# Generating maximal spanning forests for some geometric graph classes

Daniel Meister

Report No. 393 September 2006

Preprint-Reihe
Fakultät für Mathematik und Informatik
Julius-Maximilians-Universität Würzburg

**Abstract**

A spanning tree of a connected graph $G$ is a subgraph of $G$ on the vertex set of $G$ and a tree. A maximal spanning forest of a possibly disconnected graph is the disjoint union of spanning trees of its connected components. We give highly efficient algorithms for generating maximal spanning forests for different graph classes that have a geometric representation model. The running times of our algorithms depend only on the numbers of vertices and not on the numbers of edges, which is faster for graphs with many edges than usual linear time for graph algorithms. The most general graph classes that we consider are multi-level trapezoid graphs and multi-level overlap-trapezoid graph. For a number $d \geq 0$, a $d$-trapezoid can be represented as a $(d+1)$-tuple of closed intervals of the real line: on $d+1$ horizontal lines, the intervals are marked and two consecutive intervals determine a trapezoid. The union of these trapezoids is a $d$-trapezoid. By definition, 0-trapezoids are intervals. A $d$-trapezoid graph is a graph whose vertices correspond to $d$-trapezoids such that two vertices are adjacent if and only if the corresponding $d$-trapezoids have a non-empty intersection. These graphs were introduced by Flotow as a generalisation of trapezoid graphs. A $d$-overlap-trapezoid graph is a graph whose vertices correspond to $d$-trapezoids such that two vertices are adjacent if and only if the corresponding $d$-trapezoids have a non-empty intersection and the intersection is properly contained in both $d$-trapezoids (the two $d$-trapezoids overlap). These graphs are introduced here. Permutation and interval graphs are $d$-trapezoid graphs, and permutation and overlap graphs are $d$-overlap-trapezoid graphs. Our algorithms for multi-level trapezoid and overlap-trapezoid graphs decompose the input graphs, solve the maximal spanning forest generation problems for permutation, interval and overlap graphs and recompose the partial solutions to a maximal spanning forest of the input graph. All algorithms require the input graphs represented by their canonical geometric representations instead of usual adjacency lists representation.

# Generating maximal spanning forests for some geometric graph classes

Daniel Meister
Theoretische Informatik
Julius-Maximilians-Universität Würzburg
97074 Würzburg, Germany
meister@informatik.uni-wuerzburg.de

**1 Introduction** A fundamental problem in graph algorithms is the decomposition of a given graph into its connected components. Connected components are maximal connected subgraphs, i.e., connected subgraphs that are not proper subgraphs of other connected subgraphs. Standard procedures just apply simple graph search algorithms, like breadth first search or depth first search, to generate the vertex partition, that induces the decomposition into the connected components. For general graphs, represented by adjacency lists, linear-time algorithms are the best that can be expected. Here, "linear time" means that the algorithm's running time is proportional to the numbers of vertices and edges of the input graphs.

Besides adjacency lists representation, special graph classes provide other representation models. Very popular are interval graphs: every vertex corresponds to a closed interval of the real line, and two vertices are adjacent if and only if the corresponding intervals have a non-empty intersection. The family of intervals is called an interval model for the graph. The main advantage of this model is a concise representation of the graph, since it uses only space proportional to the number of vertices. The major disadvantage is the difficulty of finding the adjacent vertices. It is clear that a usual adjacency lists representation of an interval graph can be computed in time proportional to the number of vertices and edges from the interval model representation. But we are interested in faster algorithms.

A forest is a graph without cycles. The connected components of forests are trees. A spanning forest of a graph $G$ is a subgraph of $G$ that is defined on the vertex set of $G$ and a forest. A maximal spanning forest of $G$ is a spanning forest of $G$ that is not proper subgraph of another spanning forest of $G$. Maximal spanning forests provide two types of information: the connected components of the original graph and a neighbour of every vertex, if there exists one. Since traversing a tree takes linear time, given a maximal spanning forest of a graph, the vertex partition induced by the connected components can be computed in time proportional to the number of vertices. We will call this running time *vertex-linear* to emphasise that it does not depend on the number of edges. Another interesting application of maximal spanning forests is the computation of 2-colourings of bipartite graphs. It is an easy observation that a colouring with two colours for a maximal spanning forest of a bipartite graph is a 2-colouring for the maximal spanning forest if and only if it is a 2-colouring for the original bipartite graph. By 2-colouring, we mean a partition of the vertex set into two sets such that adjacent vertices are not contained in the same partition set, whereas a colouring with two colours only means an arbitrary partition into two sets. If the input graph is not 2-colourable, a 2-colouring of a maximal spanning forest "approximates" a 2-colouring of the original graph and may still be desirable. Fast algorithms of this kind are interesting especially, if the coloured graph is only an auxiliary graph and larger than the originally treated graph. Other applications are thinkable.

Our problem, generating maximal spanning forests in vertex-linear time, requires efficient graph representations. We restrict to geometric graph classes. Geometric graph classes have geometric models for graph representation. Interval models are geometric models. Another well-known geometric model is the permutation diagram for permutation graphs. Without giving a precise definition here, that can be found in Golumbic's book [6], permutation diagrams are line segments in the Euclidean plane where all start points have the same $y$-axis coordinate and all end points have the same $y$-axis coordinate. Each line segment corresponds to a vertex, and two vertices are adjacent if and only if the corresponding line segments intersect. Such graphs are called permutation graphs, where the name is motivated by another representation, that will be defined later in this paper. For interval graphs and permutation graphs, we will give vertex-linear-time algorithms for generating maximal spanning forests. These algorithms are easy but not trivial. Even though it is easy to decompose the vertex sets of permutation and interval graphs according to the connected components, the generation of edges is not trivial, and the correctness proofs require non-trivial ideas.

Another graph class that we will consider is less popular than the two others. It is the class of overlap graphs. Overlap graphs can be defined similar to interval graphs, i.e., using interval models, but the property representing adjacency is not only intersection but overlapping. Two intervals overlap if they have a non-empty intersection and their intersection is properly contained in the two intervals. In case of interval graphs, the intersection may be also equal to an interval. Gavril showed that the class of overlap graphs is equal to the class of circle graphs [5]. A circle graph is the intersection graph of the following model: every vertex corresponds to a chord of a circle, and two vertices are adjacent if and only if the corresponding chords intersect. An analogy to permutation diagrams is obvious: instead of line segment endpoints on two lines, line segment endpoints lie on a circle. So, it is not hard to see that permutation graphs are also circle graphs, hence overlap graphs. In a certain sense, the circle model for circle graphs has 2-dimensional properties whereas permutation diagram and interval model are only 1-dimensional models. Compared to permutation graphs and interval graphs, it is already "difficult" to determine the connected components of an overlap (or circle) graph given only the model. It follows from the work by Kloks, Kratsch, Wong about the minimum fill-in problem for circle graphs that the decomposition of a circle graph into its connected components can be represented by additional chords in the circle model [7]. These chords have endpoints only between endpoints of regular chords, and they do not intersect with any regular chord. However, it is not clear whether already one single such chord can be determined in vertex-linear time. Our algorithm for generating a maximal spanning forest of an overlap graph will therefore be very complex, and the construction runs in several steps.

A generalisation of permutation and interval graphs are trapezoid graphs: every vertex corresponds to a trapezoid, they are placed between two parallel horizontal lines, and two vertices are adjacent if and only if the corresponding trapezoids have non-empty intersections. The line segments of permutation diagrams can be interpreted as parallelograms to fit into this representation, and intervals of interval models can be interpreted as rectangles. Flotow generalised trapezoid graphs to multi-level trapezoid graphs, which are the intersection graphs of families of multi-level trapezoids [3]. A 0-trapezoid is a closed interval of the real line, a 1-trapezoid is a trapezoid with the parallel sides arranged horizontally, and, for $d \geq 1$, a $(d+1)$-trapezoid is composed of a $d$-trapezoid and a 1-trapezoid such that their intersection is exactly a horizontal side of the two composition trapezoids. We assume that the 1-trapezoids have unit height. For a number $d \geq 0$, a $d$-trapezoid graph has a $d$-trapezoid for every vertex, and two vertices are adjacent if and only if the corresponding $d$-trapezoids intersect. In a similar fashion, we extend overlap graphs to multi-level overlap-trapezoid graphs. For a number $d \geq 0$, a $d$-overlap-trapezoid graph has a $d$-trapezoid for every vertex, and two vertices are adjacent if and only if the corresponding $d$-trapezoids intersect and the intersection is not equal to one of the $d$-trapezoids. We will see that we can use the maximal spanning forest generation algorithms for permutation, interval and overlap graphs to obtain vertex-linear-time algorithms for generating maximal spanning forests for $d$-trapezoid and $d$-overlap-trapezoid graphs for every (fixed) number $d \geq 0$. The main idea is to decompose the graphs into easier (permutation or interval or overlap) graphs for which the maximal spanning forests are generated and to recompose these spanning forests to maximal spanning forests of the original graphs.

The outline of the paper is as follows. In the following section, basic and necessary graph notions are defined. Especially, spanning forests will be considered. In Section 3, we present the general algorithm for generating a maximal spanning forest of a graph from maximal spanning forests of a sequence of its subgraphs. This sequence must be a decomposition of the given graph, which means in our case that every edge of the given graph is contained in at least one of the decomposition graphs. In Section 4, the algorithms for permutation and interval graphs will be presented and some interesting properties will be discussed. In Section 5, the maximal spanning forest generation algorithm for overlap graphs will be presented. These three maximal spanning forest generation algorithms are used as subroutines in Section 6 for generating maximal spanning forests for $d$-trapezoid and $d$-overlap-trapezoid graphs. The paper is concluded with final remarks in Section 7.

**2 Graph-theoretic preliminaries and maximal spanning forests** An *undirected graph* $G$ is an ordered pair $(V, E)$ of sets where $V$ is called *vertex set* of $G$ and $E$ is called *edge set* of $G$ and $E$ is a set of 2-elementary subsets of $V$. For an arbitrary undirected graph $H$, the vertex set is denoted as $V(H)$ and the edge set is denoted as $E(H)$. We mostly consider simple, finite, undirected graphs, in particular, $V$ is finite, and *graph* always means undirected graph. In Sections 5, we will also meet directed graphs. A *directed graph* $G'$ is an ordered pair $(V', A)$ of sets where $V'$ is the vertex set and $A$ is the *arc set* of $G'$ and $A$ is a subset of $V' \times V'$. For an arc $(u', v')$ of $G'$, $u'$ is a *predecessor* of $v'$ and $v'$ is a *successor* of $u'$. Edges of $G$ are denoted as $uv$ instead of $\{u, v\}$, and if $uv$ is an edge of $G$, $u$ and $v$ are called *adjacent* in $G$. We also call $u$ and $v$ the *endpoints* of edge $uv$. Adjacent vertices are *neighbours* of each other. For an edge $uv$, vertices $u$ and $v$ are *incident* with $uv$. Let $u$ and $v$ be vertices of $G$. A sequence $P = (x_0, \ldots, x_k)$ of vertices of $G$ is a *$u, v$-path of length $k$* in $G$ if the vertices in the sequence are mutually different and $x_0 = u$ and $x_k = v$ and $x_i x_{i+1} \in E$ for every $i \in \{0, \ldots, k-1\}$. If $u$ and $v$ are adjacent in $G$, $P$ is a *cycle* in $G$. $G$ is called *connected* if, for every pair $u, v$ of vertices of $G$, there is a $u, v$-path in $G$; otherwise, $G$ is called disconnected. Similarly, a sequence $P' = (x'_0, \ldots, x'_{k'})$ of vertices of $G'$ is a *$u', v'$-path of length $k'$* in $G'$ for some vertices $u'$ and $v'$ of $G'$ if $x'_0, \ldots, x'_{k'}$ are mutually different and $x'_0 = u'$ and $x'_{k'} = v'$ and $(x'_i, x'_{i+1}) \in A$ for every $i \in \{0, \ldots, k'-1\}$. Note that $P$ is also a $v, u$-path in $G$, but $P'$ may not be a $v', u'$-path in $G'$. $G'$ is called *(weakly) connected* if, for every pair $u', v'$ of vertices of $G'$, there is a $u', v'$-path or a $v', u'$-path in

$G'$. (In *strongly connected* directed graphs, both paths exist.) Let $H = (W, F)$ be also a graph. If $V \subseteq W$ and $E \subseteq F$, $G$ is a *subgraph* of $H$. If $V = W$ and $E \subseteq F$, $G$ is a *spanning subgraph* of $H$. A *connected component* of $G$ is a subgraph of $G$ that is connected and not a proper subgraph of any other connected subgraph of $G$. The definition of subgraphs and connected components for directed graphs is analogous. For $x$ a vertex of $G$, $G-x$ denotes the maximal subgraph of $G$ that does not contain vertex $x$.

