, 10, 11\})$. The vertex $6 \in C$ decomposes (A, C) into four smaller lumps, namely $(\{2, 6\}, \{0\})$, $(\{4, 6\}, \{1\})$, $(\{6, 7\}, \{10\})$, and $(\{6, 9\}, \{11\})$.

The following theorem shows how to decompose lumps into smaller lumps. The basic technique was developed in [9].

Theorem 37. *Let (A, C) be a lump of G. For every $c \in C$ there exist unique partitions $\mathcal{A}$ of A and $\mathcal{C}$ of $C \setminus N[c]$ such that*

1. *for every set $B \in \mathcal{A}$ either $(B \cup \{c\}, \emptyset)$ is a lump of G or there is a set $D \in \mathcal{C}$ such that $(B \cup \{c\}, D)$ is a lump of G, and*
2. *for every set $D \in \mathcal{C}$ either $(\{c\}, D)$ is a lump of G or there is a set $B \in \mathcal{A}$ such that $(B \cup \{c\}, D)$ is a lump of G.*

Proof. Let (A, C) be a lump of $G = (V, E)$ and $c \in C$. We define

$$\mathcal{A} = \{A \cap D : D \in \mathsf{Comp}(G - N[c])\} \setminus \{\emptyset\},$$

$$\mathcal{C}_1 = \{D : D \in \mathsf{Comp}(G - N[c]) \text{ and } A \cap D = \emptyset\},$$
$$\mathcal{C}_2 = \{D : (B \cup \{c\}, D) \text{ is a lump of } G \text{ and } B \in \mathcal{A}\}, \text{ and}$$
$$\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \setminus \{\emptyset\}\,.$$

$\mathcal{A}$ is a partition of A since $\mathsf{Comp}(G - N[c])$ is a partition of $V \setminus N[c]$. By the definition of a lump (Definition 35), $B \cup \{c\}$ is an asteroidal set of G for every $B \in \mathcal{A}$. $\mathcal{C}$ is a partition of $C \setminus N[c]$ since for every set $D \in \mathsf{Comp}(G - N[c])$ either $C \cap D = \emptyset$ or there is exactly one set in $\mathcal{C}$ containing $C \cap D$. Consider the latter case in detail. Either $A \cap D = \emptyset$, $D \in \mathcal{C}_1$ and $(\{c\}, D)$ is a lump, or $A \cap D = B \neq \emptyset$ implying $B \in \mathcal{A}$ and $(B \cup \{c\}, C \cap D)$ is a lump.

Now we consider any pair $(\mathcal{A}', \mathcal{C}')$ of partitions of A and C satisfying the conditions of the theorem. Then $\mathcal{A}'$ is a refinement of $\mathcal{A}$, i.e., for every set $A' \in \mathcal{A}'$ there is a set $B \in \mathcal{A}$ such that $A' \subseteq B$. Otherwise there would be a set $A' \in \mathcal{A}'$ such that $A' \cup \{c\}$ is not asteroidal, contradicting the first assumption. Furthermore, let $A' \in \mathcal{A}'$ and $B \in \mathcal{A}$ be sets such that $b \in B \setminus A'$. The first condition implies $b \in D'$ for the lump $(A' \cup \{c\}, D')$ of G contradicting $b \notin C$. Consequently $\mathcal{A}' = \mathcal{A}$. Now the first condition implies $\mathcal{C}' = \mathcal{C}$ since $\mathcal{C}_1 \subseteq \mathcal{C}'$ by the second condition. □

Let (A, C) be a lump of G. For a fixed vertex $c \in C$, let $\mathcal{A}$ and $\mathcal{C}$ be the partitions given in the proof of the previous theorem. We define the *decomposition* of (A, C) to be the set of lumps $(A' \cup \{c\}, C')$ of G with $A' \in \mathcal{A} \cup \{\emptyset\}$ and $C' \in \mathcal{C} \cup \{\emptyset\}$. This set of lumps is denoted by $\mathsf{Dec}(A, C, c)$.

