University of Alberta

Results on Set Representations of Graphs

by

Jessica Anne Enright

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Spring 2012
Edmonton, Alberta

Abstract

A set representation of a graph is an assignment of sets to vertices such that two vertices are adjacent if and only if their assigned sets have some specified relationship. We give several results related to set representations of graphs.

We show that recognising the overlap and intersection graphs of subtrees in some types of trees is NP-hard. The subtree overlap graphs (SOGs) generalise many other graph classes with set representation characterisations. The complexity of recognising SOGs is open. The complexities of recognising many subclasses of SOGs are known. We consider several subclasses of SOGs by restricting the underlying tree. For a fixed integer $k \geq 3$, we consider:

- the overlap graphs of subtrees in a tree with k leaves,
- the overlap graphs of subtrees in trees that can be derived from a given input tree by subdivision and have at least 3 leaves,
- the overlap and intersection graphs of paths in a tree with maximum degree k,

We show that the recognition problems of these classes are NP-complete. We give characterisations of several subclasses of overlap graphs of subtrees in a tree in terms of filament representations.

List colouring with a fixed colour bound of at least three is NP-complete, even on planar bipartite graphs. We give a polynomial-time algorithm for solving list colouring with a fixed colour bound on permutation and interval graphs, two classes with intersection representations.

Finally, we describe a class of impartial combinatorial games on graphs using set representations. In these games, the players antagonistically build a set representation of a graph. We give hardness results for determining the winner of a position of these types of games in general, and give polynomial-time algorithms to solve special cases of these games on trees.

Acknowledgements

This thesis could not have been written without the help of my supervisor, Lorna Stewart. My supervisory committee has also been helpful throughout the process. Several of the papers here have been substantially improved thanks to comments from anonymous reviewers.

The entire department has been remarkably welcoming and friendly. I owe a lot to past and present administrative staff, especially Edith Drummond and Karen Berg. I can't count the number of times they made sure that I got paid, had an office, was properly registered, submitted forms on time, or saved me from my own procrastination in some other way.

Friends both here and abroad have made my time in grad school fun and productive. I certainly would not have come out of this as sane as I am without great friends. Claps for you all!

My family has been very supportive during what must have seemed like never-ending student-hood. My parents, aunts, uncles, siblings, cousins have all been very encouraging and proud, especially my grandmother, Maureen Enright. I'm sorry that I didn't quite finish this thesis in time to show her.

The production of this thesis was powered mainly by coffee and CBC Radio 3.

Table of Contents

List of Figures

Chapter 1

Introduction

A set representation of a graph is an assignment of sets to vertices such that the adjacency of the vertices corresponds to some relationship between the sets.

More precisely, let $\mathcal{S}$ be a family of sets and $G = (V, E)$ be a graph.

G is an *intersection graph* of $\mathcal{S}$ if there is a bijection f: $V \to \mathcal{S}$ such that $(v_i, v_j) \in E$ if and only if $f(v_i)$ and $f(v_j)$ intersect.

G is a *containment graph* of $\mathcal{S}$ if there is a bijection $f : V \to \mathcal{S}$ such that $(v_i, v_j) \in E$ if and only if $f(v_i) \subseteq f(v_j)$ or $f(v_j) \subseteq f(v_i)$.

Two sets *overlap* if they intersect but neither is contained in the other. G is an *overlap graph* of $\mathcal{S}$ if there is a bijection $f : V \to \mathcal{S}$ such that $(v_i, v_j) \in E$ if and only if $f(v_i)$ overlaps $f(v_j)$.

G is a *disjointness graph* of $\mathcal{S}$ if there is a bijection $f : V \to \mathcal{S}$ such that $(v_i, v_j) \in E$ if and only if $f(v_i)$ is disjoint from $f(v_j)$. See Figure 1.1.

If G is an *intersection* (containment, overlap, disjointness) *graph* of a family of sets, we say those sets are an intersection (containment, overlap, disjointness) representation of G.

Every graph has an intersection [59], overlap and disjointness [59] representation. Not every graph has a containment representation [29].

When we restrict the type of sets we use to represent a graph, we restrict the type of graph and therefore define a graph class. The first intersection graphs to be studied were the *interval graphs*, the intersection graphs of intervals on a line [27]. Since then, researchers have studied many classes of intersection, overlap and containment graphs. Given a set representation, we can solve many otherwise hard problems on intersection and overlap classes, including many problems on the intersection graphs of intervals on a line and subtrees in a tree [51], and maximum weighted clique and independent set on subtree filament graphs and interval filament graphs [23]. That intersection and overlap representations can be used to find solutions to hard problems has been a significant motivation in the study of set representation classes of graphs.

This thesis deals with graph classes with set representation characterisations. We give a characterisation in terms of several types of set representations for subclasses of overlap graphs of subtrees in a tree. We show that it is hard to recognise several other subclasses of overlap graphs of sub-

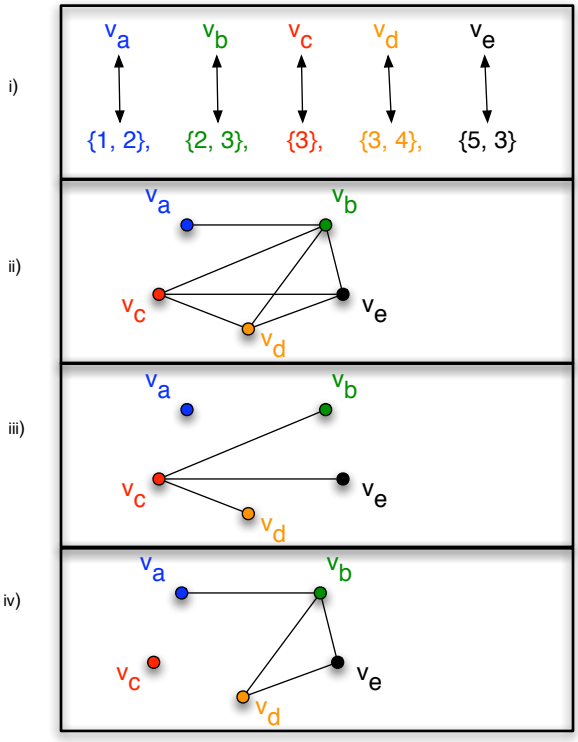

Figure 1.1: A family of sets (i), and its intersection (ii), containment (iii) and overlap (iv) graphs. In (i) we show the bijection between vertices and sets.

trees in a tree. Using a vertex ordering arising from their intersection representations, we provide a polynomial-time algorithm for list colouring with a fixed total number of colours, a problem that is hard in general on two classes of intersection graphs. We describe a family of games using set representations of graphs. We show that in general it is hard to determine who will win a position of these games, but give polynomial-time algorithms to solve cases of these games with particular intersection representation structure.

We have organised the remainder of this chapter in the same order in which our results are presented. In Section 1.1, we give general preliminaries on graphs. In Section 1.2 we describe related work on set representations and classes defined by set representations. In Section 1.3 we discuss related work on list colouring. In Section 1.4 we introduce combinatorial games.

1.1 Preliminaries

An *undirected graph* is an ordered pair $G = (V, E)$ where V is the vertex set of G, and E is a set of subsets of V of size two. We call the members of E *edges* of G, and by convention denote them using parentheses rather than braces. For example, if e is an edge in E consisting of the set of size two containing $u, v \in V$, then we write $e = (u, v)$, and $(u, v) \in E$. In future, if we use the term *graph* we mean an undirected graph, unless we state otherwise.

A *directed graph* is an ordered pair $G = (V, E)$ where V is the vertex set of G, and E is a set of ordered pairs of elements of V. We call the members of E *arcs* of G, and by convention denote them with parenthesis rather than braces, indicating the order of the pair with an arrow. For example, if e is an edge in E consisting of pair $u, v \in V$, with u being the first member of the pair, then we write $e = (u \to v)$, and $(u \to v) \in E$.

When it is clear which graph we are talking about, we denote by n the size of the vertex set, and m the size of the edge set.

Let $G = (V, E)$ be a graph. If for two vertices $v_i, v_j \in V$, $(v_i, v_j) \in E$, we say v_i and v_j are *adjacent*. Otherwise they are *nonadjacent*. An *isolated vertex* is adjacent to no other vertices. A *universal vertex* is adjacent to all other vertices

Vertices v_i and v_j are the *endpoints* of edge $e = (v_i, v_j)$, and e is *incident* at both v_i and v_j.

The graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there exists a bijection $f : V_1 \to V_2$ such that for two vertices $v_i, v_j \in V_1$, $(v_i, v_j) \in E_1$ if and only if $(f(v_i), f(v_j)) \in E_2$.

The *neighbourhood* of a vertex v, denoted $N(v)$, is the set of vertices adjacent to v.

Let $G = (V, E)$ be a graph and $V' \subset V$ a vertex subset. We say that V' *induces* the subgraph $G[V'] = (V', E')$ such that $E' = \{(v_i, v_j)$ where $v_i, v_j \in V'$ and $(v_i, v_j) \in E\}$. A subgraph H of G is an *induced subgraph* of G if there exists a vertex set V_1 such that H is isomorphic to $G[V_1]$.

A *nonedge* of G is a pair of vertices $(v_i, v_j) \subset V$ such that $(v_i, v_j) \notin E$. The *complement* of $G = (V, E)$, denoted $\overline{G} = (V, \overline{E})$, is a graph with the same vertex set as V, and an edge set composed of the nonedges of G.

An *orientation* of a graph $G = (V, E)$ is a directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$ such that if $(v_i, v_j) \in E$ then exactly one of $(v_i \rightarrow v_j)$ or $(v_j \rightarrow v_i)$ is in $\overrightarrow{E}$, and the endpoints of every edge in E' are adjacent in G.

An orientation of a graph $G = (V, E)$ is *transitive* if for every three vertices v_i, v_j, v_k such that $(v_i \rightarrow v_j) \in \overrightarrow{E}$ and $(v_j \rightarrow v_k) \in \overrightarrow{E}$ it holds that $(v_i \rightarrow v_k) \in \overrightarrow{E}$. The *in-degree* of vertex $v_i \in V$ is the number of vertices $v_j \in V$ such that $(v_j \rightarrow v_i) \in \overrightarrow{E}$. The *out-degree* of v_i is the number of vertices $v_j \in V$ such that $(v_i \rightarrow v_j) \in \overrightarrow{E}$. A vertex is a *source* if its in-degree is zero. A vertex is a *sink* if its out-degree is zero.

A graph $G = (V, E)$ is *complete* if every pair of vertices are adjacent. K_n is the complete graph on n vertices. For a graph $G = (V, E)$, a vertex subset $V_c \subseteq V$ is a *clique* if $G[V_c]$ is a complete graph. V_c is a *maximal clique* of G if it is not a proper subset of any other clique of G, and is a *maximum clique* of G if there is no larger clique of G. A vertex is *simplicial* if its neighbourhood is a clique.

An *independent set* of a graph is a set of pairwise nonadjacent vertices. An independent set of graph G is maximal if it is not a proper subset of any other independent set of G and is maximum if there is no larger independent set of G. The maximum independent set of a graph G is denoted $\alpha(G)$

A *path* is a graph with distinct vertices $v_1, v_2....v_k$ such that for every $1 \leq i < k$, v_i is adjacent to v_{i+1}.

A *cycle* is a path in which the first and last vertices of the path are also adjacent.

The *length* of a cycle or path is the number of edges in that cycle or path.

A *chord* is an edge between two vertices v_i, v_j in a path such that $i \neq j + 1$ and $j \neq i + 1$, or an edge between two vertices v_i, v_j in a cycle such that $i \neq j + 1$ and $j \neq i + 1$ and i and j are not the first and last vertices.

P_n is the chordless path on n vertices. C_n is the chordless cycle of length n. Unless stated otherwise, when we refer to a path or cycle, we intend it to be chordless.