Let $G = (V, E)$ be a graph. We call $G$ a *forest* if $G$ does not contain a cycle. If $G$ is a forest and connected, $G$ is a *tree*. In particular, connected components of forests are trees. Let $H = (W, F)$ be a graph. A *spanning forest* of $H$ is a spanning subgraph of $H$ that is a forest. If $G$ is a spanning forest of $H$ and no proper subgraph of any spanning forest of $H$, $G$ is a *maximal* spanning forest of $H$. A maximal spanning forest of a connected graph is a spanning tree; in particular, maximal spanning forests of connected components are spanning trees. Maximality of a spanning forest of a graph can be tested efficiently using the connectivity relation. The *connectivity relation* of $G$ is a binary relation over $V$ containing the pairs of vertices that are joined by a path in $G$. By definition, the connectivity relation is symmetric.

**Lemma 1** *Let $G = (V, E)$ be a graph, and let $H$ be a spanning forest of $G$. $H$ is a maximal spanning forest of $G$ if and only if the connectivity relations of $G$ and $H$ are equal.*

**Proof:** Let the connectivity relations of $G$ and $H$ be unequal. Then, there is a pair $u, v$ of vertices of $H$ such that there is a $u, v$-path $P = (x_0, \ldots, x_k)$ in $G$ and there is no $u, v$-path in $H$. Hence, $u$ and $v$ are contained in different connected components $C_u$ and $C_v$ in $H$, respectively. Let $i \geq 0$ be largest such that $x_i$ is contained in the same connected component of $H$ as $u$. By definition, $i < k$. Then, adding edge $x_i x_{i+1}$ to $H$ creates a spanning forest of $G$ that properly contains $H$ as subgraph. For the converse, let $H'$ be a spanning forest of $G$ that properly contains $H$ as subgraph. Hence, there is an edge $uv$ in $H'$ that is not in $H$, and $u$ and $v$ are not connected by a path in $H$. Otherwise, $H'$ would contain a cylce. ■

Let $G = (V, E)$ be a graph. A *vertex ordering* for $G$ is a linear arrangement $\tau = \langle x_1, \ldots, x_n \rangle$ of the vertices of $G$. For a pair $u, v$ of vertices of $G$, we write $u \prec_\tau v$ if $u = x_i$ and $v = x_j$ and $i < j$ for $i, j \in \{1, \ldots, n\}$. Let $G$ be connected, and let $x_1, \ldots, x_n$ be the vertices of $G$. Let $z$ be a vertex of $G$. A *BFS-ordering for $G$ with root vertex $z$* is a vertex ordering $\sigma = \langle x_{\pi(1)}, \ldots, x_{\pi(n)} \rangle$ for a permutation $\pi$ over $\{1, \ldots, n\}$ such that $x_{\pi(n)} = z$ and, for every pair $u, v$ of vertices of $G$, $u \prec_\sigma v$ if and only if, for every $k \geq 0$, if there is a $z, u$-path of length $k$ in $G$ then there is a $z, v$-path of length at most $k$ in $G$.

For a graph, given by adjacency lists, a BFS-ordering can be generated in linear time, i.e., in time proportional to the number of vertices and edges of the graph. In general, the number of edges of a graph can be of quadratic order in the number of vertices. For trees, the number of edges is one less the number of vertices. Hence, a BFS-ordering for trees can be generated in time proportional to the number of vertices. We will call such running times *vertex-linear*, i.e., the number of edges does not contribute to the order of the running time. In fact, all algorithms presented in this paper are vertex-linear-time algorithms.

**3 Decomposed graphs** The first problem is an abstract one. Let $G$ be a graph. A *decomposition* of $G$ is a sequence $\langle H_1, \ldots, H_k \rangle$ of spanning subgraphs of $G$ such that every edge of $G$ appears in some decomposition graph. Let $F_1, \ldots, F_k$ be maximal spanning forests of $H_1, \ldots, H_k$, respectively. In particular, $F_1, \ldots, F_k$ are spanning forests of $G$, that may not be maximal. We will see that we can generate a maximal spanning forest of $G$ from $F_1, \ldots, F_k$ where the running time depends only on the number of vertices of $G$ and $k$. The main idea is to generate spanning forests of $G$ iteratively from $F_1, \ldots, F_k$ by adding edges of given spanning forest $F_i$ only between connected components of the already generated spanning forest. The lemma below presents the central subroutine. Let $G$ be a graph, and let $\gamma$ be a colouring (function) for $G$, that partitions the vertex set of $G$ into $m$ sets $\Gamma_1, \ldots, \Gamma_m$. This means that, for every vertex $x$ of $G$, $\gamma$ assigns a number from $\{1, \ldots, m\}$ to $x$ – the *colour* of $x$ –, that is denoted as $\gamma(x)$. The *reduced graph of $G$ with respect to $\gamma$*, denoted as $G[\gamma]$, is obtained from $G$ by shrinking groups of vertices to single vertices and preserving adjacencies. More formal, $G[\gamma]$ contains the vertices $1, \ldots, m$ and $i$ and $j$ are adjacent in $G[\gamma]$ if and only if there are vertices $u \in \Gamma_i$ and $v \in \Gamma_j$ that are adjacent in $G$.

**Lemma 2** *Let $G = (V, E)$ be a graph, and let $F$ be a maximal spanning forest of $G$, given by adjacency lists representation. Let $\gamma$ be a colouring for $G$, given by an array representation. Then, $\mathfrak{F} =_{\text{def}} F[\gamma]$ is a maximal spanning forest of $G[\gamma]$, and it can be generated in vertex-linear time. Additionally, the edges of $\mathfrak{F}$ are labelled with pairs of vertices of $G$ that imply the edges.*

**Proof:** Let $G$ have $n$ vertices. At first, we show that $G[\gamma]$ can be computed from $G$ by iteratively glueing together pairs of vertices. We will see that a slightly modified variant of the same operations on $F$ always produce maximal spanning forests of the intermediate graphs. The main part of the proof is devoted to the algorithm, that processes $F$ according to the developed approach.

Let $c \in \{1, \ldots, m\}$, and let $a$ and $b$ be vertices of $G$ such that $\gamma(a) = \gamma(b) = c$. Let $G'$ be the graph obtained from $G$ by deleting $b$ and making every neighbour of $b$ in $G$ adjacent with $a$. We say that *$a$ and $b$ are glued together* or that we *glue together $a$ and $b$*. Let $\gamma'$ and $\Gamma'_1, \ldots, \Gamma'_m$ be the restrictions of $\gamma$ and $\Gamma_1, \ldots, \Gamma_m$ to the vertex set of $G'$, respectively. We show that $G[\gamma] = G'[\gamma']$. By the choice of $a$ and $b$ and the definition of $G'$, $\Gamma'_1, \ldots, \Gamma'_m$ are non-empty sets and $G'[\gamma']$ has $m$ vertices. For every pair $x, z$ of vertices of $G[\gamma]$, $x \neq z$, it holds:

$$\begin{aligned} xz \in E(G[\gamma]) \quad &\Longleftrightarrow \quad \text{there are vertices } u \in \Gamma_x \text{ and } v \in \Gamma_z \text{ such that } uv \in E \\ &\Longleftrightarrow \quad \text{there are vertices } u \in \Gamma'_x \text{ and } v \in \Gamma'_z \text{ such that } uv \in E(G') \\ &\Longleftrightarrow \quad xz \in E(G'[\gamma']) \,. \end{aligned}$$

Especially the second equivalence has to be verified carefully. It follows that $G[\gamma]$ can be obtained from $G$ by glueing together vertices of the same colour as long as possible and finally renaming the vertices. It is to emphasise that the order in which pairs of vertices are glued together does not influence the result. We call the computation of $G[\gamma]$ from $G$ in the described way the *shrinking process* for $G[\gamma]$.

We want to perform the shrinking process on $F$ directly. However, we cannot just glue together vertices as long as possible and reduce the obtained graph to a forest by deleting edges, since many edges could get new endpoints too often. That is why we modify the above-described operation "glue together", such that, performed on $F$, the result is always a forest. Let $a$ and $b$ be vertices of $G$ with the same colour. Since $F$ is cycle-free, at most one neighbour of $b$ can be vertex in a connected component of $F-b$ with $a$, and this is only the case if $a$ and $b$ are contained in the same connected component of $F$. So, if $a$ and $b$ are not contained in the same connected component of $F$, obtain $F'$ from $F$ by glueing together $a$ and $b$, as defined above. Otherwise, if $a$ and $b$ are contained in the same connected component of $F$, let $z$ be the neighbour of $b$ that is contained in the connected component of $F - b$ containing $a$. Obtain $F'$ from $F$ by deleting $b$ and making every neighbour of $b$ but $z$ adjacent with $a$. In both cases, $F'$ is a forest. Furthermore, $F'$ is a spanning forest of $G'$, that is obtained from $G$ by glueing together $a$ and $b$. For maximality of $F'$, it suffices to observe that the connected components of $F$ correspond to the connected components of $G$ due to Lemma 1, and the connected components of $F'$ correspond to the connected components of $G'$.

Our algorithm for generating a maximal spanning forest of $G[\gamma]$ is a sequence of operations "glue together" on $F$. The results above show correctness of the approach. It remains to ensure the running-time restriction, which mainly means that we have to find out in short time whether a neighbour of deleted vertex $b$ cannot become a neighbour of $a$ and, if so, which one. The construction is partitioned into two phases:

Phase (1) Let $C_1, \ldots, C_k$ be the connected components of $F$. We determine which of these connected components connect to a single connected component during the shrinking process.

Phase (2) Compute a maximal spanning forest of $G[\gamma]$.

In other words, in Phase (1), the vertex set of $F[\gamma]$ is partitioned according to its connected components, and in Phase (2), vertices with the same colours are glued together to obtain the connected components of $F[\gamma]$. The algorithm for Phase (1) implements the following property: two connected components $C_1$ and $C_2$ of $F$ contribute to the same connected component of $F[\gamma]$ if and only if there is an iteration step during the shrinking process such that $C_1$ contributes to connected component $C_1'$ and $C_2$ contributes to connected component $C_2'$ of $F'$, that is the intermediate spanning forest in this iteration step, and $C_1'$ and $C_2'$ have vertices of the same colour. Correctness follows directly from the results above. We define a forest $B$ on $k$ vertices $1, \ldots, k$, that correspond to the connected components $C_1, \ldots, C_k$ of $F$, respectively, and process the connected components of $F$ according to a breadth first search strategy for $B$, that is computed during execution:

(a) mark all vertices of $B$ and all colours as *unvisited*;
initialize an empty queue

(b) repeat until all vertices of $B$ are visited:

- (c) let $i$ be the vertex of $B$ that is to be treated next: if a vertex is contained in the queue, take the first one, if the queue is empty, choose an unvisited vertex;
  mark $i$ as visited
- (d) for all vertices $x$ of $C_i$:
  - (e) let $\gamma(x) = c$, and let colour $c$ be unvisited
    (otherwise continue with the next vertex of $C_i$)
  - (f) for all vertices $j$ of $B$:
    - (g) if (c1) $C_j$ contains a vertex with colour $c$ and (c2) $j$ is unvisited, add edge $ij$ to $B$, mark vertex $j$ and colour $c$ as visited and insert $j$ as last element into the queue.

For an efficient implementation, the *visited* properties for vertices and colours are kept in arrays. Every colour class is represented by a chained list, that can be computed from $\gamma$

in time $\mathcal{O}(n)$. Every vertex of $F$ is treated only once as vertex of a connected component of $F$ and only once as member of a colour class. The loop in algorithm line (f) does not consider every vertex of $B$ but, due to condition (c1) in algorithm line (g), only vertices of the colour class $\Gamma_c$ by running through the list.

The vertices of $B$ are processed in a breadth first search manner, where start vertices of the single connected components are chosen arbitrarily. It follows that $B$ is a forest. Let $B_1, \ldots, B_r$ be the connected components of $B$. We observe that, for every colour $c \in \{1, \ldots, m\}$, there is exactly one $i \in \{1, \ldots, r\}$ such that there is $j \in B_i$ and $C_j$ contains a vertex with colour $c$. So, the connected components of $B$ define a partition of the set of colours, and due to the shrinking process, this partition exactly corresponds to the set of connected components of $F[\gamma]$.

We finish Phase (1) as follows. Every edge of $B$ is labelled with the pair of vertices of $F$ that defined the edge in the algorithm: the one vertex is $x$ from algorithm line (e) and the other vertex is a vertex of $C_j$ with the same colour as $x$ ((c1) in algorithm line (g)). Then, we glue together the vertex pairs the edges of $B$ are labelled with. Note that every vertex of $F$ can be contained in at most one pair, since every edge corresponds to its own colour. Since the vertices of an operation "glue together" always belong to different connected components of the intermediate graph, the operation yields another forest. After carrying out all operations "glue together", let $M$ be the resulting graph. $M$ is still a forest, and remember that every colour occurs in exactly one connected component of $M$, but may have multiple occurrences. In Phase (2), we reduce $M$ such that every colour occurs exactly once, and by renaming the vertices, we will obtain $F[\gamma]$.