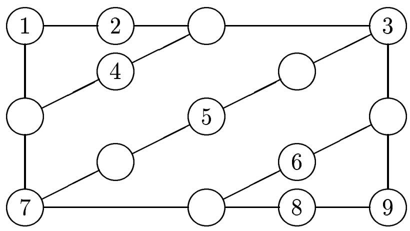

Fig. 2. The set $A = \{3, 5, 7\}$ is an asteroidal set in this graph. Since $|A| \geq 2$ there is a unique set $C = \{1, 2, 4, 6, 8, 9\}$ such that (A, C) is a lump. The vertex $6 \in C$ decomposes (A, C) into two smaller lumps, namely $(A \cup \{6\}, \{1, 2, 4\})$ and $(\{6\}, \{8, 9\})$.

Our algorithm is a dynamic programming algorithm on the lumps of G. Let (A, C) be a lump and let S be a realizable set of G such that $A \subseteq S \subseteq N[A] \cup C$. Let $B = S \cap N(A)$. If $C \cap S = \emptyset$, then $S = A \cup B$. Otherwise let $c \in C \cap S$ and consider the decomposition $\mathsf{Dec}(A, C, c) = \{(A_i, C_i) : 1 \leq i \leq l\}$. Let $U = S \cap N(c)$ and $S_i = S \cap (N[A_i] \cup C_i)$ for $1 \leq i \leq l$. Now it is easy to express

$S_i \cap N(A_i)$ and $S_i \cap C_i$ in terms of A, B and U. However, our task is the other way around: Given the decomposition $\mathsf{Dec}(A, C, c)$ and the sets S_i, $1 \le i \le l$, we have to assemble these sets to a realizable set S of G. The next lemma gives a condition that ensures that realizable sets S_i can be joined to a realizable set S.

Lemma 38. *Let (A, C) be a lump of G and $B \subseteq N(A)$. Let c be a vertex in C and $U \subseteq N(c)$ such that $U \cup \{c\}$ is realizable and $N(A) \cap U = N(c) \cap B$. Let $\mathsf{Dec}(A, C, c) = \{(A_i, C_i) : 1 \le i \le l\}$. For all realizable sets $S_i \subseteq N[A_i] \cup C_i$ of G such that $A_i \subseteq S_i$ and $S_i \cap N(A_i) = (B \cup U) \cap N(A_i)$, for all $1 \le i \le l$, then the set $S = \bigcup_{i=1}^{l} S_i$ is realizable in G, $A \cup \{c\} \subseteq S$, and $S \cap N(A) = B$.*

Proof. Let D be shorthand for $N(A)$ and $D_i = N(A_i)$ for $1 \le i \le l$.

First we observe that $S_i \cap D_i \cap D_j = (B \cup U) \cap D_i \cap D_j = S_j \cap D_i \cap D_j$ for all indices i and j with $1 \le i, j \le l$. This implies $S_i = S \cap (A_i \cup C_i \cup D_i)$ for all i.

Now we apply Lemma 2 to show that S is a realizable set of G. We consider a chordless cycle Z in $G[\![S]\!]$. Z contains a vertex $z \notin N[c]$ since $U \cup \{c\}$ is realizable in G. Let i be an index such that $z \in S_i$ and let $(b, \dots, z, \dots, d)$ be a longest subpath of Z contained in $G[\![S_i]\!]$. Then $b, d \in U$ since $C_i = C \cap \mathsf{Comp}_z(G - N[c])$. Now $(c, b, \dots, z, \dots, d)$ is a cycle in $G[\![S_i]\!]$ since $c \in A_i \subseteq S_i$ contradicting the fact that S_i is realizable. Hence such a cycle Z cannot exist and by Lemma 2 the set S is realizable in G.

By Theorem 37, $c \in A_i$ and $A_i \subseteq S_i$ for all $i = 1, \dots, k$ implies $A \cup \{c\} \subseteq S$. Finally, $S_i \cap D_i = (B \cup U) \cap D_i$ for all i, $1 \le i \le l$, implies $S \cap D = B$ since $U \subseteq N(c)$ and $N(A) \cap U = N(c) \cap B$. $\square$

Let (A, C) be a lump of G. For all $B \subseteq N(A)$ we define

$$p(A, B, C) = \max\{|S \cap C| : A \subseteq S, S \cap N(A) = B, \text{ and } S \text{ is realizable in } G\}$$

Note that $p(A, B, C) = -\infty$ if no such realizable set exists.