A *clique cover* of graph $G = (V, E)$ is a set of cliques of G such that the union of the vertices of those cliques is equal to V. The *clique cover number* of G is the size of the smallest clique cover of G. An edge clique cover of a graph $G = (V, E)$ is a set of cliques of G such that every edge in E is in a subgraph induced by at least one of the cliques.

A *colouring* of G is a partition of V into classes $C_1...C_k$ such that no two adjacent vertices are in the same class. The size of a colouring is the number of colour classes in it. The *chromatic number* of G is the size of the smallest colouring of G. A graph G is *perfect* if for every induced subgraph H of G the chromatic number of H is the same as the size of the largest clique of H.

$G = (V, E)$ is *connected* if for every two vertices $v_i, v_j \in V$ there is a path in G from v_i to v_j. If G is not connected, then we say that G is disconnected.

A *tree* is a connected graph with no cycles. A *leaf* is a vertex of a tree with degree one. A *star*

is a tree with only one non-leaf vertex. A *caterpillar* is a tree for which removing all leaves would leave a chordless path.

For a class of graphs $\mathcal{G}$, the *recognition problem* is: for a graph G, is G a member of $\mathcal{G}$?

Let A and B be two sets. A and B *intersect*, denoted $A \cap B$, if they share at least one element. A and B are *disjoint*, denoted $A|B$ if they do not intersect. A is *contained* in B (or B contains A), denoted $A \subseteq B$, if every element in A is in B. A is *strictly contained* in B (or B is strictly contains A), denoted $A \subset B$, if A is contained in B and there is at least one element of B not in A. A *overlaps* B, denoted $A \between B$, if $A \cap B$ but neither $A \subset B$ nor $B \subset A$.

Let A, B, A', B' be four sets. We say that A, B and A', B' are *similarly related*, denoted $A, B \sim A'B'$, if $A|B$ if and only if $A'|B'$, $A \between B$ if and only if $A' \between B'$, $A \subseteq B$ if and only if $A' \subseteq B'$ and $B \subseteq A$ if and only if $B' \subseteq A'$.

Let $\mathcal{S}$ be a family of sets. $\cup\mathcal{S}$ is the union of all members of $\mathcal{S}$. $\cap\mathcal{S}$ is the intersection of all members of $\mathcal{S}$. $\mathcal{S}$ is *Helly* (or has the *Helly property*) if, for every subfamily $\mathcal{S}'$ of $\mathcal{S}$ such that all members of $\mathcal{S}'$ pairwise intersect, $\cap\mathcal{S}' \neq \emptyset$.

A decision problem is in complexity class P if it can be solved in time polynomial in the size of the input by a deterministic Turing machine. It is in class NP if, for a yes instance, there is a proof that the answer is "yes" that can be verified in polynomial time by a deterministic Turing machine. It is in class PSPACE if it can be solved in polynomial space by a deterministic Turing machine. For definitions and discussion of complexity classes, completeness, and Turing machines, we refer the reader to Sipser [56].

1.2 Related Work on Set Representations

1.2.1 Size of Representations

Let family of sets $\mathcal{S}$ be a set representation of graph $G = (V, E)$. We call $|\cup\mathcal{S}|$ the *size* of the representation.

Erdös et al. [16] showed that every graph on n vertices has an intersection representation of size at most $\lfloor n^2/4 \rfloor$, and that there exist graphs for which this many elements are required. They used the relationship between edge clique covers and intersection representations in their proof. The size of a minimum edge clique cover of a graph G is equal to the size of a minimum intersection representation of G [16].

Given an edge clique cover $\mathcal{U}$ for $G = (V, E)$, one can produce an intersection representation as follows: for each $v_i \in V$, let $s_i = \{u \text{ where } u \in \mathcal{U} \text{ and } v_i \text{ is a vertex of } u\}$.

Given an intersection representation $\mathcal{S}$ for graph $G = (V, E)$ a cover can be derived as follows: for each element w_i in $\cup\mathcal{S}$, define a clique k_i where a vertex $v_j \in V$ is in k_i if and only if $w_i \in s_j$.

Erdös et al. [16] showed that the minimum edge clique cover size of a complete bipartite graph in which the two sides of the bipartition differ in size by at most one vertex is $\lfloor n^2/4 \rfloor$, and that

therefore the smallest intersection representation of such a graph has that many elements.

Poljak et al. [49] used a similar edge clique cover approach to investigate the complexity of finding the minimum size of set representations. They defined a $k-$set representation as a set representation in which the largest set assigned to a vertex is of size k, and a distinct $k-$set representation as a $k-$set representation with no set assigned to more than one vertex. A distinct $k-$cover is a $k-$cover in which, for every pair of vertices, there is a clique in the cover that contains one of them but not the other.

Poljak et al. [49] showed that the following problems are NP-complete for any fixed integer $k \geq 3$:

- Does there exist a $k-$cover of G?
- Does there exist a distinct $k-$cover of G
- Does G have a distinct 3-set representation?
- Does G have a 4-set representation?
- Let h be the smallest integer such that there exists a $h-$cover of G, and i the smallest integer such that there exists a distinct $i-$cover of G. Does $h = i$?

Poljak et al. also showed that it is NP-complete, given integer $k \geq 3$, for a given graph G and fixed integer k to determine whether there is a simple intersection representation $\mathcal{S}$ of G such that $|\cup \mathcal{S}| \leq k$. A *simple intersection representation* is an intersection representation in which intersect in at most one element, and no two sets are equal.

This dovetails nicely with the result due to Kou, Stockmeyer and Wong [40] that, given a natural number k, it is NP-complete for a given graph G to determine if there is an intersection representation $\mathcal{S}$ of G such that $|\cup \mathcal{S}| \leq k$.

Rosgen [52] found the sizes of minimum intersection representations for several classes of graphs, including the complements of graphs composed of disjoint edges. Rosgen also defined the *intersection extension problem*: given a graph $G = (V, E)$, an intersection representation $\mathcal{S}$ of G, and a vertex set $A \subseteq V$, is there is a set $s \subseteq \cup\mathcal{S}$ such that s intersects exactly the sets in $\mathcal{S}$ corresponding to vertices in A?

The motivation for this problem is the question: given a graph and a representation, can one add a new vertex with arbitrary neighborhood to the graph and still represent it by just adding a new set for the new vertex without changing any of the rest of the representation, or adding new elements to the universe of the representation? Rosgen showed that the intersection extension problem can be solved in polynomial time.

Overlap Graphs

There has been less study of these general questions on overlap graphs. Rosgen [52] studied optimally sized overlap representations. He called the size of an optimally sized overlap representation the *overlap number* of a graph. He proved the size of a minimum overlap representation for

the paths, cycles, caterpillars, cliques, and complete $k-$partite graphs. He also described the *overlap extension problem*, similar to the intersection extension problem. Rosgen showed that while the intersection extension problem can be solved in polynomial time, the overlap extension problem is NP-complete, even when the new vertex added is universal.

Cranston et al. [11] proved that the overlap number of a tree is the number of vertices in a maximum subtree in which every neighbour of a leaf has degree two, and gave a linear-time algorithm for finding such a representation.

They also gave a number of other bounds on overlap number, showing tightness in some cases. They showed that the overlap number of a graph with minimum degree two that is not a complete graph of size three is at most the number of edges minus one. They showed that this is exactly the overlap number for connected, triangle-free graphs with no star-cutsets. They showed that the overlap number of a planar graph on $n \geq 5$ vertices is at most $2n - 5$, and that this is exactly the overlap number if every face has four vertices and there is no star-cutset. Finally, they showed that the overlap number for a general graph on $n \geq 14$ vertices is at most $n^2/4 - n/2 - 1$, and that this is exactly the overlap number if the graph can be formed by deleting a perfect matching from a complete bipartite graph with $n/2$ vertices in each part of the bipartition.

Containment Graphs

Comparability graphs are undirected graphs that admit a transitive orientation of their edges. These graphs are well-studied, and have a number of characterizations.

Every containment graph is a comparability graph - see [29] - the proof of this is based on the transitivity of containment: given three sets a, b, c if $a \subseteq b$ and $b \subseteq c$, then $a \subseteq c$.

Every comparability graph is the containment graph of substars of a star [28]. A proof, by Golumbic and Scheinerman, describes how to produce from a comparability graph $G = (V, E)$ a family of substars of a star S such that G is the containment graph of $\mathcal{S}$.

Then comparability graphs are exactly containment graphs, and are also exactly containment graphs of substars of a star.

An important early paper on comparability graphs by Gallai [20] is translated in [50]. This paper gives a forbidden induced subgraph characterization of the comparability graphs. Gallai [20] also explored the idea of modular decomposition of comparability graphs.

McConnell and Spinrad gave a linear time algorithm [45] to provide a transitive orientation of a graph if one exists, using modular decomposition. Currently, the asymptotically fastest approach to checking whether this orientation is transitive uses matrix multiplication, so overall the complexity of recognizing a containment graph is $O(n^{2.376})$ [10, 57].

1.2.2 Graph Classes

Restricting the sets that establish an intersection, overlap, disjointness or containment representation restricts the graphs that can be represented. We concern ourselves mainly with representations using

geometric sets. For an illustration of the relationships between some of the classes we describe here, ee Figure 1.2.

Scheinerman [54] addressed the question: Given a graph class P, does there exist a class of sets Σ such that the graphs in P are exactly the intersection graphs of sets of type Σ?

A graph class is *monotone* if it is closed under edge and vertex removal. *Vertex expansion* is an operation that consists of taking one vertex of a graph and replacing it with a clique where each vertex in the clique has the neighborhood of the original vertex outside of the new clique. *Vertex multiplication* is an operation that consists of taking one vertex of a graph and replacing it with an independent set where each vertex in the independent set has the neighborhood of the original vertex. A graph class P is *hereditary* if for every element G of the class, every induced subgraph of G is also in P. A *composition series* for a family of graphs P is a countable collection of graphs $G_1, G_2, G_3...$ such that every G_i is in P and is an induced subgraph of G_{i+1}, and, for every graph G in P, G is an induced subgraph of G_k for some k.

Scheinerman [54] showed that a graph class P is exactly the class of the intersection graphs of some set class Σ if and only if P is monotone, closed under vertex expansion, and has a composition series. He showed a graph class P is exactly the class of the overlap graphs of some set class Σ if and only if P is hereditary, closed under vertex multiplication, and has a composition series.

Scheinerman [54] introduced the idea of injective intersection graph classes. In an injective set representation of graph $G = (V, E)$, each vertex in V is assigned exactly one distinct set - that is, no two vertices are assigned the same set. This is in contrast to general set representations, in which two vertices may be represented by the same set.

Scheinerman [54] showed that for a graph class P, there exists a family of sets Σ such that the graphs in P are exactly the injective intersection graphs of Σ if and only if P is monotone and has a composition series. Therefore, every intersection graph class is an injective intersection graph class, but not the other way around. Scheinerman pointed out the line graphs as an illustrative example - they are an injective intersection class, but not an intersection class.

Golumbic and Scheinerman's [54, 28] work suggests that a number of graph classes have undiscovered intersection, containment, and overlap characterizations.

We discuss some graph classes with known set representation characterizations.

Interval Graphs

The first intersection class to be described was interval graphs, the intersection graphs of intervals on a line. Hajós (see [29]) showed that interval graphs are chordal. Ghouila-Houra (described in translation in [29]) showed that they are cocomparability. Gilmore and Hoffman [26] showed that interval graphs are exactly the chordal cocomparability graphs.