We describe the shrinking process on one connected component of $M$. For all further connected components of $M$, it is carried out analogously. Let $M_C$ be a connected component of $M$. Compute a breadth first search ordering $\sigma_C = \langle x_1, \ldots, x_s \rangle$ for $M_C$, starting with any vertex. For every colour $c \in \{1, \ldots, m\}$ in $M_C$, let $a_c$ be the vertex rightmost with respect to $\sigma_C$ with colour $c$. (The vertex the breadth first search started with is the vertex rightmost with respect to $\sigma_C$.) We reduce $M_C$ by glueing together pairs of vertices in order defined by $\sigma_C$. For every $i \in \{0, \ldots, s\}$, we define a graph $M_C^{(i)}$, and $M_C^{(s)}$ will be the result. Let $M_C^{(0)} =_{\text{def}} M_C$. Let $i \in \{1, \ldots, s\}$, and let $M_C^{(i-1)}$ be defined. If $x_i = a_{\gamma(x_i)}$, let $M_C^{(i)} =_{\text{def}} M_C^{(i-1)}$. Otherwise, i.e., if $x_i$ is to the left of $a_{\gamma(x_i)}$ with respect to $\sigma_C$, let $M_C^{(i)}$ emerge from $M_C^{(i-1)}$ by glueing together $a_{\gamma(x_i)}$ and $x_i$. The neighbour of $x_i$ that is rightmost with respect to $\sigma_C$ is the only neighbour of $x_i$ that is contained in the same connected component as $a_{\gamma(x_i)}$ in $M_C^{(i-1)} - x_i$. The neighbours of $x_i$ in $M_C^{(i-1)}$ that are not contained in the same connected component of $M_C^{(i-1)} - x_i$ as $a_{\gamma(x_i)}$ are only vertices from $\{a_i : i \text{ is a colour of } M_C\}$. Since $M_C$ is a tree and $\sigma_C$ is a breadth first search ordering for $M_C$ and edges obtain a new endpoint at most once, the described algorithm runs in vertex-linear time with respect to $M_C$, which makes total vertex-linear time for generating $\mathfrak{F}$.

By definition of $\mathfrak{F}$, for every edge $ab$ of $\mathfrak{F}$, there is an edge of $F$ whose incident vertices have the colours $a$ and $b$. The labels attached to the edges of $\mathfrak{F}$ can be obtained by either of the following two procedures: the first one is to label the edges of $F$ and during the shrinking process keep the edge labels, the second is to attach the labels after generation of $\mathfrak{F}$, where, for every vertex of $\mathfrak{F}$, only edges of $F$ with this colour are considered. Both procedures require only vertex-linear additional time. ∎

We use the algorithm of Lemma 2 to generate maximal spanning forests of graphs for which only a sequence of maximal spanning forests of subgraphs is known. The subgraphs must represent a decomposition of the original graph, which means that every edge of the original graph is contained in a decomposition graph. In general, one can say that a decomposition decomposes a graph into smaller and easier graphs, and the main goal is to obtain solutions for problems on the original graphs by solving problems on the smaller graphs and recompose them to solutions for the original graphs. Among the most well-known types of decomposition is modular decomposition (see, for instance, the article by Gallai [4]). Let $G = (V, E)$ and $H = (W, F)$ be graphs. The *union* of $G$ and $H$, denoted as $G \cup H$, is defined as the ordered pair of the vertex-set and edge-set union, i.e., $G \cup H =_{\text{def}} (V \cup W, E \cup F)$. If the vertices of $G$ are vertices of $H$, $G \cup H$ means only adding the edges of $G$ to $H$. The definition of graph union for two graphs is extended to arbitrarily many graphs in the usual way.

**Theorem 3** *Let $G = (V, E)$ be a graph on $n$ vertices. Let $\langle H_1, \ldots, H_k\rangle$ be a decomposition of $G$, such that $G = H_1 \cup \cdots \cup H_k$. Let $F_1, \ldots, F_k$ be maximal spanning forests of $H_1, \ldots, H_k$, respectively, given by adjacency lists representation. Then, a maximal spanning forest of $G$ can be generated in time $\mathcal{O}(kn)$.*

**Proof:** Let $k \geq 2$. We assume that $F_1, \ldots, F_k$ are spanning subgraphs of $G$, which can be achieved in time $\mathcal{O}(kn)$ by adding the missing vertices otherwise. We show that we can generate a maximal spanning forest of $H_1 \cup H_2$ from $F_1$ and $F_2$ in vertex-linear time. Iterated application yields the theorem. Let $C_1, \ldots, C_r$ be the connected components of $F_1$. Every $C_i$ is subgraph of a connected component of $H_1 \cup H_2$. Let $\gamma$ be the vertex colouring for $H_1$ that partitions $F_1$ into its connected components; we assume that the vertices of $C_i$ have colour $i$.

**Claim A** *For every $a, b \in \{1, \ldots, r\}$, $C_a$ and $C_b$ are contained in the same connected component of $H_1 \cup H_2$ if and only if there is a path between $a$ and $b$ in $F_2[\gamma]$.*

Let $P = (x_0, \ldots, x_s)$ be an $a,b$-path in $F_2[\gamma]$. Due to definition of $F_2[\gamma]$, $x_i x_{i+1}$ corresponds to an edge of $F_2$ between a vertex of $C_{x_i}$ and a vertex of $C_{x_{i+1}}$, $i \in \{0, \ldots, s-1\}$. Hence, for every pair of vertices of $C_a$ and $C_b$, there is a path in $H_1 \cup H_2$. For the converse, let $P'$ be a path in $H_1 \cup H_2$ between a vertex of $C_a$ and a vertex of $C_b$. Let $G_{P'}$ be the subgraph of $G$ containing only the vertices and edges of $P'$. Then, $G_{P'}[\gamma]$ is a connected subgraph of $H_2[\gamma]$, and there is an $a,b$-path in $F_2[\gamma]$.

So, in vertex-linear time, $\gamma$ can be computed from $F_1$. Using Lemma 2, a maximal spanning forest $\mathfrak{F}_2$ of $F_2[\gamma]$ can be generated in vertex-linear time, and the edges of the forest are labelled with pairs of vertices of $F_2$ implying the edges. A maximal spanning forest of $H_1 \cup H_2$ is obtained from $F_1$ by adding the edges specified by the labels attached to the edges of $\mathfrak{F}_2$. All this takes only vertex-linear time. Since $\mathfrak{F}_2$ is a forest, the generated graph is also a forest and a maximal spanning forest of $H_1 \cup H_2$ due to Claim A. ∎

As it was mentioned above, Theorem 3 becomes interesting when a graph can be decomposed fast into smaller graphs for which the maximal spanning forest problem can be solved significantly faster or easier than for the original graph. Applications of this theorem can be found at the end of the paper, where we decompose graphs of the two most general graph classes that we consider here into smaller ones.

**4 Permutation and interval graphs** Permutation graphs and interval graphs have geometric models, that define adjacency through non-empty intersection. Let $n \geq 1$ be a natural number. Let $\sigma_U$ and $\sigma_L$ be orderings for the set $\{1,\ldots,n\}$. The *permutation graph defined by* $(\sigma_U,\sigma_L)$, denoted as $G(\sigma_U,\sigma_L)$, is defined as follows:

(vp) $\{1,\ldots,n\}$ is the vertex set of $G(\pi)$

(ep) $uv$ is an edge of $G(\sigma_U,\sigma_L)$ if and only if either $u \prec_{\sigma_U} v$ or $u \prec_{\sigma_L} v$.

A graph $G$ on the vertex set $\{1,\ldots,n\}$ is a *permutation graph* if there are orderings $\sigma_U$ and $\sigma_L$ for $\{1,\ldots,n\}$ such that $G = G(\sigma_U,\sigma_L)$. The geometric interpretation is as follows: given two horizontal lines, one of which is called *upper* line, the other is called *lower* line, mark $n$ different points on each line and label the points on the upper line with $1,\ldots,n$ in order defined by $\sigma_U$ and label the points on the lower line with $1,\ldots,n$ in order defined by $\sigma_L$. Connect points with the same labels by line segments, and label the line segments with the labels of their endpoints. The set of line segments is called the *permutation diagram* of $G(\sigma_U,\sigma_L)$. The labels of the line segments establish a correspondence between the vertices of $G(\sigma_U,\sigma_L)$ and the line segments, and two vertices of $G(\sigma_U,\sigma_L)$ are adjacent if and only if the corresponding line segments intersect. Permutation graphs have many characterisations, most of which can be found in Golumbic's book [6].

**Theorem 4** *Let $G = G(\sigma_U,\sigma_L)$ be a permutation graph, given by the pair $(\sigma_U,\sigma_L)$ of orderings that are represented by arrays. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** Let $G$ have $n$ vertices. Let $A = \{a_1,\ldots,a_k\}$ be the first-fit maximal independent set of $G$, where we assume $a_1 \prec_{\sigma_U} \cdots \prec_{\sigma_U} a_k$: $a_1$ is leftmost with respect to $\sigma_L$, and, for every $i \in \{2,\ldots,k\}$, $a_i$ is leftmost with respect to $\sigma_L$ among all vertices to the right of $a_{i-1}$ with respect to $\sigma_U$. For every vertex $x$ of $G$, let $m_x$ be the vertex among $\{u : x \prec_{\sigma_L} u\} \cup \{x\}$ that is leftmost with respect to $\sigma_U$. We define graph $\mathfrak{F}$ on the vertex set of $G$ and with the following edges:

(1) for every vertex $x$ of $G$ and every $i \in \{2,\ldots,k\}$, if $x \prec_{\sigma_U} a_1$ then $a_1x$ is an edge of $\mathfrak{F}$, if $a_{i-1} \prec_{\sigma_U} x \prec_{\sigma_U} a_i$ then $a_ix$ is an edge of $\mathfrak{F}$

(2) for every $i \in \{1,\ldots,k\}$, if $m_{a_i} \prec_{\sigma_U} a_i$ then $a_im_{a_i}$ is an edge of $\mathfrak{F}$.

Note that the edges of $\mathfrak{F}$ are edges of $G$; hence, $\mathfrak{F}$ is a spanning subgraph of $G$. Furthermore, $\mathfrak{F}$ is a forest, since $\{m_{a_1},\ldots,m_{a_k}\}$ is an independent set in $G$. Due to Lemma 1, it remains to show that the connected components of $G$ and $\mathfrak{F}$ correspond to each other.

Let $u$ and $v$ be vertices of $G$, $u \prec_{\sigma_U} v$, and let there be a $u,v$-path in $G$. Let $\mathfrak{F}'$ be the spanning subgraph of $\mathfrak{F}$ containing only the edges defined by edge definition step (1). We assume that $u$ and $v$ are not contained in the same connected component of $\mathfrak{F}'$. Let $i'$ and $i$ be the numbers such that $u$ and $a_{i'}$ are contained in the same connected component of $\mathfrak{F}'$ and $v$ and $a_i$ are contained in the same connected component of $\mathfrak{F}'$. It holds that $i' < i$. Note that there is an $a_{i'},a_i$-path in $G$. It suffices to show that this implies the existence of an $a_{i'},a_i$-path in $\mathfrak{F}$. Consider $m_{a_i}$. Let $j$ be such that $a_j$ and $m_{a_i}$ are contained in the same connected component of $\mathfrak{F}'$. If $j < i$, we can conclude that $a_i$ and $m_{a_i}$ are adjacent in $\mathfrak{F}$ due to edge definition step (2), there is an $a_{i'},a_j$-path in $G$ (via $a_i$) and the existence of an $a_{i'},a_j$-path in $\mathfrak{F}$ implies the existence of an $a_{i'},a_i$-path in $\mathfrak{F}$. Iterated application yields the result (where iteration ends after finite steps since $i', j < i$). It remains to show that $j < i$ does hold. Assume that $j \geq i$. Then, $j = i$ due to definition of $m_{a_i}$. In particular, no vertex $x$ where $a_{i-1} \prec_{\sigma_U} x$ is neighbour of $a_{i-1}$ or a vertex to the left of $a_{i-1}$ in $G$ (this fact strongly relies on the definition of $A$). Since $u = a_{i-1}$ or $u \prec_{\sigma_U} a_{i-1}$, this contradicts the existence of a $u,v$-path in $G$, so that $j < i$ must hold.

For an efficient implementation of the algorithm, it is to note that set $A$ can be computed in a greedy fashion in vertex-linear time. A similar algorithm computes $m_1, \ldots, m_n$ in vertex-linear time and stores the values in an array. Using these values, the edges of $\mathfrak{F}$ can be computed straight-forward in vertex-linear time. ∎

The maximal spanning forest in the proof of Theorem 4 is defined in a greedy fashion. The algorithm is easy, but not simple or trivial. For instance, independent set $A$ cannot be replaced by an arbitrary other maximal independent set, as it is shown in the correctness proof. There are many further possibilities of defining maximal spanning forests of permutation graphs that can be generated in vertex-linear time; however, the generation algorithms become more complex. In the following, we will present a maximal spanning forest generation algorithm for interval graphs, whose results have very different structural properties compared to the permutation graphs algorithm.