Lemma 39. *Let (A, C) be a lump of a graph G with $\mathsf{an}(G) = k$. Then for all $B \subseteq N(A)$ we have*

$$p(G) = \sum_{C \in \mathsf{Comp}(G)} p(\emptyset, \emptyset, C) \quad \textit{and}$$

$$p(A, B, C) = \begin{cases} \max(0, \max_{c \in C} p'(c)) & \textit{if } A \cup B \textit{ is realizable,} \\ -\infty & \textit{otherwise.} \end{cases}$$

Here

$$p'(c) = 1 + \max_{U \in \mathcal{U}(c)} \sum_{(A', C') \in \mathsf{Dec}(A, C, c)} p(A', (B \cup U) \cap N(A'), C')$$

$$\mathcal{U}(c) = \{U : U \subseteq N(c) \setminus M(c), B \cap N(c) = U \cap N(A), \textit{ and } U \cup \{c\} \textit{ is realizable in } G\}$$

Proof. The formula for $p(G)$ follows directly from $p(G[C]) = p(\emptyset, \emptyset, C)$ for all components $G[C]$ of G. Next we consider the trivial cases concerning the formula for $p(A, B, C)$. If there is no realizable set S with $A \cup B \subseteq S$, then $p(A, B, C) = -\infty$. Otherwise, if $C \cap S = \emptyset$ for all these realizable sets S, then $p(A, B, C) = 0$. In the formula we have $\max_{c \in C} p'(c) = -\infty$ if $C = \emptyset$. Otherwise for every $c \in C$ there is a block $(A', C') \in \mathsf{Dec}(A, C, c)$ such that $A' \cup (B \cap N(A))$ is not realizable, since $C \cap S = \emptyset$ for all S. Since every subset of a realizable set is realizable, this implies that $A' \cup B'$ is not realizable for all $c \in C$ and all $U \in \mathcal{U}(c)$ and $B' = (B \cup U) \cap N(A)$. Again, this implies $\max_{c \in C} p'(c) = -\infty$. Hence our formula is correct if $A \cup B$ is realizable, but $C \cap S = \emptyset$ for all realizable set S with $A \subseteq S$ and $S \cap N(A) = B$. This completes the base step of an induction.

Now we assume that a realizable set S exists with $A \subseteq S$, $S \cap N(A) = B$, and $C \cap S \neq \emptyset$. We choose S such that $|S \cap C|$ is maximum. Let c be an arbitrary vertex in $C \cap S$. Let $\mathsf{Dec}(A, C, c) = \{(A_i, C_i) : i = 1, \ldots, l\}$. Let $U = S \cap N(c) \setminus M(c)$. Then $U \in \mathcal{U}(c)$.

We consider an arbitrary lump $(A_i, C_i) \in \mathsf{Dec}(A, C, c)$ and define $B_i = (B \cup U) \cap N(A_i)$. First let $S_i = S \cap (N[A_i] \cup C_i)$. Then $p(A_i, B_i, C_i) \geq |S_i \cap C_i|$ by induction hypothesis. By induction we obtain $|S \cap C| \geq p'(c)$. This proves $p(A, B, C) \geq \max_{c \in C} p'(c)$.

To prove the other inequality we choose realizable sets $T_i \subseteq N[A_i] \cup C_i$ with $A_i \subseteq T_i$ and $T_i \cap N(A_i) = (B \cup U) \cap N(A_i)$ such that $|T_i \cap C_i|$ is maximum. By Lemma 38 the set $T = \{c\} \cup \bigcup_{i=1}^{l} T_i$ is realizable, $A \subseteq T$, and $T \cap N(A) = B$. Now $|S \cap C| \leq |T \cap C|$ since $|S_i \cap C_i| \leq |T_i \cap C_i|$ for all i. This proves $p(A, B, C) \leq \max_{c \in C} p'(c)$. □

Now it is easy to derive a recursive algorithm from the formulas in Lemma 39. We consider the running time of our algorithm on an input graph $G = (V, E)$ with $|V| = n$, $|E| = m$, and $\mathsf{an}(G) = k$. We use a data structure to memorize values of $p(A, B, C)$ already computed. This data structure supports the following operations:

- `store`(A, B, C, p) stores the value p for the lump (A, C) and the set $B \subseteq N(A)$,
- `present`(A, B, C) returns **true**, if an operation `store`(A, B, C, p) has been performed before, for any value of p, and **false** otherwise, and
- `value`(A, B, C) returns the value p of the (last) `store`(A, B, C, p) operation, if `present`$(A, B, C) =$ **true**.