While this characterization implies a polynomial time recognition algorithm for interval graphs, a more efficient algorithm can be derived from another result: interval graphs are exactly graphs for

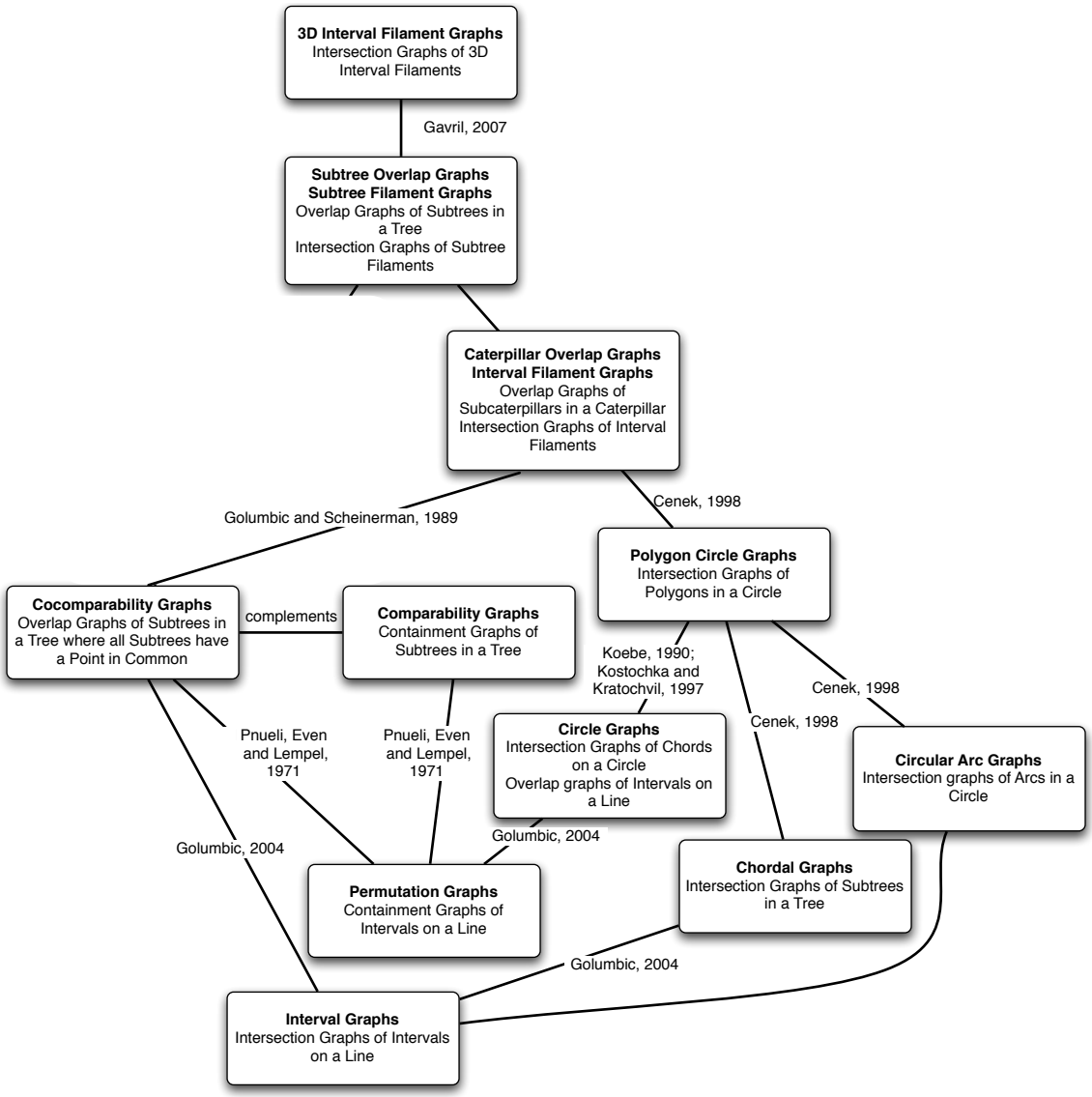

Figure 1.2: An illustration of the relationships of a number of graph classes as a hierarchy. If two graph classes are linked and one is below the other, then the upper one is a superclass of the lower one. The name of each graph class is given in bold text - for classes that appear to have two names, those two names indicate the class was described initially as two different set representation characterizations which were later shown to be equivalent. Citations indicate a source for the relationship between the graph classes.

which all maximal cliques can be linearly ordered such that the cliques containing each vertex of the graph occur consecutively [29].

Booth and Lueker [6] gave a linear time algorithm using this maximal clique ordering property: first, check whether the graph is chordal, and if so enumerate the maximal cliques. Chordal graphs have a number of maximal cliques bounded by the number of vertices. Booth and Lueker provided a linear time algorithm to check if these maximal cliques can be ordered in the appropriate way.

Chordal Graphs

Gavril [21] showed that chordal graphs are exactly intersection graphs of subtrees of a tree. Chordal graphs are well studied independent of this characterization. They are perfect and have a linearly bounded number of maximal cliques. For any graph G the following are equivalent [29, 22]:

- G is chordal,
- all minimal cutsets of G are cliques,
- G is a subtree intersection graph,
- G has a perfect vertex elimination scheme.

The last of these requires explanation, and is the basis for an efficient chordal graph recognition algorithm. A perfect vertex elimination scheme is an ordering $P = [v_1...v_n]$ of the vertices of G such that every v_i for $1 \leq i \leq n$ is simplicial in $G[v_i...v_n]$. Lexicographic breadth first search can be used to find a perfect vertex elimination scheme if one exists in $O(n + m)$ time.

Given that chordal graphs have at most a linear number of maximal cliques [21], it follows that the maximum clique of a chordal graph can be found in polynomial time. Chordal graphs are perfect, so the size of the maximum clique is the chromatic number.

Using a perfect elimination scheme, Gavril [21] provided an algorithm for finding an independent set and clique cover of the same size, guaranteeing that the independent set is maximum, and the clique cover is minimum.

Permutation Graphs

Let $G = (V, E)$ be a graph. G is a permutation graph if there exists a numbering from 1 to n of the vertices in V and a permutation π on integers $1, 2...n$ such that two vertices v_i and v_j are adjacent if and only if i and j are inverted in π.

There are several other characterizations of permutation graphs. Permutation graphs are exactly the comparability cocomparability graphs [13], and intersection graphs of line segments that connect two parallel lines [29].

These characterizations lead to polynomial-time recognition algorithms for comparability graphs and cocomparability graphs.

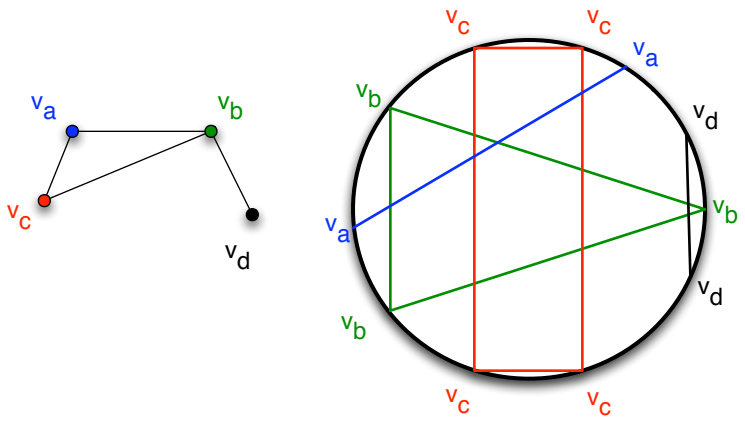

Figure 1.3: A polygon circle graph and its representation as intersecting polygons inscribed in a circle. The corners of each polygon are labeled with the name of the vertex that polygon is associated with.

An algorithm we present in this thesis uses the layers of a breadth-first search of a permutation graph rooted at a particular vertex in what we call a multi-chain ordering. This ordering is closely related to the strong ordering used by Heggernes *et al.* [34] to compute the bandwidth of bipartite permutation graphs in polynomial time.

Let $G = (V, E)$ be a bipartite permutation graph with bipartition V_1, V_2. A strong ordering of G is a pair of orderings, one of V_1 and one of V_2, such that if $a, a' \in V_1$, $b, b' \in V_2$ $(a, b) \in E$ and $(a', b') \in E$, and a is before a' in the ordering of V_1 but b' is before b in the ordering of V_2, then $(a, b') \in E$ and $(a', b) \in E$. Then the neighbourhood of a vertex $a \in V_1$ is contiguous in the ordering of V_2. The multi-chain ordering we used later in this thesis shares this property when we only consider two adjacent layers of a breadth-first search.

Polygon Circle Graphs

Polygon circle graphs were first described under the name of spider graphs by Koebe [37] as a generalization of the intersection graphs of chords in a circle.

Koebe described spiders - a generalization of chords with more than two terminals in a circle. Polygon circle graphs are the intersection graphs of spiders in a circle. They are also the intersection graphs of polygons in a circle. For an example of a polygon circle graph and its representation, see Figure 1.3.

Koebe observed that since the colouring problem is hard on circle graphs it is also hard on polygon circle graphs. However, he showed that for $k \geq 4$ and a polygon circle graph of constant bounded degree d, one can test if the graphs is k colourable in polynomial time. His algorithm generalizes an earlier one for circle graphs. He also devised an $O(s^2)$ time algorithm for finding a maximum independent set of a polygon circle graph, where s is the number of terminals in an input representation.

Kostochka and Kratochvíl [39] also studied the chromatic number of polygon circle graphs, asymptotically bounding the chromatic number of a polygon circle graph as a function of the size of that graph's largest clique.

Koebe [38] claimed to have discovered a polynomial time recognition algorithm for polygon circle graphs. This algorithm was never fully published. Recently, Pergel showed that the recognition of polygon circle graphs is NP-complete [48] using a reduction from not-all-equal-3-CNF satisfiability. In the same paper, Pergel showed a hardness result on interval filament graphs, a class we will discuss later.

The girth of a graph is the size of its shortest induced chordless cycle. Kratochvíl and Pergel [43] studied the impact of restricted girth on the recognition of polygon circle graphs. They provided a polynomial-time algorithm to recognize polygon circle graphs with girth greater than four.

Kratochvíl and Pergel [42] also defined a notion of complexity of a polygon circle representation. Consider a polygon circle representation for graph G. The *complicacy* of that representation is the number of corners of the polygon with the maximum number of corners. The complicacy of G, denoted $cmp(G)$, is the minimum complicacy of all the polygon circle representations of G. The complicacy of n, denoted $cmp(n)$, is the maximum complicacy of all polygon circle graphs on n vertices. Kratochvíl and Pergel showed that $cmp(n) = n - log_2 n + o(log_2 n)$.

Paths in Trees

There has been extensive study of intersection graphs of paths in trees. Gavril [24] studied the intersection graphs of directed paths in directed trees. Monma and Wei [46] investigated the intersection graphs of many categories of intersection graphs of paths in a tree, including the path graphs, directed path graphs, edge path graphs, directed edge path graphs, rooted directed path graphs, rooted directed edge path graphs, and Helly edge path graphs.

Monma and Wei [46] presented a unified approach to the recognition of all these classes using clique cutset decomposition, giving polynomial-time recognition algorithms for the directed path graphs, Helly edge path graphs, and directed edge path graphs.

Gavril's [24] recognition algorithm for the directed path graphs uses a different approach. His algorithm builds a canonical directed tree from a graph and a representation on that tree such that the graph has a directed path representation if and only if it has one on the canonical tree. He also showed that directed path graphs are chordal, and that proper directed path graphs are exactly directed path graphs.

Filament Intersection Graphs

Gavril [23] introduced filaments and $\mathcal{G}-$mixed partitions of graphs.

Let $G = (V, E)$ be a graph and $\mathcal{G}$ a graph class. G is $\mathcal{G}-$mixed if there is a partition of its edges into E_1 and E_2 such that:

- $G_1 = (V, E_1)$ is a comparability graph and
- $G_2 = (V, E_2)$ is in $\mathcal{G}$ and
- there is a transitive orientation $\overrightarrow{E_1}$ of E_1 such that for every pair of edges $(u \rightarrow w) \in \overrightarrow{E_1}$ and $(v, w) \in E_2$, we have $(u, v) \in E_2$.

Filaments are curves in a surface above some geometric object [23].