Interval graphs are the intersection graphs of families of intervals. Let $\mathfrak{I}$ be a family of closed intervals of the real line. The *interval graph defined by* $\mathfrak{I}$, denoted as $G(\mathfrak{I})$, is defined as follows:

- (vi) $G(\mathfrak{I})$ has a vertex for every interval in $\mathfrak{I}$
- (ei) $uv$ is an edge of $G(\mathfrak{I})$ if and only if the intervals corresponding to $u$ and $v$ have a non-empty intersection.

A graph $G$ is an *interval graph* if there is a family $\mathfrak{I}$ of closed intervals of the real line such that $G = G(\mathfrak{I})$. Family $\mathfrak{I}$ is called an *interval model* for $G$. An interval graph has several interval models. We restrict to only special interval models. For a closed interval $[a, b]$ of the real line, $a$ and $b$ are the *endpoints* of $[a, b]$, and we distinguish *left* and *right* endpoints, denoted as $L([a, b])$ and $R([a, b])$, respectively. We consider only interval models where intervals have two different endpoints, no pair of intervals has a common endpoint and endpoints are natural numbers from the set $\{1, \ldots, 2n\}$, where $n$ is the number of intervals in the interval model. We will call such interval models *normalised*. Every interval graph has a normalised interval model.

For a set $\mathcal{I}$ of closed intervals of the real line, $\bigcup \mathcal{I}$ denotes the union of the intervals in $\mathcal{I}$. The union of intervals is called a *complete interval* if it is equal to $[a, b]$ for some numbers $a \leq b$. The following lemma is important for the correctness of our algorithms.

**Lemma 5** *Let $G = G(\mathfrak{I})$ be an interval graph. Let $H$ be a subgraph of $G$, and let $\mathcal{I}$ denote the set of intervals from $\mathfrak{I}$ that correspond to the vertices of $H$. If $H$ is connected, then $\bigcup \mathcal{I}$ is a complete interval.*

**Proof:** Let $H$ be connected. Let $\sigma$ be a BFS-ordering for $H$. Starting with the root vertex, compute $\bigcup \mathcal{I}$ by iteratively adding intervals to the union according to ordering $\sigma$. Since $H$ is connected and $\sigma$ is a BFS-ordering for $H$, every but the first added interval has a non-empty intersection with some already added interval, i.e., has a non-empty intersection with the already computed union. Hence, after adding an interval, the computed union is a complete interval. ∎

It is an interesting observation that the correctness of our permutation graphs algorithm relied on a similar property. In fact, the projection on the lower horizontal line of the line segments in a permutation diagram corresponding to a path in the defined permutation graph is a complete interval.

**Theorem 6** *Let $G = G(\mathfrak{I})$ be an interval graph, given by normalised interval model $\mathfrak{I}$. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** Let $G = (V, E)$. Let $\mathfrak{I} = \{I_x\}_{x \in V}$ and $I_x = [L(x), R(x)]$ for every vertex $x$ of $G$. Let $\tau = \langle x_1, \ldots, x_n \rangle$ be the vertex ordering for $G$ such that $u \prec_\tau v$ if and only if $L(u) < L(v)$ for every pair $u, v$ of vertices of $G$. The algorithm generates a spanning forest by assigning neighbours in a greedy fashion. For $x \in V$, $x_1 \neq x$, let $c_x$ be the vertex $v \prec_\tau x$ of $G$ with largest right endpoint $R(v)$. Then, $\mathfrak{F}$ is the graph on the vertex set of $G$ with the following edges: for every $x \in V$, $x_1 \neq x$, if $L(x) < R(c_x)$ then $c_x x$ is an edge of $\mathfrak{F}$. Note that, since $c_x \prec_\tau x$, $L(c_x) < L(x)$, and $\mathfrak{F}$ is a subgraph of $G$. It is also clear that $\mathfrak{F}$ is a forest: a cycle contains a vertex rightmost with respect to $\tau$, and this rightmost vertex must have two neighbours, contradicting the definition of $\mathfrak{F}$. It remains to show that the connected components of $\mathfrak{F}$ are exactly the connected components of $G$ due to Lemma 1.

Let $u$ and $v$ be vertices of $G$, $u \prec_\tau v$, and let $P$ be a $u, v$-path in $G$. Let $\mathcal{I}$ be the set of intervals corresponding to the vertices on $P$. Then, due to Lemma 5, $\bigcup \mathcal{I}$ is a complete interval, and $L(v) < R(c_v)$ (for correctness, assume the contrary). So, $v$ and $c_v$ are adjacent in $G$, and there is a $u, c_v$-path $P'$ in $G$. If $u = c_v$ or $c_v \prec_\tau u$, which means that $c_v = c_u$, $u$ and $v$ are in the same connected component of $\mathfrak{F}$. If $u \prec_\tau c_v$, $c_v$ and $c_{c_v}$ are adjacent in $G$, and we can repeat the argument for a $u, c_{c_v}$-path. By induction, we conclude that $u$ and $v$ are contained in the same connected component of $\mathfrak{F}$, which proves correctness of our algorithm.

It remains to consider running-time aspects. Since $\mathfrak{I}$ is a normalised interval model, ordering $\tau$ is well-defined and can be generated in vertex-linear time using an array of size $2n$. Vertices $c_x$ can also be computed in vertex-linear time, which is similar to determining local maxima. Saving vertices $c_x$ in an array, edges $c_u u$ can be determined in constant time for every vertex $u$ of $G$. ∎

Using the constructions of the algorithms for permutation and interval graphs, maximal spanning forest generation algorithms can be designed that generate maximal spanning forests with special properties. The interval graphs algorithm of Theorem 6 generates forests whose connected components have paths that see each vertex of the connected component. The following result is a consequence of this fact.

**Theorem 7** *There is a vertex-linear-time algorithm that, given an interval graph $G$ represented by a normalised interval model, generates a maximal spanning forest of $G$ represented by a normalised interval model.*

**Proof:** It suffices to consider the definitions and the algorithm given in the proof of Theorem 6 and to generate an interval model for the generated forest $\mathfrak{F}$. Let $\mathfrak{I}$ be the normalised interval model defining $G$. Let $F$ be a connected component of $\mathfrak{F}$, and let $v$ be the vertex of $\mathfrak{F}$ whose corresponding interval has the largest right endpoint among the vertices of $F$. Let $x_0 =_{\text{def}} v$. If $c_v$ and $v$ are adjacent in $\mathfrak{F}$, let $x_1 =_{\text{def}} c_v$. If $c_{x_1}$ and $x_1$ are adjacent in $\mathfrak{F}$, let $x_2 =_{\text{def}} c_{x_1}$. Continue the construction as long as possible, and we obtain a path $P = (x_0, \ldots, x_r)$ in $F$ for some number $r$. Note that $L(x_r)$ is the smallest left endpoint of an interval corresponding to a vertex of $F$, which follows from Lemma 5.

We show that every vertex of $F$ that is not on $P$ is a neighbour of a vertex on $P$ in $F$. Assume there is a vertex $u$ of $F$ that is not on $P$ such that $c_u$ is not on $P$ and $u$ and $c_u$ are adjacent in $F$. Let $u$ be chosen such that there is $i$ and $x_i = c_{c_u}$ and $x_i$ and $c_u$ are adjacent in $F$. Remember that $L(x_i) < L(c_u) < R(x_i)$ and $L(c_u) < L(u) < R(c_u)$. Since $x_i \neq c_u$, $R(x_i) < R(c_u)$ by construction, in particular, $i > 0$. Consider $x_{i-1}$. Since $c_{x_{i-1}} \neq c_u$, $L(x_{i-1}) < L(c_u)$ and $R(x_{i-1}) < R(x_i)$. Hence, $i > 1$, and $x_{i-1} \neq c_{x_{i-2}}$ in contrast to the construction.

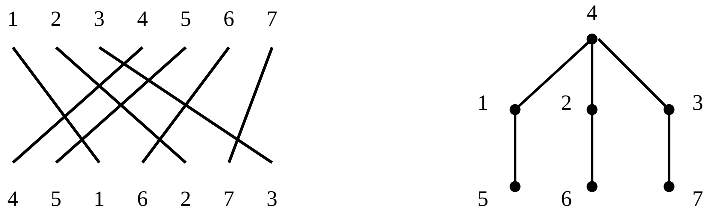


**Figure 1** On the left hand side, the permutation diagram of the permutation graph $G = G(\langle 1,2,3,4,5,6,7\rangle, \langle 4,5,1,6,2,7,3\rangle)$ is shown, and on the right hand side, the maximal spanning forest $\mathfrak{F}$ of $G$ generated by the permutation graphs algorithm of Theorem 4 is shown. Forest $\mathfrak{F}$ is not AT-free, hence not a permutation graph.

In vertex-linear time, it is easy to generate an interval model for $P$ and to add intervals for the remaining vertices of $F$ such that the intervals intersect only with the approprite intervals corresponding to the vertices of $P$. ■

To be precise, the proof of Theorem 7 shows that every connected component of the generated forest contains a dominating path. A path in a graph is called *dominating* if every vertex of the graph has distance at most 1 to some path vertex. It is known that connected AT-free graphs have dominating paths [1]. In a graph, an *asteroidal triple*, *AT* for short, is a set $A$ of three vertices such that every pair of vertices from $A$ is joined by a path that does not contain a neighbour of the third vertex. A graph is called *AT-free* if it does not contain an asteroidal triple. It is easy to see that a tree with a dominating path cannot contain an asteroidal triple. Since AT-free trees are AT-free chordal graphs, they are interval graphs ([8]).

In case of permutation graphs, our algorithm does not have the property that the generated forests are always permutation graphs. Consider the sample permutation graph $G = G(\langle 1,2,3,4,5,6,7\rangle, \langle 4,5,1,6,2,7,3\rangle)$ and the permutation graphs algorithm on $G$. The first-fit maximal independent set of $G$ is $A = \{4,5,6,7\}$. The maximal spanning forest of $G$ generated by our permutation graphs algorithm of Theorem 4 is depicted in Figure 1. Using the definitions from the proof of Theorem 4, $m_4 = m_5 = m_1 = 1$, $m_6 = m_2 = 2$, $m_7 = m_3 = 3$, and $\{4,1\}$ (since $m_4 < 4$), $\{5,1\}$ (since $m_5 < 5$), $\{6,2\}$ (since $m_6 < 6$), $\{7,3\}$ (since $m_7 < 7$) are edges of the forest. This forest is one of the most popular examples of a graph containing an asteroidal triple, formed by the vertices $5,6,7$. Hence, this graph is not a permutation graph.

**5 Overlap graphs** Overlap graphs are defined using a model similar to interval models. Let $I = [L(I), R(I)]$ and $J = [L(J), R(J)]$ be intervals of the real line. We say that $I$ and $J$ *overlap* if $L(I) < L(J) < R(I) < R(J)$ or $L(J) < L(I) < R(J) < R(I)$. In other words, two intervals overlap if they have a non-empty intersection and the intersection is properly contained in both intervals. Let $\mathfrak{I}$ be a normalised family of closed intervals of the real line, i.e., no pair of intervals has a common endpoint and the endpoints of the intervals in $\mathfrak{I}$ where $\mathfrak{I}$ contains $n$ intervals are in $\{1,\ldots,2n\}$. The *overlap graph defined by* $\mathfrak{I}$, denoted as $\Omega(\mathfrak{I})$, is defined as follows:

(vo) $\Omega(\mathfrak{I})$ has a vertex for every interval in $\mathfrak{I}$

(eo) $uv$ is an edge of $\Omega(\mathfrak{I})$ if and only if the intervals corresponding to $u$ and $v$ overlap.

A graph $G$ is an *overlap graph* if there is a (normalised) family $\mathfrak{I}$ of closed intervals of the real line such that $G = \Omega(\mathfrak{I})$. We call $\mathfrak{I}$ an *overlap model* for $\Omega(\mathfrak{I})$. As for interval graphs, an overlap graph can have several overlap models.

By the definition of overlap graphs, it is clear that, for a family $\mathfrak{I}$ of closed intervals of the real line, the overlap graph defined by $\mathfrak{I}$ is a spanning subgraph of the interval graph defined by $\mathfrak{I}$. Gavril showed that circle graphs are overlap graphs and vice versa [5], so that overlap graphs can also be considered the intersection graphs of an appropriate geometric model, like permutation and interval graphs. Circle graphs are defined by the following intersection model: every vertex corresponds to a chord of a circle, and two vertices are adjacent if and only if the corresponding chords intersect within the circle. Nevertheless, we will use overlap models for the representation of overlap graphs.

For our maximal spanning forest algorithm for overlap graphs, we need an auxiliary result. Let $\mathcal{I}$ be a set of closed intervals of the real line. A number $i$ is *endpoint* of an interval in $\mathcal{I}$, if there is $I \in \mathcal{I}$ such that $L(I) = i$ or $R(I) = i$. The *endpoint set* of $\mathcal{I}$ is the set of endpoints of the intervals in $\mathcal{I}$.