All three operations can be executed by iterated search for a vertex in the universe V. A single search can be done in time $O(\log n)$ by standard techniques. To find a whole triple (A, B, C) we need at most $|A| + |B| + 1$ single searches. If (A, C) is a lump, then A, B, and C are pairwise disjoint and consequently $|A| + |B| + 1 \leq n$. This implies that the operations `store`, `present`, and `value` can be executed in time each of $O(n \log n)$.

Consider the algorithm in Table 1 (on Page 23).

This algorithm calls `value`(A, B, C) only if `present`$(A, B, C) =$ **true**. Furthermore if `store`(A, B, C) is called, then we have `present`$(A, B, C) =$ **false**,

```
procedure main;
begin
  p ← 0;
  for C ∈ Comp(G) do p ← p + access(∅, ∅, C);
  return(p)
end.

procedure access(A, B, C);
begin
  if not present(A, B, C) then compute(A, B, C);
  return(value(A, B, C))
end;

procedure compute(A, B, C);
begin
  if A ∪ B is realizable in G
    then
      begin
        p ← 0;
        for c ∈ C do
          begin
            Q ← {B ∩ N(c)};
            while Q ≠ ∅ do
              begin
                choose a set U ∈ Q of minimum cardinality;
                Q ← Q \ {U};
                r ← 1 + |U \ N(A)|;
                for (A', C') ∈ Dec(A, C, c) do
                  r ← r + access(A', (B ∪ U) ∩ N(A'), C');
                p ← max{p, r};
                for x ∈ N(c) \ (U ∪ M(c) ∪ N(A)) do
                  if U ∪ {c, x} is realizable then Q ← Q ∪ {U ∪ {x}};
              end
          end
      end
    else p ← −∞;
  store(A, B, C, p)
end;
```

Table 1. The algorithm

i.e., for each triple (A, B, C), `store` is called at most once. The number of such triples is bounded by the number of lumps (A, C) times the number of subsets $B \subseteq N(A)$. By Lemma 34, $|B \cap N(a)| \leq k^2$ for every $a \in A$. By Lemma 36 we have to evaluate `compute` for at most $O(2^{k^3} n^k)$ triples (A, B, C).

Finally we consider the running time of a single call of $\texttt{compute}(A, B, C)$ without counting the running time of those recursive calls $\texttt{compute}(A', B', C')$ for which $\texttt{present}(A', B', C') = \textbf{false}$ when $\texttt{compute}(A', B', C')$ is called. We consider at most n vertices $c \in C$. By Lemma 34 we know $|U| \leq k^2$ for every $U \in \mathcal{U}(c)$. Hence $\mathcal{U}(c)$ can be computed in time $O(2^{k^2} n)$. Clearly $|\mathsf{Dec}(A, C, c)| < n$. Consequently, we have at most $2^{k^3} n^2$ calls $\texttt{access}(A', B', C')$. Thus the total time for one call of `compute` is $O(2^{k^3} n^3 \log n)$.

Theorem 40. *There is an algorithm to solve the degree-preserving spanning tree problem for any graph G in time $O(2^{k^3} n^{k+3} \log n)$, where $k = \mathsf{an}(G)$.*

9 Open problems

It follows from Theorem 6 that the DPST problem is NP-complete for the class of bipartite planar graphs with maximum degree six. It can be easily seen that a proper subclass of this class is the class of grid graphs (a grid graph is a vertex induced finite subgraph of the infinite grid). So an interesting question is whether the DPST problem restricted to grid graphs remains NP-complete.

Of course, of practical interest is the question whether finding maximum realizable sets of a certain type, such as paths, is tractable in a more general sense and in how far these solutions can help to approximate the optimal solution for the general problem in practical situations.

References

1. Aaron, M. and M. Lewinter, 0-deficient vertices of spanning trees, *NY Acad. Sci. Graph Theory Notes* **XXVII**, (1994), pp. 31–32.
2. Arnborg S., J. Lagergren and D. Seese, Easy problems for tree-decomposable graphs, *Journal of Algorithms* **12**, (1991), pp. 308–340.
3. Baker, B. S., Approximation algorithms for NP-complete problems on planar graphs, *J. ACM* **41**, (1994), pp. 153–180.
4. Bellare, M., O. Goldreich and M. Sudan, Free bits, PCPs and non-approximability - towards tight results, *SIAM J. Comput.* **27** (1998), pp. 804–915.
5. Bienstock, D. and C. L. Monma, On the complexity of embedding planar graphs to minimize certain distance measures, *Algorithmica* **5**, pp. 93–109.
6. Bodlaender, H. L., A linear-time algorithm for finding tree-decompositions of small treewidth, *SIAM J. Comput.* **25**, (1996), pp. 1305–1317.
7. Bocchi, T., D. Gagliardi and M. Lewinter, K-deficient spanning trees, *NY Acad. Sci. Graph Theory Notes* **XXIX**, (1995), pp. 42–43.
8. Booth, K. S. and G. S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *Journal of Computer and System Sciences* **13** (1976), pp. 335–379.