As an example, consider the interval filament graphs. Let $\mathcal{I}$ be a family of intervals on line L, embedded in plane Q. Let P be a surface orthogonal to Q, above Q, and intersecting Q at exactly L. For each interval $i_j \in \mathcal{I}$, let f_j be a curve in P connecting the endpoints of i_j such that for two intervals i_j, i_k, if $i_j | i_k$ then $f_j | f_k$. These curves are called interval filaments, and their intersection graphs are called interval filament graphs. For an example of an interval filament graph and an interval filament representation, see Figure 1.4. Gavril showed that the cocomparability and polygon circle graphs are subclasses of interval filament graphs.

Subtree filaments are curves in a surface above a tree. Let $\mathcal{T}$ be a family of subtrees of a tree T that is embedded in a plane P. The *filament surface* defined by T is the surface orthogonal to P that intersects P at exactly T. This surface can be imagined to be formed by drawing T upwards from P to form a surface. Filaments $\mathcal{F} = \{f_1 ... f_n\}$ on the elements of $\mathcal{T} = \{t_1 ... t_n\}$ are curves in the filament surface where each $f_i, 1 \leq i \leq n$ connects the leaves of t_i, and for two filaments $f_i, f_j \in \mathcal{F}$ corresponding to $t_i, t_j \in \mathcal{T}$:

- if $t_i | t_j$ then $f_i | f_j$ and
- if $t_i \between t_j$ then f_i intersects f_j.

Gavril [23] showed that the subtree filament graphs are exactly the complements of cochordal-mixed graphs, and the interval filament graphs (the intersection graphs of filaments on intervals on a line) are the complements of cointerval-mixed graphs.

In the same work, Gavril described an algorithm to find a maximum clique in a $\mathcal{G}-$mixed graph $G = (V, E)$ if the partition of E into E_1, E_2 is given, and there exists a polynomial-time algorithm to find maximum weight cliques for graph class $\mathcal{G}$. A similar algorithm was discovered independently by Čenek and Stewart [9].

Since there exist polynomial-time algorithms for finding maximum weight cliques in chordal graphs, cochordal graphs and cointerval graphs, circular arc graphs, and subtree cactus graphs, this approach yields polynomial-time algorithms for finding maximum independent sets in interval filament, circular arc filament, subtree filament, and subtree on a cactus filament graphs.

In 2007, Gavril further generalized his idea of filament graphs, describing the 3D-interval filament graphs [25]. This class contains the class of subtree overlap graphs.

Given that the filament framework can describe so many graph classes, and that, given an edge partition, so many problems can be solved that are otherwise hard, the recognition and place in the graph hierarchy of filament graphs is interesting.

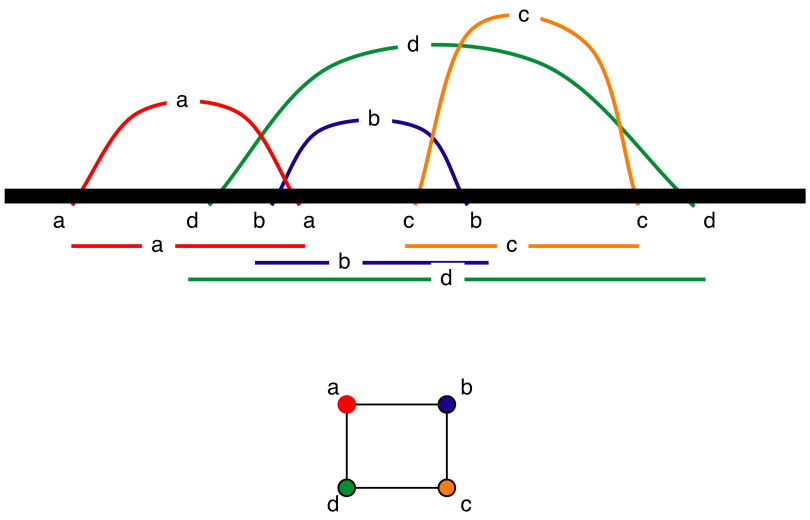

Figure 1.4: An interval filament graph below and its representation as intersecting interval filaments above. The line L is shown in thick black line and is embedded in a plane orthogonal to the page. The filament surface is in the plane of the page above L. The endpoints of the filaments are labeled with the name of the vertex they are representing. The intervals are also shown, and labeled.

Interval filament graphs are a proper superclass of interval overlap graphs. Circular arc filament graphs are a proper superclass of circular arc overlap graphs.

In contrast, the subtree overlap graphs are exactly subtree filament graphs, as shown by Enright and Stewart [15]. Given a family of filaments $\mathcal{F}$ on the family of subtrees $\mathcal{T}$ of tree T, Enright and Stewart produced a new tree T' and a new family of subtrees $\mathcal{T}'$ such that T' is T with some added leaves, and each subtree in $\mathcal{T}'$ is either the same as a subtree in $\mathcal{T}$, or else is a subtree in $\mathcal{T}$ with new leaves added. These leaves are used to ensure that all intersections between elements of $\mathcal{F}$ correspond to overlapping in $\mathcal{T}'$.

The complexities of recognising many of these filament classes are open, including the complexity of recognising subtree filament graphs. Pergel [48] showed that recognising interval filament graphs, polygon circle graphs, and any graph class that is a superclass of one and a subclass of the other is NP-hard.

1.2.3 Subtree Overlap Graphs

A graph G is a subtree overlap graph if there exists a family of subtrees $\mathcal{T}$ of tree T such that G is the overlap graph of $\mathcal{T}$.

Novillo [47] showed that not all graphs are subtree overlap graphs. He showed that a graph is not subtree overlap if it can be partitioned into two parts G_1 and G_2 such that G_1 is an induced cycle of length at least five, G_2 is a connected graph of at least five vertices, and every vertex in G_1 is adjacent to some vertex in G_2 that is adjacent to no other vertex in G_1 [47]. The smallest known non-subtree overlap graph is the cube, shown in Figure 1.5.

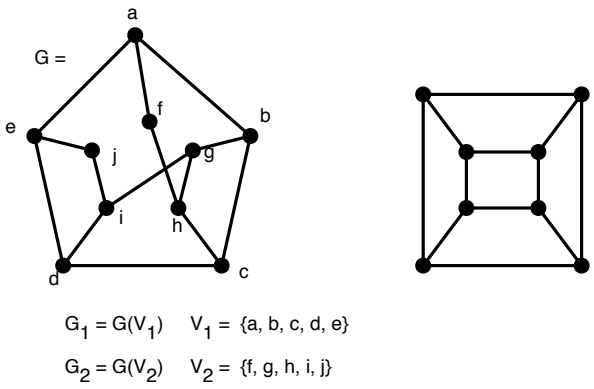

Figure 1.5: Two non-subtree overlap graphs. On the left is a non-subtree overlap graph described by Novillo [47], on the right is the cube.

Cocomparability graphs, chordal graphs, circle graphs, circular arc graphs, polygon circle graphs and interval filament graphs [8, 23], are all strict subclasses of subtree overlap graphs. Subtree overlap graphs are exactly subtree filament graphs [15].

Čenek [8] showed that every minimal subtree overlap representation of a subtree overlap graph on n vertices, is of size at most $3n$ nodes. Čenek and Stewart [9] presented a polynomial time algorithm to find a maximum clique and maximum independent set given a subtree overlap representation.

Enright [14] showed that every graph which has an induced chordal subgraph of size $n - 1$ is a subtree overlap graph; and every graph $G = (V, E)$ with a vertex partition V_1, V_2 such that both $G[V_1]$ and $G[V_2]$ are noncocomparability graphs and every vertex in V_1 is adjacent to every vertex in V_2 is not a subtree overlap graph.

The complexity of the recognition problem for subtree overlap graphs remains open.

In studying chordal graphs as intersection graphs of subtrees in a tree, the correspondence between cliques in the graph and nodes in the tree has been used to develop algorithms [22]. We comment here on the corresponding idea in studying subtree overlap graphs, and its limitations.

Cocomparability subgraphs of subtree overlap graphs

Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation of graph $G = (V, E)$. Let p be a node of T. If the subtree family $\mathcal{T}_p$ is composed of exactly the subtrees in $\mathcal{T}$ that contain p, and V_p is the set of vertices corresponding to subtrees in $\mathcal{T}_p$, then $G[V_p]$ is a cocomparability graph. At first glance, this property seems to be very useful - analogous to the fact that in a subtree intersection representation of a chordal graph the set of vertices corresponding to a family of subtrees having a node in common are a clique.

Sadly, there is a critical difference between cocomparability graphs in subtree overlap graphs and cliques in chordal graphs, with respect to the subtree representations of those classes. Assuming

a minimal subtree intersection representation for the chordal graph and a minimal subtree overlap representation for the subtree overlap graph, there is a bijection between maximal cliques in the chordal graph and nodes in its tree, but there is not between the cocomparability subgraphs of the subtree overlap graph and the nodes of its tree.

1.3 List Colouring

As described, many otherwise-hard problems have polynomial-time algorithms on intersection, overlap, and containment classes of graphs. In this thesis we will give a polynomial-time algorithm for list colouring with a fixed total number of colours on permutation and interval graphs. We proceed to introduce the list colouring problem and previous work in the area.

In the vertex colouring problem, we try to assign each vertex in a graph a colour such that no two adjacent vertices are assigned the same colour using the minimum number of colours.

In the vertex list colouring problem, each vertex has a list of colours, and we try to assign each vertex a colour from its list such that no two adjacent vertices are assigned the same colour. Determining if this is possible is NP-complete, as it is a generalization of vertex colouring [36]. List colouring remains hard even on interval graphs [2], as well as split graphs, cographs, and bipartite graphs [35]. It is solvable in $O(n^{t+2})$ time on graphs of constant treewidth t [35].

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (V_T, E_T)$ and an assignment of subsets of vertices in V to the vertices in V_T such that each vertex in V is assigned to at least one vertex in V_T, vertices that are adjacent in G are both assigned to at least one of the same vertices in V_T, and for every vertex $v \in V$, the vertices of T to which v is assigned induce a connected subtree of T. The *width* of a tree decomposition is one less than the largest number of vertices assigned to a vertex of the tree. The *treewidth* of a graph is the smallest width over all tree decompositions of that graph. Intuitively, the treewidth of a graph indicates how tree-like it is.

List colouring is $W[1]$-hard parameterized by treewidth [18], which means that there is unlikely to be an algorithm with running time $O(f(t) * (n + m)^c)$ where t is the treewidth of the graph, $f(t)$ is some function of t, and c is a constant depending on none of t, n or m.

List colouring was first described by Vizing [60] and Erdös et al. [17] independently.

Kratochvìl and Tuza [44] showed that list colouring is NP-complete even if the size of each list assigned to a vertex is at most three, each colour appears in at most three lists, each vertex in the graph has degree at most three, and the graph is planar. However, they gave polynomial-time algorithms to solve list colouring on a graph if the maximum list size is at most two, or each colour appears in at most two lists, or each vertex has degree at most two.

List Colouring with Fixed Colour Bound

List colouring with fixed colour bound k, or k-list colouring asks: given a graph and colour lists assigned to each vertex such that there are at most k colours over all the lists, is there a proper

colouring of the graph such that every vertex is assigned a colour from its list and no two adjacent vertices are assigned the same colour.

List colouring with fixed colour bound k is a generalization of k-vertex colouring, and so is NP-complete. It remains NP-complete on planar bipartite graphs [41], even with a fixed colour bound of three. It is solvable in $O(n * k^{t+2})$ (linear for fixed k) time on graphs of constant treewidth t [35].

Gravier et al. [30] studied a version of list colouring with fixed colour bound in which we are given an integer $p(j)$ for each colour j, and want to know if there is a proper colouring in which every vertex is assigned a colour on its list and each colour j is assigned to exactly $p(j)$ vertices. Following de Werra [12] they call this list colouring with fixed colour bound and *cardinalities*.

Gravier et al. [30] gave polynomial-time algorithms for solving this problem when the colour bound is two, for graphs of fixed treewidth, for chordal graphs, and for treed graphs, where treed graphs are the closure of forests under substitution. The substitution operation replaces a single vertex v of a graph with another new graph, adding edges between all neighbours of v and all vertices in the new graph.