**Lemma 8** *Let $\mathfrak{I}$ be a normalised family of closed intervals of the real line. Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be disjoint subsets of $\mathfrak{I}$ that define connected overlap graphs. Let $D_1$ and $D_2$ be the endpoint sets of $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. There are intervals $I_1$ in $\mathcal{I}_1$ and $I_2$ in $\mathcal{I}_2$ such that $I_1$ and $I_2$ overlap if and only if there are numbers $a, c$ in $D_1$ and $b, d$ in $D_2$ such that $a < b < c < d$ or $b < a < d < c$.*

**Proof:** Since $\mathfrak{I}$ is normalised and $\mathcal{I}_1$ and $\mathcal{I}_2$ are disjoint, $D_1$ and $D_2$ are disjoint. It holds by definition: if $I_1 \in \mathcal{I}_1$ and $I_2 \in \mathcal{I}_2$ overlap, the criterion is fulfilled using the endpoints of $I_1$ and $I_2$. For the converse, suppose there is no pair of overlapping intervals from $\mathcal{I}_1$ and $\mathcal{I}_2$. In other words, for every pair $I_1, I_2$ of intervals where $I_1 \in \mathcal{I}_1$ and $I_2 \in \mathcal{I}_2$, either $I_1$ and $I_2$ are disjoint or one properly contains the other. We distinguish two cases. The first case: let no interval from $\mathcal{I}_1$ have a non-empty intersection with an interval from $\mathcal{I}_2$. Since $\Omega(\mathcal{I}_1)$ and $\Omega(\mathcal{I}_2)$ are connected, $\bigcup \mathcal{I}_1$ and $\bigcup \mathcal{I}_2$ are complete intervals due to Lemma 5. Hence, $\max D_1 < \min D_2$ or $\max D_2 < \min D_1$, and we conclude the lemma in the first case.

The second case is more complex: let an interval from $\mathcal{I}_1$ have a non-empty intersection with an interval from $\mathcal{I}_2$. We first show an auxiliary result.

**Claim A** *Let there be intervals $J_1 \in \mathcal{I}_1$ and $J, J_2 \in \mathcal{I}_2$ such that $J \subset J_1 \subset J_2$. Then, there is a pair of overlapping intervals in $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Let $J, J_1, J_2$ be as assumed. Let $P = (x_0, \ldots, x_k)$ be a path in $\Omega(\mathcal{I}_2)$ between the vertices corresponding to $J$ and $J_2$, and let $\mathcal{K} = \{K_0, \ldots, K_k\}$ be the set of intervals from $\mathcal{I}_2$ corresponding to the vertices on $P$, where $K_i$ corresponds to $x_i$ and $K_0 = J$. It holds that $J \subset J_1 \subset J_2 \subset \bigcup \mathcal{K}$. Similar to the proof of Lemma 5, we compute the union $\bigcup \mathcal{K}$ iteratively along $P$ starting with $J$. So, consider the sequence $\bigcup_{i \leq 0} K_i, \ldots, \bigcup_{i \leq k} K_i$. It is clear that every union in the sequence is a complete interval due to Lemma 5. There is a smallest number $j \geq 1$ such that $\bigcup_{i \leq j} K_i \not\subset J_1$. Since $K_{j-1} \subset J_1$, $K_j$ overlaps with $J_1$: $K_j$ overlaps with $K_{j-1}$ due to the properties of $P$, so that one endpoint of $K_j$ is contained in $J_1$, and since $K_j \not\subset J_1$, one of the endpoints of $\bigcup_{i \leq j} K_i$ is the other endpoint of $K_j$, which is not contained in $J_1$.

Let $I_1$ be an interval from $\mathcal{I}_1$ that has a non-empty intersection with an interval $I_2$ from $\mathcal{I}_2$. Since they do not overlap by assumption, $I_1 \subset I_2$ or $I_2 \subset I_1$. Let it be the former case; the latter case can be proved analogously. From Claim A, we conclude that every interval in $\mathcal{I}_2$ is either disjoint with $I_1$ or properly contains $I_1$. So, $I_1$ partitions $D_2$

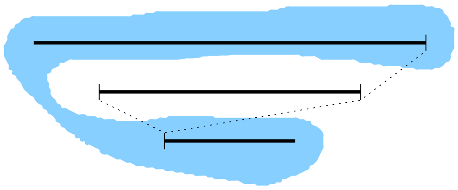

**Figure 2** An overlap model corresponding to a graph on three vertices and without any edges. We partition the intervals into two sets: the smallest and largest interval in the one set, the remaining interval in the other set. The endpoint criterion of Lemma 8 can be fulfilled, but there is no pair of adjacent vertices. This shows that the requirement of connected overlap graphs is essential.

into two sets $\langle D', D'' \rangle$: numbers smaller than $L(I_1)$ and numbers larger than $R(I_1)$. Since $L(I_2) \in D'$ and $R(I_2) \in D''$, the partition is non-trivial.

Suppose that $\bigcup \mathcal{I}_1$ intersects with $D'$ or $D''$. We show that this would imply the existence of a pair of overlapping intervals from $\mathcal{I}_1$ and $\mathcal{I}_2$. Let $\sigma$ be a BFS-ordering for $\Omega(\mathcal{I}_1)$ with root the vertex corresponding to $I_1$. We iteratively compute $\bigcup \mathcal{I}_1$ along $\sigma$ starting with $I_1$, and since every added interval overlaps with a previously added interval, there is a first added interval $J'$ that intersects with either $D'$ or $D''$. Let the intersection be with $D'$; the other case is analogous. Let $J''$ be the interval from $\mathcal{I}_2$ with endpoint $\max D'$. By construction, there are two cases to be considered.

(1) Let $\max D' = L(J'')$. Then, $L(J') < L(J'') < R(J')$. Since $J'$ intersects either with $D'$ or with $D''$, $R(J') < R(J'')$, which would mean that $J'$ and $J''$ overlap. This is not possible, so that this case is not possible.

(2) Let $\max D' = R(J'')$. Then, $L(J'') < R(J'') < R(J')$. Since $L(J'') < L(J')$ would imply that $J'$ and $J''$ overlap, $L(J') < L(J'')$ must hold. In other words, $J'$ properly contains $J''$, more precisely, $J'' \subset J' \subset I_2$. The second inclusion holds due to assumption, since $J'$ and $I_2$ have a non-empty intersection. But Claim A shows that $J'' \subset J' \subset I_2$ implies the existence of overlapping intervals, which is not possible.

We conclude that $\bigcup \mathcal{I}_1$ does not intersect with neither $D'$ nor $D''$, and there are no numbers $a, c \in D_1$ and $b, d \in D_2$ such that $a < b < c < d$ or $b < a < d < c$, which concludes the proof. ■

A simple example shows that the connectivity prerequisites of Lemma 8 are essential. Consider the overlap model in Figure 2: three intervals, that are ordered by inclusion. Let $\mathcal{I}_1$ be the set containing the smallest and largest interval, and let $\mathcal{I}_2$ be the set containing the remaining middle-sized interval. Note that $\Omega(\mathcal{I}_2)$ is connected, and endpoints $a, b, c, d$ can be found to fulfill the criterion of Lemma 8. But, obviously, there is no pair of adjacent vertices in the defined overlap graph, since $\Omega(\mathcal{I}_1)$ is not connected

**Theorem 9** *Let $H = \Omega(\mathfrak{I})$ be an overlap graph, given by normalised overlap model $\mathfrak{I}$. A maximal spanning forest of $H$ can be generated in vertex-linear time.*

**Proof:** The construction is partitioned into two construction steps, an easy one and a hard one. The main challenge is obvious: determine overlapping only for a small number of pairs of intervals. Based on $\mathfrak{I}$, we define an auxiliary directed forest (that is different from the generated spanning forest), and this forest is reduced by path contraction. Every contraction operation defines edges in the generated spanning forest. At first, we present the algorithms and define a spanning forest of $H$, then we prove that the defined forest is a maximal spanning forest of $H$, and finally, we consider running-time questions.

Let $\mathfrak{I} = \{I_x\}_{x \in V(H)}$, and let $I_x = [L(x), R(x)]$ for every vertex $x$ of $H$. Let $\sigma$ denote the order of the vertices of $H$ such that $u \prec_\sigma v$ if and only if $L(u) < L(v)$. Since no pair of intervals has a common endpoint, $\sigma$ is well-defined. Based on $\sigma$, we define the auxiliary directed forest $F$ as follows:

(va) the vertex set of $F$ is $V(H)$

(ea) $(u, v)$ is an arc of $F$ if and only if $u \prec_\sigma v$ and $L(v) < R(u)$ and there is no vertex $w$ such that $u \prec_\sigma w \prec_\sigma v$ and $L(v) < R(w)$.

Note that the undirected version of $F$ is a forest: the vertex in a cycle rightmost with respect to $\sigma$ has two neighbours to the left, which is not possible according to definition (ea). Furthermore, without giving an explicit proof, the undirected version of $F$ is a maximal spanning forest of the interval graph defined by $\mathfrak{I}$.

We extend the definition of $F$ by attaching sets to its vertices: to vertex $x$, set $S_x$ is attached, that contains exactly $x$ as element. With this definition, $F$ defines a partition of $V(H)$. We apply a reduction algorithm to $F$ that contracts paths and coarsens the partition. Contracting a path is done by a sequence of arc contraction operations. An arc is contracted by replacing its start and end vertex by a single new vertex and making the new vertex adjacent with the remaining neighbours of the two replaced vertices, where orientations are kept. A path is contracted by iteratively contracting the arcs on the path. Note that the result of the path contraction does not depend on the order in which arcs are contracted. The algorithm processes vertices in the order defined by $\sigma$, which is compatible with the orientations of $F$. For a set $X$ of vertices of $H$, $\mathfrak{I}(X)$ denotes the set of intervals corresponding to the vertices in $X$, and $\bigcup \mathfrak{I}(X)$ denotes the union of the intervals in $\mathfrak{I}(X)$. For a connected component of the forest, the *root vertex* is the (unique) vertex that has no predecessor. The following algorithm is called the *easy construction algorithm*. It iteratively generates a sequence $F_0, \ldots, F_n$ of forests.

(a) Let $\sigma = \langle x_1, \ldots, x_n \rangle$

(b) let $F_0$ be $F$

(c) for every $i \in \{1, \ldots, n\}$ do:

  (d) let $T$ be the connected component of $F_{i-1}$ containing $x_i$

  (e) let $R$ be the root vertex of $T$

  (f) if $x_i \neq R$ then

    (g) let $P$ be the $R, x_i$-path in $T$

    (h) let $y$ be the vertex on $P$ that is closest to $R$ such that $R(\bigcup \mathfrak{I}(S_y)) < R(x_i)$

    (i) let $P'$ be the $y, x_i$-path in $T$

    (j) obtain $F_i$ from $F_{i-1}$ by contracting $P'$ to a single vertex and attaching the union of the sets attached to the vertices on $P'$ to the new vertex

    (k) let the new vertex have name $x_i$.

It is important to note that, for every $i \in \{1, \ldots, n\}$, path $P$ in iteration step $i$ contains only vertices from $\{x_1, \ldots, x_i\}$. It follows that vertex $x_i$ has not been vertex on a chosen root-vertex-path in an interation step $j < i$ and is still contained in $F_{i-1}$. In particular, when $x_i$ is processed its attached set contains exactly one vertex: $x_i$. We summarise important properties of $F_0, \ldots, F_n$ in the following claim.

**Claim A** *For every $i \in \{0, \ldots, n\}$, the following holds:*

*(1) $F_i$ is a directed forest*

*(2) $F_i$ defines a partition on $V(H)$*

*(3) for every vertex $x$ of $F_i$, the overlap graph defined by $\mathfrak{I}(S_x)$ is connected, where $S_x$ is the set attached to $x$ in $F_i$*

*(4) for every arc $(u, v)$ of $F_i$, $L(\bigcup \mathfrak{I}(S_u)) < L(\bigcup \mathfrak{I}(S_v)) < R(\bigcup \mathfrak{I}(S_u))$, where $S_u$ and $S_v$ are the sets attached to $u$ and $v$ in $F_i$, respectively*

*(5) for every arc $(u, v)$ of $F_i$ where $u, v \in \{x_1, \ldots, x_i\}$, $R(\bigcup \mathfrak{I}(S_v)) < R(\bigcup \mathfrak{I}(S_u))$, where $S_u$ and $S_v$ are the sets attached to $u$ and $v$ in $F_i$, respectively.*

We prove the claim by induction. By the definition of $F$, all statements hold for $i = 0$, i.e., for $F_0$. Let $i < n$, and let all statements be true for $F_i$. We prove that the statements hold for $F_{i+1}$. Statement (1) is clear: contraction of a path in a forest yields a forest. For statement (2), it suffices to observe that only the sets attached to the vertices on $P'$ are affected during iteration step $i + 1$, and the union of these sets is attached to the new vertex (algorithm line (j)), so that every vertex of $H$ is contained in exactly one of the sets attached to the vertices of $F_{i+1}$.