9. Broersma, H. J., T. Kloks, D. Kratsch and H. Müller, Independent sets in asteroidal triple-free graphs, *Proceedings of the 24th International Colloquium on Automata, Languages and Programming, ICALP'97*, Springer-Verlag, Lecture Notes in Computer Science 1256, (1997), pp. 760–770.
10. Camerini, P. M., G. Galbiati and F. Maffioli, Complexity of spanning tree problems: Part I, *Eur. J. Oper. Res.* **5**, (1980), pp. 346–352.
11. Camerini, P. M., G. Galbiati and F. Maffioli, The complexity of weighted multi-constrained spanning tree problems, *Colloq. Math. Soc. Janos Bolyai* **44**, (1984), pp. 53–101.
12. McConnell, R. M. and J. P. Spinrad, Linear time transitive orientation, *Proceedings of SODA'97*, pp. 19–25.
13. Corneil, D.G., S. Olariu and L. Stewart, Asteroidal triple-free graphs, *SIAM Journal on Discrete Mathematics* **10** (1997), pp. 399-430.
14. Coppersmith, D. and S. Winograd, Matrix multiplication via arithmetic progressions, *J. Symb. Comput.* **9** (1990), pp. 251–280.
15. Damaschke, P., Degree-preserving spanning trees and coloring bounded degree graphs, Manuscript 1997.
16. Dell'Amico, M., M. Labbé and F. Maffioli, Complexity of spanning tree problems with leaf-dependent objectives, *Networks* **27**, (1996), pp. 175–181.
17. Fluiter, B. de, *Algorithms for graphs of small treewidth*, PhD Thesis, Utrecht University, Utrecht, The Netherlands, 1997.
18. Garey, M. R. and D.S. Johnson, *Computers and Intractability: A guide to the Theory of NP-completeness*, Freeman, New York, 1979.
19. Golumbic, M. C., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
20. Håstad, J., Clique is hard to approximate within $n^{1-\epsilon}$, Proc. 37th Ann. IEEE Symp. on Foundations of Comput. Sci., (1996), IEEE Computer Society, pp. 627–636.
21. Kloks, T., *Treewidth–Computations and Approximations*, Springer-Verlag, LNCS 842, (1994).
22. Leeuwen, J. van, Graph algorithms. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, A: Algorithms and Complexity*, Elsevier Science Publ., Amsterdam, 1990, pp. 527–631.
23. Lekkerkerker, C. G. and J. Ch. Boland: Representation of a finite graph by a set of intervals on the real line, *Fundamenta Mathematicae* **51** (1962), pp. 45-64.
24. Kratsch, D. and L. Stewart, Domination on cocomparability graphs, *SIAM Journal on Discrete Mathematics* **6**, (1993), 400-417.
25. Lewinter, M., Interpolation theorem for the number of degree-preserving vertices of spanning trees, *IEEE Trans. Circ. Syst.* **CAS-34**, (1987), 205.
26. Lewinter, M. and M. Migdail-Smith, Degree-preserving vertices of spanning trees of the hypercube, *NY Acad. Sci. Graph Theory Notes* **XIII**, (1987), 26–27.
27. Pothof, I. W. M. and J. Schut, *Graph-theoretic approach to identifiability in a water distribution network*, Memorandum 1283, Faculty of Applied Mathematics, University of Twente, Enschede, the Netherlands, (1995).
28. Rahal, H., A co-tree flows formulation for steady state in water distribution networks, *Adv. Eng. Softw.* **22**, (1995), pp. 169–178.
29. Walter, E., *Identifiability of state space models with applications to transformation systems*, Springer-Verlag, New York NY, USA, 1982.

This article was processed using the LaTeX macro package with LLNCS style