Precolouring Extension and Intersection Graphs

Precolouring extension is a restriction of list colouring in which some of the vertices are preassigned colours, and we want to know if there is k-colouring of the graph in which the vertices preassigned colours are assigned those colours. In a k-list colouring framework, this means that each vertex's list either contains all the colours, or all but one of the colours.

Precolouring extension is NP-complete on planar bipartite graphs [41] with a fixed colour bound of three.

Let H be a fixed graph with fixed treewidth t. Recall that a *subdivision* of H is a graph formed from H by repeated subdivisions of edges. Biro et al. [2] showed that precolouring extension with fixed colour bound k can be solved in polynomial time on the intersection graphs of subgraphs of subdivisions of H.

They used this to show that precolouring extension with fixed colour bound can be solved in polynomial time on graphs of fixed treewidth because every graph is an intersection graph of a subdivision of itself.

1.4 Combinatorial Games

A *combinatorial game* is a two player game with no chance or hidden information. The players alternate turns. Under the *normal play* condition the last player to make a legal move wins. We consider only finite games - games that are guaranteed to end after a finite number of moves. A game that will continue forever is a draw. A combinatorial game must specify what constitutes a legal move from every position, and for each player.

An *impartial* combinatorial game is a combinatorial game in which from a given position the same moves are available to either player. All of the games we describe are impartial.

A game position S' is *reachable* from game position S if there is a legal sequence of moves from S that results in S'.

Perfect play is a strategy that will lead to the best possible outcome regardless of the strategy of the opponent. We generally analyse games assuming that both players are exhibiting perfect play, and assume so unless stated otherwise.

A *winning move* is a move that, with perfect play, will lead to a win for the player making that move. A *losing move* is a move that, with perfect play, will lead to a loss for the player making the move. A *winning position* is a game position from which there is a winning move. A *losing position* is a game position from which there is no winning move.

An *end position* is a game position from which there are no legal moves.

Let S be a position of a game. When we refer to the *next player* with reference to S we mean the player whose turn it is at S. That is, the player who sees the position S before he has made his move. We also use a convention from the literature, and call a winning position a $\mathcal{N}-$position, and a losing position a $\mathcal{P}-$position.

The *outcome class* of a game position is who will win with perfect play. That is, the outcome class of a position is that it is either a $\mathcal{N}-$position or a $\mathcal{P}-$position. When we say that we are *solving a position* we mean that we are finding the outcome class of that position.

The idea of the sum of two games is important in combinatorial game theory. If G and H are two games, by $G+H$ we mean a game in which the players may play on each turn in either G or H. Under normal play, the last player to make a legal move across both G and H wins $G+H$. We say that two games G and G' are *equivalent* if for every third game H the games $G+H$ and $G'+H$ are in the same outcome class.

1.4.1 Nim and Nimbers

Impartial combinatorial games can be analysed using Sprague-Grundy theory. We start our discussion of Sprague-Grundy theory with the game of nim.

In the game of nim, a position consists of some number of piles of stones, with some number of stones in each pile. Two players alternate turns. On his turn, a player can take any number of stones from any pile. He may not take stones from more than one pile on one turn. Under normal play the last player to remove a stone wins. Nim is an impartial combinatorial game.

Bouton [7] gave nim its current name, and Sprague [58], Grundy [31] (later Smith and Guy [33]) developed an extensive theory from it Nim is completely solved for all numbers and sizes of piles. Bouton [7] described the *nim-sum*, a means of adding together nim piles.

The *nimber* of each pile of nim stones is the number of stones in that pile.

The nimbers are the class of ordinals, with nim-sum (sometimes called nimber addition) and

nimber multiplication. We will not use nimber multiplication in this thesis.

The *minimum excluded ordinal* of a set of ordinals S (denoted $mex(S)$) is the smallest ordinal not in S. The nim-sum of two ordinals a, b is recursively defined as the minimum excluded ordinal of the set $\{\{a' + b \text{ where } a' < a\} \cup \{a + b' \text{ where } b' < b\}\}$ and a', b' are ordinals. The nim-sum of two ordinals is also their bitwise exclusive or value if they are expressed in binary. Nim-sum is associative and commutative [1]. Bouton [7] showed that:

Theorem 1. *A nim position is a win for the next player if and only if the nim-sum of its components (the number of stones in each of the piles) is greater than zero.*

This is helpful for analysing impartial combinatorial games in general because of the Sprague-Grundy theorem:

Theorem 2. *Every impartial combinatorial game position is equivalent to a nim-position.*

The nimber of a position in an impartial combinatorial game is the nimber of its equivalent nim-position. If we can find the nimber of a game position, we know the outcome class. Nimbers also let us add games together. Given a set of impartial combinatorial game positions and their nimbers, we can determine the outcome class of the sum of these game positions by taking the nim-sum of their nimbers.

1.4.2 Recursive definitions of games

Combinatorial game theory uses formal notation for games that we rarely use. A game position G is defined recursively based on the options for the two players, left and right, such that $G = \{\mathcal{G}^L | \mathcal{G}^R\}$ where $\mathcal{G}^L$ is the set of positions the left player could produce from G if it were his turn and $\mathcal{G}^R$ is the set of positions the right player could produce from G if it were his turn.

The base of this recursive definition is the empty game: $0 = \{|\}$, in which there are no legal moves for either player. Because there are no moves for either player, 0 is a $\mathcal{P}-$position. The game $* = \{0|0\}$ is the game in which the next player can move the game to a 0 game, and so is a $\mathcal{N}$ position. Sometimes the nim-sum of k games of $*$ is abbreviated as $k*$. We will occasionally use this notation.

In an impartial game $\mathcal{G}^L = \mathcal{G}^R$ for all positions. We will not use this notation for most of our games, instead describing a game position as the moves already made with the possible positions resulting from the next move being implicit.

1.4.3 Kayles

Row-Kayles is a combinatorial version of a game in which players knock over pins in a row. It was originally called Kayles, but as this name is now commonly used to refer to a generalisation of Row-Kayles, we will not use it.

A Row-Kayles position consists of a row of tokens, representing the pins. A player may remove one token, or two adjacent tokens. Any game position consisting of a single contiguous row of tokens with at least one token is a $\mathcal{N}-$position, by a symmetry argument. The next player need only remove the middle one or two pins, producing two contiguous rows of equal size. He can then subsequently mirror the moves of the other player in the opposite row. Guy and Smith [32] provided an algorithm for solving positions composed of more than one contiguous row.

Kayles (sometimes called Node-Kayles) is a generalisation of Row-Kayles played on a graph. In a game of Kayles, players alternate turns. On a player's turn, he chooses a vertex of the graph. A player may not choose a vertex already chosen, or a vertex adjacent to a vertex that has already been chosen. The last player to make a legal move wins.

While solving an arbitrary Kayles position is PSPACE-complete [53], positions on trees with only one vertex of degree greater than two [19], graphs with bounded asteroidal number [4], and cocomparability, permutation, interval, and circular arc graphs [3] can be solved in polynomial time. The complexity of resolving Kayles positions on trees remains open.

Bodlaender and Kratsch [5] gave an $O(1.6052^n)$ time algorithm for resolving Kayles positions on graphs on n vertices. Their approach works from the naive algorithm for solving Kayles positions on a graph $G = (V, E)$: recursively checking each induced subgraph of G and determining the winner of a Kayles game on that subgraph. They define a $K-$set of a graph $G = (V, E)$ as vertex set W that induces a connected subgraph of G such that $G[V \backslash W]$ is an independent set plus all neighbours of that independent set. They bound the number of these $K-$sets in a graph by $O(1.6052^n)$. Because the subgraphs induced by these $K-$sets are really the only subgraphs one would need to recursively consider in the trivial algorithm, this gives their $O(1.6052^n)$ time algorithm.

1.4.4 Misere Games

Under normal play, the last player to make a legal move wins. Under *misere* play the last player to make a legal move loses. Misere play games have proved to be far more resistant to theoretical analysis than normal play games. For example, Row-Kayles was solved in 1956, misere Row-Kayles was not solved until 1992 by Sibert and Conway [55].

We briefly consider a misere version of a game in this thesis. We are able to solve it because we can abstract the game to reduce the total number of possible positions of the game to a polynomial number.

1.5 Contribution

In this thesis, we present work on several aspects of set representations of graphs.

In Chapter 2 we focus on the complexity of recognising subclasses of subtree overlap graphs. We show that recognising the intersection and overlap graphs of paths in a tree with fixed maximum degree at least three is NP-complete. We show that recognising the overlap graphs of subtrees in

subdivisions of a fixed host tree with at least three leaves is NP-complete. We show that recognising the overlap graphs of subtrees in a tree with a fixed number of leaves of at least three is NP-complete.

In Chapter 3 we prove a relationship between filament graphs on specified hosts and subtree overlap graphs in which specified subgraphs of the underlying host tree are guaranteed to intersect all representing subtrees. This result implies two previous results: subtree filament graphs are subtree overlap graphs [15] and caterpillar overlap graphs are interval filament graphs.

In Chapter 4 we give a polynomial-time algorithm for list colouring permutation and interval graphs with a fixed colour bound. We define and use a vertex ordering that we call a multi-chain ordering. This ordering is related to the strong ordering of bipartite permutation graphs, and may be useful for other problems and other graph classes.

In Chapter 5 we define several combinatorial games on graphs, all of which generalise Kayles.

First, we define the *set representation game*, in which players pick sets from a provided pool to build a set representation of a given graph. At each step, the set chosen must be consistent with the set representation already partially built. The last player to make a legal move wins.

We show that the problem of determining which player has a winning strategy from a position of the set representation game is PSPACE-hard using a reduction from Kayles.

We consider several specific varieties of the set representation game, including playing the game with intervals on a line. We define a separability condition on the sets used to play the game and the set relationship corresponding to adjacency in the graph. We show that resolving positions of the set relationship game on separable sets and set relationships is PSPACE-hard. We also show that several restricted separable versions of the set representation game are in PSPACE, and are therefore PSPACE-complete.

We describe a subgraph game, in which players must choose vertices from a given graph such that the subgraph induced by the chosen vertices remains in some specified graph class. The subgraph game in which the specified graph class is the class of independent sets is exactly Kayles.

Finally, we define a set representation growing game similar to the set representation game, but instead of choosing sets from a given pool, the sets are subgraphs of a graph that we build as the game proceeds. We show the equivalence of some of our set representation games to these subgraph and growing games.

While showing that resolving positions of these games is, in general, PSPACE-hard, we give algorithms for resolving the outcomes of positions of cases of these games in polynomial time.

1.5.1 Thesis Format

This thesis is in a paper-based format. Chapters 2, 3, 4, and 5 consist of papers submitted for publication. These each include their own introduction, and so some definitions and introduction will be repeated in each chapter. Each chapter also has its own bibliography. Chapter 5 has a section of results following the paper and bibliography that contains results on set representation

and related games that was removed from the journal submission in the interest of conserving space, but has been included here for completeness.

Bibliography

[1] Michael H Albert, Richard J Nowakowski, and David Wolfe. *Lessons in play: An introduction to the combinatorial theory of games*. A K Peters, Ltd, 2007.

[2] Milos Biro, Mihaly Hujter, and Zsolt Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992.

[3] Hans L. Bodlaender. Kayles on special classes of graphs - an application of sprague-grundy theory. In *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '92, pages 90–102, London, UK, 1993. Springer-Verlag.

[4] Hans L. Bodlaender and Dieter Kratsch. Kayles and nimbers. *J. Algorithms*, 43(1):106–119, 2002.

[5] Hans L. Bodlaender and Dieter Kratsch. Exact algorithms for kayles. Technical Report UU-CS-2011-003, Department of Information and Computing Sciences, Utrecht University, 2011.