For the proofs of statements (3–5), let $P' = (y_0, \ldots, y_k)$ be the path defined in algorithm line (i), where $y_k = x_{i+1}$ and $y_0 = y$ for $y$ the vertex chosen in algorithm line (h). Due to the definition of ordering $\sigma$, $y_0, \ldots, y_k \in \{x_1, \ldots, x_{i+1}\}$. We first prove statement (5). Let $S_{x_{i+1}}$ denote the set attached to vertex $x_{i+1}$ in $F_{i+1}$; in $F_i$, $x_{i+1}$ is attached set $\{x_{i+1}\}$. By induction hypothesis (statement (5)), $R(\bigcup \mathfrak{I}(S_{y_j})) < R(x_{i+1})$ for every $j \in \{0, \ldots, k-1\}$ where $S_{y_j}$ is the set attached to $y_j$ in $F_i$. Hence, $R(x_{i+1}) = R(\bigcup \mathfrak{I}(S_{x_{i+1}}))$ (here, we already use statement (3), that $\bigcup \mathfrak{I}(S_{x_{i+1}})$ is a complete interval, which is proved in the next paragraph). Let $y$ have a predecessor $x$ in $F_i$; $x$ is predecessor of $x_{i+1}$ in $F_{i+1}$. By the choice of $y$, $R(\bigcup \mathfrak{I}(S_x)) > R(\bigcup \mathfrak{I}(S_{x_{i+1}}))$ where $S_x$ is the set attached to $x$ in $F_{i+1}$. Now, let $x \in \{x_1, \ldots, x_{i+1}\}$ be a vertex of $F_{i+1}$ that is a successor of $x_{i+1}$ in $F_{i+1}$. Then, there is $j < k$ such that $x$ is successor of $y_j$ in $F_i$, and $R(\bigcup \mathfrak{I}(S_x)) < R(\bigcup \mathfrak{I}(S_{y_j}))$ holds by induction hypothesis, where $S_x$ is the set attached to $x$ in $F_i$ and $F_{i+1}$ and $S_{y_j}$ is the set attached to $y_j$ in $F_i$. Hence, $R(\bigcup \mathfrak{I}(S_x)) < R(\bigcup \mathfrak{I}(S_{y_j})) < R(\bigcup \mathfrak{I}(S_{x_{i+1}}))$.

For proving statements (3–4), we partition the path contraction operation into a sequence of arc contraction operations and show by induction that statements (3–4) hold after each arc contraction operation. Since $F_{i+1} = F_i$ for $k = 0$, we assume $k \geq 1$. We define the sequence $F^{(k)}, \ldots, F^{(0)}$ of intermediate graphs: $F^{(k)} =_{\text{def}} F_i$ and $F^{(j)}$ emerges from $F^{(j+1)}$ by contracting $(y_j, x_{i+1})$ for every $j \in \{0, \ldots, k-1\}$ where the new vertex created in every contraction step is named $x_{i+1}$ (i.e., in every contraction step, the last arc of the path is contracted). By induction hypothesis, the statements hold for $F^{(k)}$. Let the statements be true for $F^{(j)}$ for $1 \leq j \leq k$. We prove the statements for $F^{(j-1)}$. Let $S^{(j)}_{j-1}$ denote the set attached to $y_{j-1}$ in $F^{(j)}$, and let $S^{(j)}_{k+1}$ denote the set attached to $x_{i+1}$ in $F^{(j)}$. By induction hypothesis, $L(\bigcup \mathfrak{I}(S^{(j)}_{j-1})) < L(\bigcup \mathfrak{I}(S^{(j)}_{i+1})) < R(\bigcup \mathfrak{I}(S^{(j)}_{j-1})) < R(\bigcup \mathfrak{I}(S^{(j)}_{i+1}))$. The last relation is due to the choice of $y$ and holds as shown in the last paragraph. Since $\mathfrak{I}(S^{(j)}_{j-1})$ and $\mathfrak{I}(S^{(j)}_{i+1})$ define connected overlap graphs, there is a pair of overlapping intervals in both sets due to Lemma 8, and $\mathfrak{I}(S^{(j)}_{j-1} \cup S^{(j)}_{i+1})$ defines a connected overlap graph.

For statement (4), we observe that $L(\bigcup \mathfrak{I}(S^{(j)}_{j-1} \cup S^{(j)}_{i+1})) = L(\bigcup \mathfrak{I}(S^{(j)}_{j-1}))$ due to induction hypothesis. So, if $x_{i+1}$ has a predecessor $x$ in $F_{j-1}$, $x$ is the predecessor of $y_{j-1}$ in $F^{(j)}$, and statement (4) holds for arc $(x, x_{i+1})$ in $F^{(j-1)}$. Now, let $x$ be a successor of $x_{i+1}$ in $F^{(j-1)}$. If $x$ is successor of $y_{j-1}$ in $F^{(j)}$, statement (4) for arc $(x_{i+1}, x)$ of $F^{(j-1)}$ follows from $L(\bigcup \mathfrak{I}(S^{(j)}_{j-1})) < \left\{L(\bigcup \mathfrak{I}(S^{(j)}_{x})), L(\bigcup \mathfrak{I}(S^{(j)}_{i+1}))\right\} < R(\bigcup \mathfrak{I}(S^{(j)}_{j-1})) < R(\bigcup \mathfrak{I}(S^{(j)}_{i+1}))$, where $S^{(j)}_x$ denotes the set attached to $x$ in $F^{(j)}$. If $x$ is successor $x_{i+1}$ in $F^{(j)}$, the claim immediately follows. For all other arcs, neither start not end vertex are involved in the arc contraction operation, so that we can conclude statement (4), and we conclude the proof of Claim A.

The hard construction step is based on a reduction algorithm, that works in a fashion similar to the easy construction algorithm. Iteratively, $F_n$ will be reduced by path contraction operations according to special rules. Instead of attaching vertex sets to the vertices and considering intervals, we will attach only endpoint sets and work on these endpoint sets. For every vertex $x$ of $F_n$, let $S_x$ be the set of vertices attached to $x$ in $F_n$. Let $D_x$ denote the endpoint set of the interval set corresponding to the vertices in $S_x$, as it was defined in the paragraph preceding Lemma 8. For every vertex $x$ of $F_n$, attach $D_x$ to $x$ instead of $S_x$ and obtain $F'$. The following algorithm is called the *hard construction algorithm.* A set of natural numbers is called a *complete interval*, if it contains every natural number between the set's minimum and maximum.

- (a) For every connected component $T$ of $F'$:
  - (b) let $R$ be the root vertex of $T$
  - (c) repeat as long as there are two vertices in $T$:
    - (d) let $x$ be a leaf of $T$
    - (e) if $D_x$ is a complete interval then
      - (f) delete $x$
      - (g) add $D_x$ to the number set attached to the predecessor of $x$
    - (h) if $D_x$ is not a complete interval then
      - (i) let $P$ be the $R, x$-path in $T$
      - (j) let $y$ be the vertex on $P$ that is closest to $R$ such that $D_y \cap [\min D_x, \max D_x] \neq \emptyset$
      - (k) let $P'$ be the $y, x$-path in $T$
      - (l) contract $P'$ to a single vertex and attach to it the union of the number sets attached to the vertices on $P'$.

Since $\mathfrak{I}$ is normalised and due to statement (2) of Claim A, $F'$ together with the number sets attached to its vertices defines a partition of the set $\{1, \dots, 2n\}$. After every treatment of a leaf vertex in the algorithm, the computed forest still defines a paritition of $\{1, \dots, 2n\}$. To see that the hard construction algorithm terminates, we show that the forest is reduced by at least one vertex in every iteration step. It suffices to show the following two properties: vertex $y$ in algorithm line (j) can be found and it is different from $x$. We consider an arbitrary iteration step of the loop in algorithm line (c). Let $x$ be the chosen leaf, and let $D_x \neq [\min D_x, \max D_x]$, i.e., there is a number $a$ such that $\min D_x < a < \max D_x$ and $a \notin D_x$. Since the current forest $F'$ defines a partition of $\{1, \dots, 2n\}$ there is a (unique) vertex $z$ whose currently attached number set contains $a$. By induction, it can be shown that statements (4–5) of Claim $A$ hold adapted to the new situation, where $L$ and $R$ must be replaced by min and max, so that $z$ lies on the $R, x$-path in $F'$, where $R$ denotes the root vertex of the connected component of $F'$ containing $x$. Since $a \in D_z$ and $a \notin D_x$, $x$ and $z$ are different. Hence, in every iteration step, $F'$ is reduced by at least one vertex.

Based on the two reduction algorithms, the easy construction algorithm and the hard construction algorithm, we define a spanning forest of $H$. Let the forest be denoted as $\mathfrak{F}$, and the vertex set of $\mathfrak{F}$ is $V(H)$. We begin with edges that are defined by the easy construction algorithm:

(e) in iteration step $i$, that defines $F_i$, $x_i$ is considered and path $P' = (y_0, \ldots, y_k)$ is contracted. Let $S_{y_0}, \ldots, S_{y_{k-1}}$ be the vertex sets attached to $y_0, \ldots, y_{k-1}$ in $F_{i-1}$, respectively. Let $z_0, \ldots, z_{k-1}$ be the vertices whose corresponding intervals define the right endpoints of $\bigcup \mathfrak{I}(S_{y_0}), \ldots, \bigcup \mathfrak{I}(S_{y_{k-1}})$, respectively. Make $x_i$ adjacent with $z_0, \ldots, z_k$.

It is not difficult to see that $x_i$ is connected only to vertices from different connected components of the generated forest in the current iteration step, so that the result after execution of the easy construction algorithm is a forest. Furthermore, every added edge is an edge of $H$: consider the edge $x_i z_j$. Due to the definition of $F$ and the reduction procedure, $z_j$ lies on the $R, x_i$-path in $F$ where $R$ is the root vertex of the connected component of $F$ containing $x_i$, and this means that $L(z_j) < L(x_i)$. Due to the easy construction algorithm and Claim A, $L(x_i) < R(z_j) < R(x_i)$. Hence, $I_{x_i}$ and $I_{z_j}$ overlap. It holds that edge definition (e) defines a subgraph of $H$ that is a spanning forest and whose connected components exactly correspond to the partition of $V(H)$ defined by the vertex sets attached to the vertices of $F_n$.

In a manner similar to the easy construction algorithm, the path contraction operation of the hard construction algorithm is used to define further edges of $\mathfrak{F}$. We consider the hard construction algorithm in some iteration step. Let $x$ be the selected leaf whose attached number set is not a complete interval. Furthermore, let $P' = (y_0, \ldots, y_k)$ be the found $y, x$-path in $T$. Due to Claim $A$ and the hard construction algorithm, $\min D_{y_0} < \cdots < \min D_{y_k} < \max D_{y_k} < \cdots < \max D_{y_0}$ and there is a number $a \in D_{y_0}$ such that $\min D_{y_k} < a < \max D_{y_k}$ and $a \notin D_{y_k}$. Hence, $\min D_{y_0} < \min D_{y_i} < a < \max D_{y_i}$ for every $i \in \{1, \ldots, k\}$. Due to Lemma 8, the vertex sets corresponding to $D_{y_0}$ and $D_{y_i}$ contain vertices that are adjacent in $H$ (the name *hard construction algorithm* is motivated by the fact that Lemma 8 must be used for showing the existence of overlapping intervals; in case of the easy construction algorithm, overlapping intervals are found *easi*ly). For every $i \in \{1, \ldots, k\}$, add such an edge to $\mathfrak{F}$. Since edges connect only connected components, after adding edges, we still have a forest. This concludes the definition of $\mathfrak{F}$.

By construction, $\mathfrak{F}$ is a forest and a subgraph of $H$. It remains to show that $\mathfrak{F}$ is a maximal spanning forest of $\mathfrak{F}$, i.e., that the connected components of $\mathfrak{F}$ correspond to the connected components of $H$ according to Lemma 1. Suppose this is not the case. Then, there are connected components of $\mathfrak{F}$ that are subgraphs of the same connected component of $H$. Hence, there are two connected components $C_1$ and $C_2$ of $\mathfrak{F}$ containing vertices that are adjacent in $H$, i.e., containing vertices whose corresponding intervals overlap. Let $u \in V(C_1)$ and $v \in V(C_2)$ be such vertices. We assume $L(u) < L(v)$. Since $L(u) < L(v) < R(u)$, $u$ and $v$ are contained in the same connected component of $F$ and $u$ lies on the path from the root vertex to $v$ in the connected component. Due to assumption and by statement (3) of Claim A, after application of the easy construction algorithm, vertices $u$ and $v$ are contained in different attached sets of $F_n$; let the corresponding vertices of $F_n$ be $u'$

and $v'$. Due to Claim A, it holds that $\bigcup \mathfrak{I}(S_{v'}) \subset \bigcup \mathfrak{I}(S_{u'})$. Let $D_{u'}$ and $D_{v'}$ be the endpoint sets attached to $u'$ and $v'$ in $F'$. Since $I_u$ and $I_v$ overlap, $D_{u'} \cap [\min D_{v'}, \max D_{v'}] \neq \emptyset$, and a vertex whose attached set contains $D_{u'}$ can be deleted not before its set also contains $D_{v'}$. This is only possible after a path contraction operation, which adds edges such that $u$ and $v$ cannot be contained in different connected components of $\mathfrak{F}$. Hence, there is a $u,v$-path in $\mathfrak{F}$, and $\mathfrak{F}$ is a maximal spanning forest of $H$.