[6] Kellogg S. Booth and George S. Lueker. Linear algorithms to recognize interval graphs and test for the consecutive ones property. In *STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing*, pages 255–265, New York, NY, USA, 1975. ACM Press.

[7] C.L Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3:35–39, 1901.

[8] Eowyn Čenek. Subtree overlap graphs and the maximum independent set problem. Master's thesis, University of Alberta, Department of Computing Science, 1998.

[9] Eowyn Čenek and Lorna Stewart. Maximum independent set and maximum clique algorithms for overlap graphs. *Discrete Appl. Math.*, 131(1):77–91, 2003.

[10] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.

[11] Daniel W. Cranston, Nitish Korula, Timothy D. LeSaulnier, Kevin G. Milans, Christopher J. Stocker, Jennifer Vandenbussche, and Douglas B. West. Overlap number of graphs. *Journal of Graph Theory*, pages n/a–n/a, 2011.

[12] Dominique de Werra. Restricted coloring models for timetabling. *Discrete Mathematics*, 165-166:161–170, 1997.

[13] B. Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63:600–610, 1941.

[14] Jessica Enright. Subtree overlap graphs - towards recognition. Master's thesis, University of Alberta, Department of Computing Science, 2006.

[15] Jessica Enright and Lorna Stewart. Subtree filament graphs are subtree overlap graphs. *Inf. Process. Lett.*, 104(6):228–232, 2007.

[16] Paul Erdos, A.W. Goodman, and L. Posa. The representation of a graph by set intersections. *Canadian Journal of Mathematics*, 18:106–112, 1966.

[17] Paul Erdos, Avi Rubin, and Herbert Taylor. Choosability in graphs. *Proc. West Coast Conference on Combinatorics, Graph Theory and Computing, Arcata, Congressus Numerantium*, 22:125–157, 1979.

[18] Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Inf. Comput.*, 209(2):143–153, 2011.

[19] Rudolf Fleischer and Gerhard Trippen. Kayles on the way to the stars. In H. Jaap van den Herik, Yngvi Björnsson, and Nathan S. Netanyahu, editors, *Computers and Games*, volume 3846 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2004.

[20] Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18:25–66, 1967. 10.1007/BF02020961.

[21] Fanica Gavril. Algorithms for minimum colouring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal of Computing*, 1(2):180–187, 1972.

[22] Fanica Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3:261–273, 1973.

[23] Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.*, 73(5-6):181–188, 2000.

[24] Fănică Gavril. A recognition algorithm for the intersection graphs of directed paths in directed trees. *Discrete Math.*, 13:237–249, 1975.

[25] Fănică Gavril. 3d-interval-filament graphs. *Discrete Applied Mathematics*, 155(18):2625–2636, 2007.

[26] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and interval graphs. *Canadian Journal of Mathematics*, 16:539–548, 1964.

[27] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539 – 548, 1964.

[28] M. C. Golumbic and E. R. Scheinerman. Containment graphs, posets and related classes of graphs. *Annual New York Academy of Science*, 55:192–204, 1989.

[29] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland, 2004.

[30] Sylvain Gravier, Daniel Kobler, and Wieslaw Kubiak. Complexity of list coloring problems with a fixed total number of colors. *Discrete Applied Mathematics*, 117(1-3):65–79, 2002.

[31] P. M. Grundy. Mathematics and Games. *Eureka*, 2:6–8, 1939.

[32] R. K. Guy and C. A. B. Smith. The g-values of various games. *Proc. Cambridge Philos. Soc*, 52:514–526, 1956.

[33] Richard K. Guy and Cedric A.B. Smith. The G-values of various games. *Proc. Camb. Philos. Soc.*, 52:514–526, 1956.

[34] Pinar Heggernes, Dieter Kratsch, and Daniel Meister. Bandwidth of bipartite permutation graphs in polynomial time. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, pages 216–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[35] Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135 – 155, 1997.

[36] T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, NY, USA, 1994.

[37] Manfred Koebe. Spider graphs - a new class of intersection graphs. Master's thesis, Ernst-Moritz-Arndt-Universitaet, Sektion Mathematik, 1990.

[38] Manfred Koebe. Spider graphs - a new class of intersection graphs. Master's thesis, Ernst-Moritz-Arndt-Universitaet, Sektion Mathematik, 1990.

[39] Alexandr Kostochka and Jan Kratochvil. Covering and coloring polygon-circle graphs. *Discrete Math.*, 163(1-3):299–305, 1997.

[40] L. T. Kou, L. J. Stockmeyer, and C. K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Commun. ACM*, 21(2):135–139, 1978.

[41] Jan Kratochvil. Precoloring extension with fixed color bound. *Acta Math. Univ. Comen.*, 62:139–153, 1994.

[42] Jan Kratochvíl and Martin Pergel. Two results on intersection graphs of polygons. In Giuseppe Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2003.

[43] Jan Kratochvíl and Martin Pergel. Geometric intersection graphs: Do short cycles help? In Guohui Lin, editor, *COCOON*, volume 4598 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2007.

[44] Jan Kratochvíl and Zsolt Tuza. Algorithmic complexity of list colorings. *Discrete Applied Mathematics*, 50(3):297–302, 1994.

[45] Ross M. McConnell and Jeremy P. Spinrad. Linear-time modular decomposition and efficient transitive orientation of comparability graphs. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 536–545, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[46] Clyde L. Monma and Victor K.-W. Wei. Intersection graphs of paths in a tree. *J. Comb. Theory, Ser. B*, 41(2):141–181, 1986.

[47] Diego Novillo. Overlap graphs of subtrees in a tree. *Project for CMPUT 506*, 1994.

[48] Martin Pergel. Recognition of polygon-circle graphs and graphs of interval filaments is NP-complete. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *WG*, volume 4769 of *Lecture Notes in Computer Science*, pages 238–247. Springer, 2007.

[49] Svatopluk Poljak and Vojtech Rödl. On set systems determined by intersections. *Discrete Mathematics*, 34(2):173–184, 1981.

[50] J. L. Ramírez Alfonsín and B. Reed, editors. *Perfect graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Ltd., Chichester, 2001.

[51] Donald J. Rose, R. Endre Tarjan, and George S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 5(2):266–283, 1976.

[52] William Rosgen. Set representations of graphs. Master's thesis, University of Alberta, Department of Computing Science, 2005.

[53] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *J. Comput. Syst. Sci.*, 16(2):185–225, 1978.

[54] Edward R. Scheinerman. Characterizing intersection classes of graphs. *Discrete Mathematics*, 55(2):185–193, 1985.

[55] W. L. Sibert and J.H. Conway. Mathematical kayles. *Int. J. Game Theory*, 20:237–246, March 1992.

[56] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996.

[57] Jeremy P. Spinrad. *Efficient Graph Representations*. AMS, Providence, the fields institute monographs 19th edition, 2003.

[58] R. P. Sprague. Über mathematische Kampfspiele. *Tohoku Mathematical Journal*, 41:438–444, 1936.

[59] Edward Szpilrajn-Marczewski. Sur deux proprietes des classes des ensembles. *Fundamenta Mathematicae*, 33:303–307, 1945.

[60] Vadim G. Vizing. Coloring the vertices of a graph in prescribed colors. (in russian). *Diskret. Analiz. No. 29, Metody Diskret. Anal. v. Teorii Kodov i Shem 101*, 1976.

Chapter 2

Recognising the overlap graphs of subtrees in restricted trees is hard[1]

Intersection graphs of geometric objects are both theoretically and practically important. Given a set representation, we can solve some otherwise hard problems on intersection classes, including many problems on chordal graphs [16], and maximum weighted clique and independent set on subtree filament graphs and interval filament graphs [8].

A graph $G = (V, E)$ is an intersection graph of a set family $\mathcal{S}$ if there exists a function $f : V \to \mathcal{S}$ such that two vertices u, v are adjacent if and only if $f(u)$ intersects $f(v)$. Similarly, a graph $G = (V, E)$ is an overlap graph of a family $\mathcal{S}$ if there exists a function $f : V \to \mathcal{S}$ such that two vertices u, v are adjacent if and only if $f(u)$ overlaps $f(v)$. Two sets overlap if they intersect but neither is contained in the other.

When we consider the overlap and intersection graphs of particular types of sets, we define graph classes.

Part of the theoretical interest in geometric intersection and overlap graphs stems from efficient algorithms for otherwise NP-hard problems on these graph classes. Often, these algorithms require as input a set intersection representation of a particular type. Thus we are interested in whether or not a given graph has a particular type of intersection representation. This is called the *recognition problem.*

Probably the oldest intersection-defined graphs are *interval graphs*, the intersection graphs of interval on a line [9]. The interval graphs are generalised by intersection graphs of paths in a tree [4, 14]. Intersection graphs of paths in a tree are in turn generalised by *chordal graphs*. While primarily defined as the graphs without induced cycles of length greater than three, chordal graphs are also exactly the intersection graphs of subtrees in a tree [7]. The leafage of a subtree intersection representation is the number of leaves in the host tree. The leafage of a chordal graph is the leafage of a subtree intersection representation of that graph with smallest leafage. The study of leafage of

[1] This is joint work with Martin Pergel, submitted to the Journal of Discrete Math and Theoretical Computer Science on September 16, 2011

chordal graphs has spanned decades.

The overlap analogue of chordal graphs is the class of *subtree overlap graphs*, the overlap graphs of subtrees in a tree. Subtree overlap graphs generalize many set representation characterized classes, including chordal graphs and therefore interval graphs.

In this paper we consider an overlap extension of leafage. The leafage of a subtree overlap representation is the number of leaves in the host tree. The leafage of a subtree overlap graph is the leafage of a subtree overlap representation of that graph with minimum leafage.

Gavril [8] defined *interval filament graphs* and *subtree filament graphs* as intersection graphs of *filaments on intervals* and *filaments on subtrees*, respectively. Filaments are curves above some geometric structure (in this case above intervals or subtrees) such that filaments above disjoint structures must not intersect, while filaments above overlapping structures must intersect.

Interval filament graphs are a subclass of subtree overlap graphs, and subtree filament graphs are exactly subtree overlap graphs [6, 5].

Recognising interval filament graphs is known to be hard [8, 15]. In contrast, we can recognise interval graphs and chordal graphs in linear time [3, 16], and intersection graphs of paths in a tree in $O(nm)$ time, where n is the number of vertices and m the number of edges in the input graph [18]. The complexity of recognising subtree overlap graphs is open.

With this in mind, we define three subclasses of subtree overlap graphs: we define k**-SOG** to be the *overlap graphs of subtrees in a tree with leafage* k, class k**-degree-POG** to be the overlap graphs of subpaths in a tree such that the tree has maximum degree at most k, and the class **T-SOG** to be the overlap graphs of subtrees of a trees derived from an input tree T by subdivision of edges.

Though we expect the recognition of subtree overlap graphs to be NP-complete, we expected the recognition of these simplified SOGs to be polynomial time. We were therefore surprised when we obtained hardness results for the recognition problems of k-SOG and k-degree-POG for fixed integer $k \geq 3$ and for $T - SOG$ provided that T has at least three leaves. We present these hardness results in this paper.

Our result on the hardness of recognising k-degree-POG also holds for the corresponding class of intersection graphs. Our reduction also shows that it is NP-complete to recognise intersection graphs of paths in a tree with a fixed maximum degree greater than two. In contrast, intersection graphs of paths in a tree can be recognised in polynomial time.

Our result on the hardness of recognising the subtree overlap graph with k leafage for fixed integer $k \geq 3$ provides a counterpoint to work on the intersection leafage of chordal graphs. Stacho and Habib [20] give a polynomial-time algorithm for determining the leafage of a chordal graph and constructing a representation that achieves that leafage. In contrast, we show that determining the overlap leafage of a subtree overlap graph is NP-Hard.

There has been substantial work on the intersection graphs of subtrees or paths in a tree with parameterisation of the subtrees or the underlying tree.