It remains to show that a maximal spanning forest like $\mathfrak{F}$ can be computed in vertex-linear time. Since $\mathfrak{I}$ is normalised, vertex ordering $\sigma$ can be computed in vertex-linear time. Forest $F$ is computed using a stack: vertices are put onto the stack such that they are ordered by the right endpoints of the corresponding intervals and the interval corresponding to the top-of-stack element has the smallest right endpoint. This is achieved by deleting from the stack all vertices that do not fit into the order with the new vertex. An arc is added to the current forest $F$ from the current treated vertex to the topmost vertex on the stack whose corresponding interval has a non-empty intersection with the interval corresponding to the currently treated vertex. This algorithm takes vertex-linear time, since every vertex is treated once, put onto the stack once and popped from the stack at most once. In an array, right interval endpoints are stored.

We consider the running time of the easy construction algorithm. The connected components of $F$ can be determined in vertex-linear time by standard algorithms like breadth first search. It is not necessary to know the root vertices, since the paths can be determined by simply following the directions of the arcs in $F$. For the time taken by path contraction, the main idea is to not compute it iteratively arc by arc, since giving arcs new end vertices takes too much time, but to delete the path vertices at once and make the predecessors and successors of deleted path vertices predecessors or successors of the newly introduced vertex. Attached sets are implemented as chained lists, and the union of the attached sets is computed by changing pointers. All this takes amortised vertex-linear time. The only remaining part is to find vertex $y$. It is not possible to check all vertices on the vertex-root path; in particular, it has to be clear when the correct vertex $y$ has been found. But statement (5) of Claim A shows that the vertices on the path from the root are ordered by the corresponding right interval endpoints, so that $y$ can be found by traversing the path backwards to the root vertex and stopping at the first vertex that does not fit. All traversed vertices are deleted during the contraction, so that we conclude overall vertex-linear time for the easy construction algorithm. The edges of the maximal spanning forest can be listed straightforward as they were defined.

We consider the running time of the hard construction algorithm. It is clear that the running time mainly depends on algorithm line (j) and the way attached sets are implemented. The implementation question can be answered easily: we use just unordered lists for representing the attached sets. Then, computing the union is only updating some pointers. For finding vertex $y$, we describe an algorithm. At the beginning, we initialize two auxiliary arrays $A$ and $B$ of size $2n$ and an auxiliary array $C$ of size $n$:

(1) array $A$ has Boolean entries, "false" by default, and an entry is "true" when the vertex whose attached set contains this endpoint has already been considered as leaf

(2) array $B$ has pointer entries from the set $\{0, \ldots, 2n\}$, 0 by default, and the index and a non-zero entry represent an interval where all entries in $A$ are assumed to be "true"

(3) array $C$ has vertex entries, and the entry of index $a$ is the vertex whose attached set contains $a$.

Finally, for every vertex of $F'$, compute the distance to the root vertex of its connected component. This value is used for finding vertices closest to the root. Now, consider an arbitrary iteration step, and let $x$ be the chosen leaf. At first, the entries in array $A$ are set "true" for every index number in the set $D_x$ attached to $x$. In array $B$, the entry at index $\min D_x$ is set to $\max D_x$; if the pointer entry at index $\max D_x + 1$ is different from 0, use this value instead of $\max D_x$. By traversing array $A$ from $\min D_x$ to $\max D_x$, for every index whose entry is "false", determine the vertex whose attached set contains it. However, intervals in $A$ that are assumed to contain only "true" entries are not considered; here, array $B$ is always checked. If no entry "false" has been found, $D_x$ in the algorithm is a complete interval, and the algorithm procedes as described in algorithm lines (f–g). Otherwise, i.e., if there was an entry "false", the vertex closest to the root vertex, which is determined using the distance values, is vertex $y$ of algorithm line (j). The path contraction operation is executed, and the union of the attached sets is computed without taking the set attached to $x$. All this takes overall vertex-linear time, especially, since entries of array $A$ are checked only once.

We still have to show that vertex $y$ found by the algorithm in the last paragraph really has the properties required in algorithm line (j). Precisely, does $y$ lie on the path from the root vertex to $x$? By the definition of forest $F$ and due to statements (4–5) of Claim A and the definition of the path contraction operation, all numbers between $\min D_x$ and $\max D_x$ that are not contained in $D_x$ are contained only in sets attached to vertices on that path, which concludes the running-time analysis of the hard construction algorithm. The last paragraph also motivates the name "hard".

For the maximal spanning forest to be generated, the edges implied by the hard construction algorithm remain to be computed. With every vertex of $F'$, we associate the set of intervals from $\mathfrak{I}$ that correspond to the attached vertex sets of $F_n$, which means to the attached endpoint sets of $F'$. Let $\mathcal{I}$ be such a set of intervals. We order the intervals in $\mathcal{I}$ by the left endpoints and obtain the sequence $\langle I_1, \ldots, I_r \rangle$. Now, the maximal subsequence is selected starting with $I_1$ such that the intervals are also ordered by the right endpoints. It holds that every interval that is not contained in the final sequence is properly contained in a selected interval. These interval sequences are maintained and updated during the execution of the hard construction algorithm. In every iteration step, to every vertex of the remaining forest, such an interval sequence is associated. We describe the procedure executed in an iteration step. Let $y$ be the vertex found in algorithm line (j), and let $a$ be the endpoint selecting $y$. Due to the construction of forest $F$, $a$ is right endpoint of some interval $I$ corresponding to a vertex of the connected component associated with $y$ after execution of the easy construction algorithm. By the choice of $a$ and due to statements (4–5) of Claim A and Lemma 8, $I$ overlaps with intervals of every vertex between $y$ and $x$. For every such vertex (including $x$), the computed interval sequence is traversed from left to right. Since every possible interval is contained in the sequence or is properly contained in a sequence's interval, an overlapping interval can be found. Every such pair of overlapping intervals defines an edge in the spanning forest, and we obtain a new connected component. For this new connected component, a new interval sequence is computed. The single sequences are put together in a way described above by deleting subsequences. One only has to be careful not to pass all the sequences completely but going to the next sequence as early as possible. This can be done by storing the interval sequences in chained lists.

By the way, it does not have to be the case that $I$ appears in the computed sequence. The described algorithm has a vertex-linear-time implementation.

Now, we can conclude the running-time analysis of the generation algorithm for overlap graphs. All subroutines run in overall vertex-linear time, so that the total algorithm runs in vertex-linear time. ■

Again, the reader is encouraged to study the significance of Lemma 8 for the correctness of the hard construction algorithm.

**6 Multi-level trapezoid and overlap-trapezoid graphs** Trapezoid graphs are generalisations of permutation and interval graphs. They were introduced by Dagan, Golumbic, Pinter [2]. A *trapezoid* is a quadrilateral with two parallel sides. A *trapezoid graph* is the intersection graph of the following *trapezoid diagram*: given a set of trapezoids such that the parallel trapezoid sides all are on the same two parallel lines, vertices correspond to trapezoids, and two vertices are adjacent if and only if the corresponding trapezoids have a non-empty intersection. Such graphs are also called 1-*level trapezoid graphs*, or 1-*trapezoid graphs*, for short. Flotow generalised the definition of trapezoid graphs to multi-level trapezoid graphs [3]. Let $d \geq 0$ and $n \geq 1$. A *$d$-trapezoid over $n$* is a $(d+1)$-tuple $(l_1, \ldots, l_{d+1})$ where $l_i = [a_i, b_i]$ and $1 \leq a_i < b_i \leq 2n$ for all $i \in \{1, \ldots, d+1\}$. The numbers $a_1, b_1, \ldots, a_{d+1}, b_{d+1}$ are called the *trapezoid vertices*. The geometric representation of the corresponding polygon is achieved by connecting the trapezoid vertices. Let $T_1 = ([a'_1, b'_1], \ldots, [a'_{d+1}, b'_{d+1}])$ and $T_2 = ([a''_1, b''_1], \ldots, [a''_{d+1}, b''_{d+1}])$ be two $d$-trapezoids over $n$, where we assume $a'_1 \leq a''_1$. We say that $T_1$ and $T_2$ *have a non-empty intersection* if there is $i \in \{1, \ldots, d+1\}$ such that $a''_i \leq b'_i$. It is not hard to see that this definition of non-emtpy intersection exactly realises the fact of non-empty intersection of the corresponding polygons. A family of $n$ $d$-trapezoids over $n$ is called *normalised* if no pair of trapezoids has a common trapezoid vertex. Let $\mathfrak{T}$ be a normalised family of $n$ $d$-trapezoids over $n$. The *$d$-trapezoid graph defined by* $\mathfrak{T}$, denoted as $G(\mathfrak{T})$, is defined as follows:

(vt) $G(\mathfrak{T})$ has a vertex for every trapezoid in $\mathfrak{T}$

(et) $uv$ is an edge of $G(\mathfrak{T})$ if and only if the trapezoids corresponding to $u$ and $v$ have a non-empty intersection.

A graph $G$ on $n$ vertices is a *$d$-trapezoid graph* if there is a normalised family $\mathfrak{T}$ of $d$-trapezoids over $n$ such that $G = G(\mathfrak{T})$. We call $\mathfrak{T}$ a *$d$-trapezoid diagram* for $G$. It holds that the interval graphs are exactly the 0-trapezoid graphs [3], permutation graphs are 1-trapezoid graphs, and the trapezoid graphs are the 1-trapezoid graphs. We show in the following lemma that trapezoid graphs can be decomposed into three permutation and interval graphs, and this leads to easy generation algorithms for maximal spanning forests of trapezoid and $d$-trapezoid graphs.

**Lemma 10** *Let $G = G(\mathfrak{T})$ be a trapezoid graph, given by normalised trapezoid diagram $\mathfrak{T}$. In vertex-linear time, the pair of orderings for a permutation graph $P$ and normalised interval models for interval graphs $H_1$ and $H_2$ can be generated such that $G = P \cup H_1 \cup H_2$.*

**Proof:** Let $G$ have $n$ vertices, and let $\mathfrak{T} = \{T_x\}_{x \in V(G)}$. For every vertex $x$ of $G$, let $T_x = ([a'_x, b'_x], [a''_x, b''_x])$. The two interval models, denoted as $\mathfrak{I}_1$ and $\mathfrak{I}_2$, are defined as: $\mathfrak{I}_1 =_{\text{def}} \{[a'_x, b'_x]\}_{x \in V(G)}$ and $\mathfrak{I}_2 =_{\text{def}} \{[a''_x, b''_x]\}_{x \in V(G)}$. Since $\mathfrak{T}$ is normalised, $\mathfrak{I}_1$ and $\mathfrak{I}_2$ are also normalised. For the permutation graph, let $\sigma_1 = \langle x_1, \ldots, x_n \rangle$ be the vertex ordering for $G$ such that $b'_{x_1} < \cdots < b'_{x_n}$, and let $\sigma_2 = \langle y_1, \ldots, y_n \rangle$ be the vertex ordering for $G$ such that $b''_{y_1} < \cdots < b''_{y_n}$. Then, $(\sigma_1, \sigma_2)$ is the pair of orderings for the permutation graph. We first show that $P =_{\text{def}} G(\sigma_1, \sigma_2)$, $H_1 =_{\text{def}} G(\mathfrak{I}_1)$ and $H_2 =_{\text{def}} G(\mathfrak{I}_2)$ are subgraphs of $G$. Let $u$ and $v$ be vertices of $G$. It holds that $uv$ is an edge of $P$ if and only if either $u \prec_{\sigma_1} v$ or $u \prec_{\sigma_2} v$, so $uv \in E(P)$ implies that $T_u$ and $T_v$ have a non-empty intersection. For $H_1$ and $H_2$, the situation is obvious. Hence, $P \cup H_1 \cup H_2$ is a subgraph of $G$.

For the converse, let $u$ and $v$ be vertices of $G$, where $a'_u < a'_v$. Let $T_u$ and $T_v$ have a non-empty intersection. This is the case if and only if $a'_v < b'_u$ or $a''_v < b''_u$. If $a'_v < b'_u$, $a'_u < a'_v < b'_u$, and the intervals $[a'_u, b'_u]$ and $[a'_v, b'_v]$ have a non-empty intersection, i.e., $uv$ is an edge of $H_1$. Let $b'_u < a'_v < b'_v$. If $b''_u < b''_v$, then $uv$ is an edge of $H_2$; otherwise, i.e., if $b''_v < b''_u$, then $uv$ is an edge of $P$. Hence, $G = P \cup H_1 \cup H_2$. It is clear that $(\sigma_1, \sigma_2)$, $\mathfrak{I}_1$ and $\mathfrak{I}_2$ can be generated in vertex-linear time. ■

**Theorem 11** *Let $G = G(\mathfrak{T})$ be a trapezoid graph, given by normalised trapezoid diagram $\mathfrak{T}$. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** Applying Lemma 10, $G$ can be decomposed in vertex-linear time into a permutation graph $P = G(\sigma_1, \sigma_2)$ and two interval graphs $H_1 = G(\mathfrak{I}_1)$ and $H_2 = G(\mathfrak{I}_2)$. Maximal spanning forests $F_1, F_2, F_3$ for $P, H_1, H_2$ can be generated in vertex-linear time due to Theorems 4 and 6. And due to Theorem 3, a maximal spanning forest of $G$ is generated from $F_1$, $F_2$ and $F_3$. ■

**Corollary 12** *Let $d \geq 1$ be fixed. Let $G = G(\mathfrak{T})$ be a $d$-trapezoid graph, given by normalised $d$-trapezoid diagram $\mathfrak{T}$. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** Let $\mathfrak{T} = \{T_x\}_{x \in V(G)}$, and let $T_x = ([a_x^1, b_x^1], \ldots, [a_x^{d+1}, b_x^{d+1}])$ for every vertex $x$ of $G$. We define $d$ trapezoid graphs defined by the trapezoid models $\mathfrak{T}_1, \ldots, \mathfrak{T}_d$: for every $i \in \{1, \ldots, d\}$, let $\mathfrak{T}_i =_{\text{def}} \{([a_x^i, b_x^i], [a_x^{i+1}, b_x^{i+1}])\}_{x \in V(G)}$. It directed follows from the definition, that two vertices $u$ and $v$ of $G$ are adjacent if and only if there is $i \in \{1, \ldots, d\}$ such that $u$ and $v$ are adjacent in $G(\mathfrak{T}_i)$. Hence, $G = G(\mathfrak{T}_1) \cup \cdots \cup G(\mathfrak{T}_d)$. Applying Theorems 11 and 3, we obtain a vertex-linear-time algorithm for generating maximal spanning forests for $d$-trapezoid graphs in vertex-linear time. This time-bound is only achieved, since $d$ is a constant and does not depend on the input. ■

In the fashion, trapezoid graphs generalise interval graphs, we define overlap-trapezoid graphs as generalisations of overlap graphs. We begin with the definition of multi-level overlap-trapezoid graphs. Let $d \geq 0$ and $n \geq 1$, and let $T_1 = ([a'_1, b'_1], \ldots, [a'_{d+1}, b'_{d+1}])$ and $T_2 = ([a''_1, b''_1], \ldots, [a''_{d+1}, b''_{d+1}])$ be two $d$-trapezoids over $n$. Informally (and geometrically) spoken, $T_1$ and $T_2$ overlap if the corresponding polygons have a non-empty intersection and this intersection is properly contained in both polygons. We formally define *overlapping* for $d$-trapezoids by reduction to *overlapping* of trapezoids. We say that two trapezoids $([c'_1, d'_1], [c'_2, d'_2])$ and $([c''_1, d''_1], [c''_2, d''_2])$, where $c'_1 \leq c''_1$, *overlap* if neither $c'_1 \leq c''_1 \leq d''_1 \leq d'_1$ and $c'_2 \leq c''_2 \leq d''_2 \leq d'_2$ nor $d'_1 < c''_1$ and $d'_2 < c''_2$. Then, we say that $d$-trapezoids $T_1$ and $T_2$ *overlap* if there exists $i \in \{1, \ldots, d\}$ such that $([a'_i, b'_i], [a'_{i+1}, b'_{i+1}])$ and $([a''_i, b''_i], [a''_{i+1}, b''_{i+1}])$ overlap. Let $\mathfrak{T}$ be a normalised family of $n$ $d$-trapezoids over $n$. The *$d$-overlap-trapezoid graph defined by* $\mathfrak{T}$, denoted as $\Omega(\mathfrak{T})$, is defined as follows:

(vd) $\Omega(\mathfrak{T})$ has a vertex for every trapezoid in $\mathfrak{T}$

(ed) $uv$ is an edge of $G(\mathfrak{T})$ if and only if the trapezoids corresponding to $u$ and $v$ overlap.

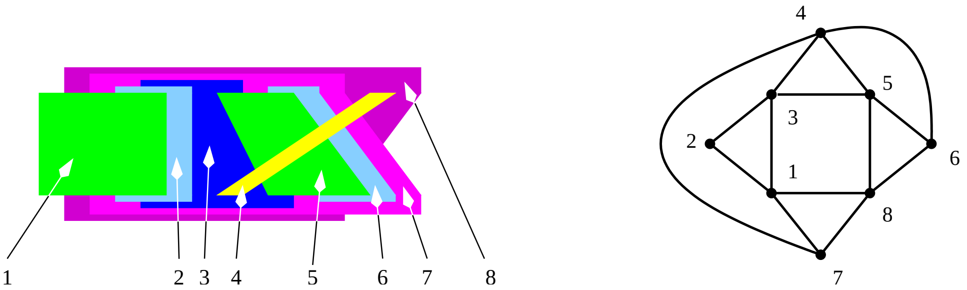


**Figure 3** The figure shows an overlap-trapezoid diagram and the defined overlap-trapezoid graph. In the model picture, the arrows point from the vertex names to the corresponding trapezoids. Note that the trapezoids are enlarged in vertical direction by additional rectangles to make the picture better understandable.

A graph $G$ on $n$ vertices is a *$d$-overlap-trapezoid graph* if there is a normalised family $\mathfrak{T}$ of $n$ $d$-trapezoids over $n$ such that $G = \Omega(\mathfrak{T})$. We call $\mathfrak{T}$ a *$d$-overlap-trapezoid diagram* for $G$. It is clear that the overlap graphs are exactly the 0-overlap-trapezoid graphs; in particular, permutation graphs are 0-overlap-trapezoid graphs, since permutation graphs are circle graphs (which is equivalent to being overlap graph) [5]. We call 1-overlap-trapezoid graphs *overlap-trapezoid graphs* and 1-overlap-trapezoid diagrams *overlap-trapezoid diagrams*.

A sample overlap-trapezoid diagram and the defined overlap-trapezoid graph are depicted in Figure 3. Note that the graph is not an overlap graph, which can be verified by exhaustive search for an appropriate overlap or circle model. Since cycles have almost unique representations as circle models, the cycle $(1, 3, 5, 8)$ already determines the structure of a model. As a remark, the depicted graph without edge $\{4, 7\}$ is a circle (an overlap) graph.

Similar to trapezoid graphs, we can decompose overlap-trapezoid graphs into permutation and interval graphs. However, when we needed only three decomposition graphs in case of trapezoid graphs, we need four decomposition graphs in case of overlap-trapezoid graphs.

**Lemma 13** *Let $G = G(\mathfrak{T})$ be an overlap-trapezoid graph, given by normalised overlap-trapezoid diagram $\mathfrak{T}$. In vertex-linear time, pairs of orderings for permutation graphs $P_1$ and $P_2$ and overlap models for overlap graphs $H_1$ and $H_2$ can be generated such that $G = P_1 \cup P_2 \cup H_1 \cup H_2$.*

**Proof:** We reconsider the construction and proof of Lemma 10. Let $\mathfrak{T} = \{T_x\}_{x \in V(G)}$, and let $T_x = ([a'_x, b'_x], [a''_x, b''_x])$ for every vertex $x$ of $G$. Let $\mathfrak{I}_1 =_{\mathrm{def}} \{[a'_x, b'_x]\}_{x \in V(G)}$ and $\mathfrak{I}_2 =_{\mathrm{def}} \{[a''_x, b''_x]\}_{x \in V(G)}$, and let $H_1 =_{\mathrm{def}} G(\mathfrak{I}_1)$ and $H_2 =_{\mathrm{def}} G(\mathfrak{I}_2)$, as in the proof of Lemma 10. For the permutation graphs, we define the vertex orderings $\sigma'_1$, $\sigma'_2$, $\sigma''_1$, $\sigma''_2$ for $G$ according to increasing $a'$-, $a''$-, $b'$-, $b''$-values, respectively. Let $P_1 =_{\mathrm{def}} G(\sigma'_1, \sigma'_2)$ and $P_2 =_{\mathrm{def}} G(\sigma''_1, \sigma''_2)$. Similar to the proof of Lemma 10, it is shown that $P_1 \cup P_2 \cup H_1 \cup H_2$ is a subgraph of $G$. So, let $uv$ be an edge of $G$, $u \prec_{\sigma'_1} v$, i.e., $T_u$ and $T_v$ overlap. If $[a'_u, b'_u]$ and $[a'_v, b'_v]$ overlap or if $[a''_u, b''_u]$ and $[a''_v, b''_v]$ overlap, $uv$ is an edge of $H_1 \cup H_2$. Otherwise, if $b'_u < a'_v$ or $b''_u < a''_v$, $uv$ is an edge of $P_2$, and if $b'_v < b'_u$ and $b''_v < a''_u$, $uv$ is an edge of $P_1$. Hence, $G = P_1 \cup P_2 \cup H_1 \cup H_2$. ■

Vertex-linear-time maximal spanning forest algorithms for overlap-trapezoid and $d$-overlap-trapezoid graphs follow similar to trapezoid and $d$-trapezoid graphs.

**Theorem 14** *Let $G = G(\mathfrak{T})$ be an overlap-trapezoid graph, given by normalised overlap-trapezoid diagram $\mathfrak{T}$. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** Similar to the proof of Theorem 11, the algorithms of Lemma 13 and Theorems 4, 9 and 3 are applied. ∎

**Corollary 15** *Let $d \geq 1$ be fixed. Let $G = \Omega(\mathfrak{T})$ be a $d$-overlap-trapezoid graph, given by normalised $d$-overlap-trapezoid diagram $\mathfrak{T}$. A maximal spanning forest of $G$ can be generated in vertex-linear time.*

**Proof:** The algorithm is based on the decomposition of $d$-overlap-trapezoid graphs into overlap-trapezoid graphs in a fashion similar to $d$-trapezoid graphs (see the proof of Corollary 12) and the application of the algorithms of Theorems 14 and 3. ∎

**7 Final remarks** We have presented fast algorithms for generating maximal spanning forests for geometric graph classes like permutation graphs, interval graphs and trapezoid graphs on the one hand and overlap graphs and overlap-trapezoid graphs on the other hand. The running times of the algorithms are all proportional to the numbers of vertices of the input graphs, where we required graphs to be given by appropriate geometric models. The algorithms for the most general graph classes that we considered applied procedures for smaller graph classes as subroutines. For instance, the trapezoid graphs algorithm applied the permutation graphs and interval graphs algorithm as subroutine. This approach became applicable since trapezoid graphs could be decomposed into at most three permutation and interval graphs such that every edge of the given graph is contained in at least one decomposition graph.

Decomposition methods for graphs are always interesting. The hope is that hard problems on a given graph can be solved more efficiently on the decomposition graphs, and the partial solutions can be used for efficiently generate a solution for the given graph ("divide-and-conquer"). The general decomposition that we used here is simple, since it is only required that every edge appears in a decomposition graph. It is clear that it does not provide a suitable approach for solving most problems. The main reason is the fact that structural properties are lost when arbitrarily decomposing a graph in this way. Nevertheless, it would be interesting to find further problems for which this decomposition also helps designing nice and efficient algorithms.

In the last section, we defined the class of $d$-overlap-trapezoid graphs, where $d$ is a fixed number. It seems that these graphs have not yet been considered and studied before. Natural questions ask for the complexity of fundamental algorithmic problems on these graphs. Further interesting questions consider graph class inclusion aspects. One striking question relates interval graphs to $d$-overlap-trapezoid graphs and asks whether, for every interval graph $G$, there is a number $d$ such that $G$ is a $d$-overlap-trapezoid graph. Determine a small graph class containing the $d$-overlap-trapezoid graphs for all $d \geq 0$. In case of $d$-trapezoid graphs, this upper bound is the class of cocomparability graphs [3].

## Bibliography


[1] D.G. Corneil, St. Olariu, L. Stewart, *Asteroidal triple-free graphs*, SIAM Journal on Discrete Mathematics 10, pp. 399–430, 1997.

[2] I. Dagan, M.C. Golumbic, R.Y. Pinter, *Trapezoid graphs and their coloring*, Discrete Applied Mathematics 21, pp. 35–46, 1988.

[3] C. Flotow, *On powers of $m$-trapezoid graphs*, Discrete Applied Mathematics 63, pp. 187–192, 1995.

[4] T. Gallai, *Transitiv orientierbare Graphen*, Acta Mathematica Academiae Scientiarum Hungaricae 18, pp. 25–66, 1967.

[5] F. Gavril, *Algorithms for a maximum clique and a minimum independent set of a circle graph*, Networks 3, pp. 261–273, 1973.

[6] M.Ch. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[7] T. Kloks, D. Kratsch, C.K. Wong, *Minimum Fill-in on Circle and Circular-Arc Graphs*, Journal of Algorithms 28, pp. 272–289, 1998.

[8] C.G. Lekkerkerker, J.Ch. Boland, *Representation of finite graphs by a set of intervals on the real line*, Fundamenta Mathematicae 51, pp. 45–64, 1962.