Jamison and Mulder [13] considered the intersection graphs of subtrees of a tree parameterised by maximum degree of both the underlying tree and the individual subtrees. They showed that the intersection graphs of subtrees of a tree in which the subtrees and the underlying tree have bounded maximum degree 3 are exactly chordal graphs, and so can be recognized in linear time. This contrasts to our work, which shows that recognizing the intersection graphs of paths in an underlying tree of bounded maximum degree 3 in NP-hard.

Golumbic and Jamison [11] showed that recognizing the edge-intersection graphs of paths in a tree is NP-complete, and showed that on a tree with maximum degree three, the edge-intersection and vertex-intersection graphs are the same classes.

Golumbic et al [12] explore the complexity of recognizing the intersection graphs of paths in a tree parameterised by both the maximum degree of the underlying tree and the number of vertices that must be shared between two paths for them to be considered as intersecting. They provide a complete hierarchy of graph classes using these parameters.

2.1 Preliminaries and Definitions

All graphs discussed here are simple, undirected, loopless and finite. If $G = (V, E)$ is a graph, and $V' \subset V$ a vertex subset, then $G[V']$ denotes the subgraph of G induced by V'. We generally adhere to notation used in [10].

Let s_i and s_j be two sets. If $s_i \cap s_j = \emptyset$ then we write $s_i | s_j$, pronounced s_i is **disjoint** from s_j. If $s_i \cap s_j \neq \emptyset$, $s_i \not\subseteq s_j$ and $s_j \not\subseteq s_i$ then we write $s_i \between s_j$, pronounced s_j **overlaps** s_j. Let $G = (V, E)$ be the overlap graph of family of sets $\mathcal{S}$. Then for convenience, we refer to the set corresponding to vertex $v_i \in V$ as $s_i \in \mathcal{S}$. For example, in a subtree representation v_i is a vertex corresponding to a subtree t_i.

Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation. We say that T is the **host tree**.

Let t be a subtree of tree $T = (V_T, E_T)$. A **boundary node** of t is a node of T that is in t, and either has a neighbour $u \in V_T$ such that u is not in t, or is a leaf of T.

Observation 1. *Let q be a node of subtree t of tree T. Node q is a boundary node of t if and only if either q is a leaf of T or t does not contain all neighbours of q in T.*

SUB(T) is the set of all trees that can be derived from tree T by subdividing the edges of T any number of times. A **twig** of tree T is a maximal path of T that includes a leaf of T and no node of T that has degree greater than two. A **lastbranch** of $T = (V_T, E_T)$ is a node p of T of degree at least three such that the forest formed by $T[V_T \backslash \{p\}]$ has at most one connected component that is not a path. For an example of lastbranch nodes, boundary nodes and twigs see Figure 2.1.

Unless otherwise noted, if we say that a graph G **can be represented on a tree** T we mean that there exists an overlap representation of G with T as the host tree.

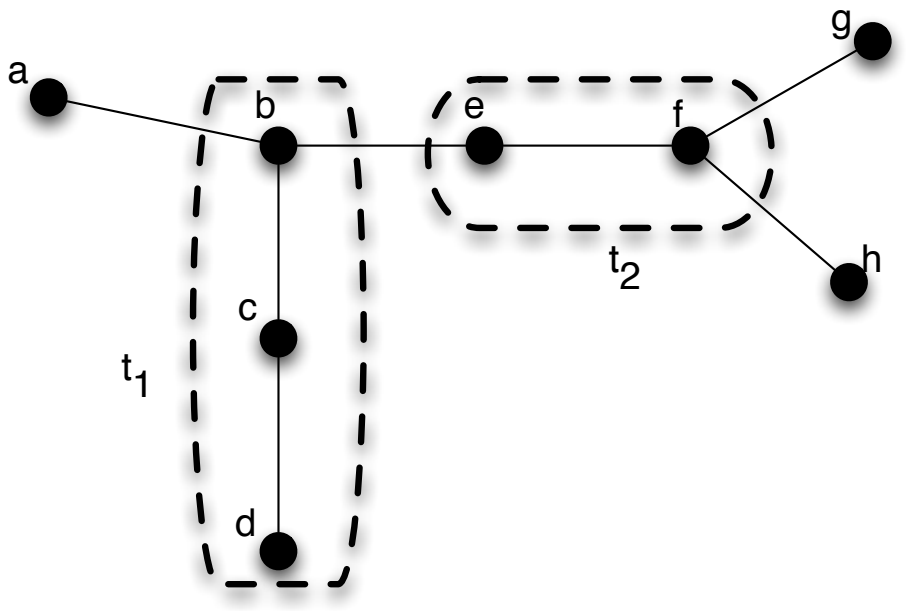

Figure 2.1: A tree T and two indicated subtrees: t_1 and t_2. Nodes b and f are the only lastbranch nodes of T. The nodes b and d are boundary nodes of t_1 and nodes e and f are boundary nodes of t_2. The paths consisting of the single vertices a, g, h, as well as the two vertex path of c and d form the twigs of T.

Let $G = (V, E)$ and $G'' = (V'', E'')$ be graphs such that there exists a vertex set $V' \subset V$ and $V' \subset V''$ and G'' is the overlap graph of subtrees $\mathcal{T}''$ of tree T''. Let $\mathcal{T}' \subset \mathcal{T}''$ be the family of subtrees corresponding to the members of V'. We say that V' is **nicely represented with respect to** G if every member of $\mathcal{T}'$ is contained in a twig of T'', and there are no two members v_i, v_j of V' such that $(v_i, v_j) \in E$ and t_i and t_j are on the same twig of T''. For an example of nicely represented vertex sets, see Figure 2.2.

If a vertex set V' is nicely represented with respect to a graph G, then each twig of the nice representation corresponds to an independent subset of V' in G, and so:

Observation 2. *Let $G = (V, E)$ be a graph. If there exists a representation of some graph on tree T with k twigs in which vertex subset $V' \subset V$ is nicely represented with respect to G, then there is a k-colouring of $G[V']$.*

2.2 Preliminary Lemmas

We prove a number of preliminary lemmas on subtrees and subtree representations that we will refer to in later proofs.

Definition 1. *Let i_j, i_k be intervals on line I. The* interpath *of i_j, i_k is the interval of I that contains i_j, i_k, and the portion of I between them.*

Lemma 1. *Let $\mathcal{I}$ be a set of disjoint intervals on a line I, such that each $i \in \mathcal{I}$ is assigned at least one partner in I - this partnership is symmetric. Then at least one of every three intervals contained in $\mathcal{I}$ is disjoint from the interpath of at least one pair of partners in $\mathcal{I}$.*

Proof:

Let $\mathcal{I}$ be as described above. Let i_j, i_k, i_l be three members of $\mathcal{I}$. One of these three intervals must be between the other two in I. Without loss of generality, let i_k be between i_j and i_l on I. For convenience, we will arbitrarily say that i_j is to the left of i_k and i_l is to the right of i_k. Let i_h be a partner of i_k. Note that it is possible that $i_h = i_j$ or $i_h = i_l$.

If i_h is to the right of i_k, then the interpath of i_h, i_k is disjoint from i_l. If i_h is to the left of i_k, then the interpath of i_h, i_k is disjoint from i_j. □

Lemma 2. *Let $t_i \subseteq t_j$ be subtrees of tree T. If t_i has k boundary nodes, t_j has at least k boundary nodes.*

Proof. Let $t_i \subseteq t_j$ be trees. Consider the boundary nodes of t_i. We show that there is at least one boundary node of t_j for every boundary node of t_i. Let p be a boundary node of t_i. Either p is a boundary node of t_j or it is not. If it is not, then, as t_j is finite, there must be a leaf of t_j that is disconnected from t_i by removing p. This leaf is not disconnected from all nodes of t_i by removing any other boundary node of t_i, by the fact that there are unique paths between points in a tree. Therefore there is at least one boundary node of t_j for every boundary node of t_i. □ □

Lemma 3. *Let $\mathcal{T}$ be a family of subtrees of tree T such that $G = (V, E)$ is the overlap graph of $\mathcal{T}$. Let p be a non-leaf node of T. Let T_1 and T_2 be two components of $T \backslash p$. Let $V_p \subseteq V$ be the vertices represented by subtrees in $\mathcal{T}$ that contain p. Let $V_1 \subseteq V$ be the vertices represented by subtrees in $\mathcal{T}$ that are contained in T_1. Let $V_2 \subseteq V$ be the vertices represented by subtrees in $\mathcal{T}$ that are contained in T_2. Then removing V_p from G disconnects every member of V_1 from every member of V_2.*

Proof. This follows from the fact that subtrees contained in T_1 are disjoint from subtrees contained in T_2. □ □

Lemma 4. *Let $t_i \subseteq t_j$ be subtrees of tree T such that t_i and t_j both have k boundary nodes and every boundary node of t_j has degree in t_j at most two. Then every node of t_j of degree greater than two is contained in t_i.*

Proof. Let $t_i \subseteq t_j$ be subtrees with k boundary nodes. Assume that there is a node p of degree $d > 2$ contained in t_j that is not contained in t_i.

Let q be the node of t_i closest in number of edges to p. Let t_h be the union of p, the path between p and q, the neighbourhood of p, and t_i. By this construction $t_h \subseteq t_j$. How many boundary nodes does t_h have? Every boundary node of t_i except for possibly q is a boundary node of t_h. Every neighbour of p except the one on the path between p and q is a boundary node of t_h. Therefore t_h

has at least $d+k-2$ boundary nodes. Since $d > 2$, then t_h has at least $k+1$ boundary nodes. Since $t_h \subseteq t_j$, and t_j has k boundary nodes, this is a contradiction to Lemma 2. □

□

Lemma 5. *Let T be a tree with k leaves, and subtrees t_i and t_j such that $t_i|t_j$. If t_i has l boundary nodes and t_j has $k-l+2$ boundary nodes, then all nodes of T of degree greater than two are in either t_i or t_j.*

Proof. Let T, t_i, t_j be as described in the lemma statement.

Let $p \in t_i$ and $q \in t_j$ be two nodes that minimise the distance between them. By these conditions on p and q it holds that p is a boundary node of t_i and q a boundary node of t_j, and the path between p and q excluding those vertices is disjoint from both t_i and t_j. Let t_h be the subtree that is the union of t_i, t_j, and the path between p and q.

How many boundary nodes does t_h have? Every boundary node of t_i except for p is a boundary node of t_h. Every boundary node of t_j except for q is a boundary node of t_h. Therefore t_h has at least $l+k-l+2-2=k$ boundary nodes. Since $t_h \subseteq T$ and T has k leaves, by Lemma 2 t_h also has at most k boundary nodes, and so has exactly k boundary nodes. Therefore a node is a boundary node of t_h if and only if it is a boundary node of t_i or t_j that is not p nor q.

There are therefore no boundary nodes of t_h contained in the path between p and q. Then no node in that path has a neighbour outside that path, and so every node in that path is of degree two. Since that path is the only portion of t_h that is not is one of t_i or t_j, every node in t_h of degree greater than two is contained in one of t_i or t_j.

Since t_h has k boundary nodes by Lemma 4, the subtree t_h contains all nodes of degree greater than two in T. Then by our earlier statement that every node in t_h of degree greater than two is in one of t_i or t_j, we have that every node in T of degree greater than two is contained in either t_i or t_j. □

Lemma 6. *Let $t_i|t_j$ be two disjoint subtrees of tree T where T has k leaves. If t_i has l boundary nodes, then t_j has at most $k-l+2$ boundary nodes.*

Proof. Let T, t_i, t_j be as described in the lemma statement. Assume that t_j has more than $k-l+2$ boundary nodes, and proceed by contradiction.

Let $p \in t_i$ and $q \in t_j$ be two nodes that minimise the distance between them. By these conditions on p and q it holds that p is a boundary node of t_i and q a boundary node of t_j, and the path between p and q excluding those vertices is disjoint from both t_i and t_j. Let t_h be the subtree that is the union of t_i, t_j, and the path between p and q.

How many boundary nodes does t_h have? Every boundary node of t_i except for p is a boundary node of t_h. Every boundary node of t_j except for q is a boundary node of t_h. Therefore t_h has at least $((l-1)+(k-l+2))=k+1$ boundary nodes. Since T has k leaves, and $t_h \subseteq T$, this is a contradiction. □

□

Then, as a corollary:

Corollary 1. *Let T be a tree with subtrees t_i and t_j such that $t_i|t_j$, t_i has k boundary nodes, and t_j has l boundary nodes. Then T has at least $k + l - 2$ leaves.*

Lemma 7. *Let T be a tree with k leaves, and p a node of T of degree two that is not adjacent to a leaf. Then the forest created by removing p from T has two connected components such that the sum of their numbers of leaves is $k + 2$.*

Proof. Let t_i, t_j be the subtrees of T in the forest created by removing p from T. Every leaf of T is a leaf of exactly one of t_i or t_j. In addition, the neighbours of p in t_i, t_j are leaves in each of those. There are no other leaves of t_i or t_j. Therefore t_i, t_j have $k + 2$ leaves between them. □

From Rosgen [17] we have the following lemma:

Lemma 8. *Let graph $G = (V, E)$ be represented by subtrees $\mathcal{T}$ of tree T. Let v_i, v_j be non-adjacent vertices in V. If $t_i \subseteq t_j$ then for every vertex v_k such that there exists a path of G from v_i to v_k that does not intersect the neighbourhood of v_j, it holds that $t_k \subseteq t_j$.*

We will later need this technical lemma on adjusting overlapping in a subtree overlap representation:

Lemma 9. *Let $\mathcal{T}$ be a family of subtrees of tree T. Let $\mathcal{T}_1$ be a subfamily of $\mathcal{T}$. If:*

- *there is a node q of T that is in every subtree in $\mathcal{T}_1$, no subtree in $\mathcal{T}_1$ contains all neighbours of q, and there is no subtree $t \in \mathcal{T} \backslash \mathcal{T}_1$ for which q is a boundary node*

Then:

- *there exists a subtree overlap representation with subtrees $\mathcal{T}'$ of tree T' and a bijection $f : \mathcal{T} \to \mathcal{T}'$ such that for every two trees $t_i \in \mathcal{T} \backslash \mathcal{T}_1$ and $t_j \in \mathcal{T}$ the set relationship between t_i and t_j is the same as the set relationship between $f(t_i)$ and $f(t_j)$, for every two subtrees $t_i, t_j \in \mathcal{T}_1$ $f(t_i) \between f(t_j)$, and T' can be derived from T by repeatedly subdividing the edges between q and its neighbours.*

Proof. Let O be an ordering of the subtrees in $\mathcal{T}_1$ by non-increasing size. We will denote the position of subtree t in O by $O(t)$.

Let tree T' be the tree produced from T by subdividing the edges between q and its neighbours $|\mathcal{T}_1|$ times, so that there is a path of $|\mathcal{T}_1|$ nodes, not including q and its neighbour in T, in T' between q and each of the nodes that were its neighbours in T.

We define each subtree $f(t_i)$ for $t_i \in \mathcal{T}$. If $t_i \in \mathcal{T} \backslash \mathcal{T}_1$ and $q \notin t_i$, then $f(t_i)$ is t_i. If $t_i \in \mathcal{T} \backslash \mathcal{T}_1$ and $q \in t_i$, then $f(t_i)$ is t_i and the new paths between q and each node p that was in t_i and was a neighbour of q in T.

If $t_i \in \mathcal{T}_1$ then $f(t_i)$ is t_i and the new paths between q and each node p that was in t_i and was a neighbour of q in T and all other nodes within distance $O(t_i)$ of q.

Since we produced T' by iterated subdivision of T, it remains to prove that for every two trees $t_i \in \mathcal{T}\backslash\mathcal{T}_1$ and $t_j \in \mathcal{T}$ the set relationship between t_i and t_j is the same as the set relationship between $f(t_i)$ and $f(t_j)$ and for every two subtrees $t_i, t_j \in \mathcal{T}_1$ $f(t_i) \between f(t_j)$.

Let t_i be a subtree in $\mathcal{T}\backslash\mathcal{T}_1$ and t_j a subtree in $\mathcal{T}$. Consider the difference between a subtree $t \in \mathcal{T}$ and $f(t)$. By the construction of $f(t)$, we know that for every subtree $t \in \mathcal{T}$, it holds that $t \subseteq f(t)$, and the only nodes in $f(t)$ that might not be in t are nodes that are in T' but not T. That is, $f(t)\backslash t \subseteq T'\backslash T$.

We consider cases: $t_i \between t_j$, $t_i | t_j$, $t_i \subseteq t_j$, and $t_j \subseteq t_i$. If $t_i \between t_j$ then there exists a node a in both t_i and t_j, b in t_i but not t_j and c in t_j but not t_i. Then because $f(t_j)\backslash t_j \subseteq T'\backslash T$ and $f(t_i)\backslash t_i \subseteq T'\backslash T$ the node a is in both $f(t_i)$ and $f(t_j)$, b is in $f(t_i)$ but not $f(t_j)$ and c is in $f(t_j)$ but not $f(t_i)$. Therefore $f(t_i) \between f(t_j)$.

If $t_i | t_j$ then there exist no nodes in both t_i and t_j. By the construction of each $f(t)$, if $q \notin t$, then $f(t) = t$. The node q is in at most one of t_i or t_j. Without loss of generality, let t_i not contain q. Then $f(t_i) = t_i$. Since $f(t_j)\backslash t_j \subseteq T'\backslash T$ the only nodes that could possibly be in $f(t_j)$ that are not in t_j, are nodes that are not in T, and therefore not in $f(t_i)$. Then no nodes of $f(t_j)$ are in $f(t_i)$, and $f(t_i) | f(t_j)$.

If $t_i \subseteq t_j$, we need consider two possibilities: either $q \in t_i$ or $q \notin t_i$. If $q \notin t_i$ then $f(t_i) = t_i \subseteq t_j \subseteq f(t_j)$. If $q \in t_i$ then $q \in t_j$, and any neighbour of q in t_i is also in t_j. Since t_i is not in $\mathcal{T}_1$, every neighbour of q in T is in t_i, and so also in t_j. Therefore t_j is also not in $\mathcal{T}_1$.

Then, by the construction of $f(t_i)$ and $f(t_j)$, any node that is is $f(t_i)\backslash t_i$ is also in $f(t_j)$. Therefore $f(t_i) \subseteq f(t_j)$.

If $t_j \subseteq t_i$, then both t_i and t_j contain q, and t_i contains all neighbours of q in T because q is not a boundary node for any vertex in $\mathcal{T}\backslash\mathcal{T}_1$. Since $f(t_i)$ then contains all nodes in $T'\backslash T$ and $f(t_j)\backslash t_j \subseteq T'\backslash T$ we have that $f(t_j) \subseteq f(t_i)$.

All that remains is to show that for $t_i, t_j \in \mathcal{T}_1$, it holds that $f(t_i) \between f(t_j)$. Without loss of generality, assume that $O(t_i) < O(t_j)$. By the conditions on $\mathcal{T}_1$, there is at least one neighbour p of q in T that is not contained in t_i. By the construction of $f(t_i)$, the node p is not contained in $f(t_i)$, but the first $O(t_i)$ nodes from q on the path to p are. If p is contained in t_j, then it is also contained in $f(t_j)$. If p is not contained in t_j, then the first $O(t_j)$ (recall that this is greater than $O(t_i)$) nodes from q on the path to p are. In either case, there is a node contained in $f(t_j)$ that is not contained in $f(t_i)$, and therefore $f(t_j) \not\subseteq f(t_i)$.

Since $O(t_i) < O(t_j)$ and we have assumed that no two subtrees are equal, we know that $t_i \not\subseteq t_j$. Therefore there is some node of t_i not in t_j. Because $f(t_j)\backslash t_j \subseteq T'\backslash T$ that node is not in $f(t_j)$, but is in $t_i \subseteq f(t_i)$, and $f(t_i) \not\subseteq f(t_j)$.

Since $f(t_i)$ and $f(t_j)$ both contain q but neither contains the other they overlap. $\square$

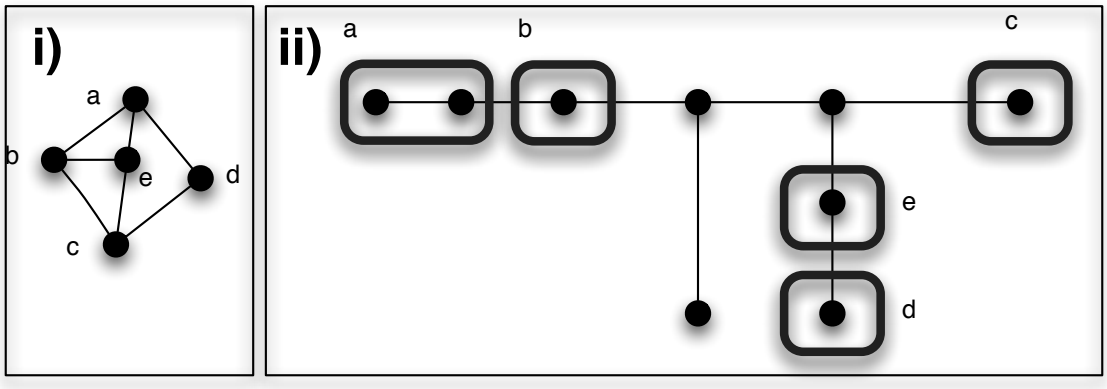

Figure 2.2: In i) a graph $G = (V, E)$, and in ii) a family $\mathcal{T}$ of subtrees of a tree T, with one subtree corresponding to each of the vertices of G. Note that $\mathcal{T}$ is not a representation of G, but of some other graph that has V as a subset of its vertices. In the representation on the right vertex sets $\{a, c, d, e\}$, $\{b, c, d, e\}$ are nicely represented with respect to G. Any vertex set including both a and b is not nicely represented with respect to the graph on the left, as a and b are adjacent in G but are both on the same twig of the tree on the right.

□

2.3 Problem Descriptions

Given an input graph $G = (V, E)$:

Problem 1. *The problem REC-PMD-k is the decision problem for a fixed natural number $k \geq 3$: does there exist a tree T with maximum degree k and a family $\mathcal{T}$ of paths of T such that G is the overlap graph of $\mathcal{T}$?*

Problem 2. *The problem REC-SUB-T is the decision problem for a fixed tree T with at least three leaves: does there exist a tree $T' \in SUB(T)$ and a family $\mathcal{S}$ of subtrees of T' such that G is the overlap graph of $\mathcal{S}$?*

Problem 3. *The problem REC-LEAFAGE-k is the decision problem for a fixed natural number $k \geq 3$: does there exist a tree T with k leaves and a family $\mathcal{S}$ of subtrees of T such that G is the overlap graph of $\mathcal{S}$?*

Cenek [2] showed that every minimal subtree overlap representation of a graph G is of size polynomial in the size of G, and can be checked for correctness in polynomial time. This would serve as a certificate, so we can conclude that REC-SUB-T, REC-LEAFAGE-k, and REC-PMD-k are in NP.

Problem 4. *The problem 3-CON-k-COLOURING is the decision problem for a fixed natural number $k \geq 3$: given a 3-connected graph G, is there a proper vertex colouring of G using k colours?*

Theorem 3. *3-CON-k-COLOURING is NP-Complete for fixed natural number $k \geq 3$.*

Proof. This proof is straightforward, but we have provided it here for completeness.

Firstly, the problem is in NP, as it is a decision problem and a colouring serves as a polynomial-sized certificate.

We now supply a reduction from $k-$colouring a connected graph. Let $G = (V, E)$ be a connected graph. Let V_1, V_2, V_3 be three vertex sets of the same size as V such that there are bijections f_1, f_2, f_3 from V_1, V_2, and V_3, respectively, to V.

We now define edge sets:

1. $E_1 = \{(v_i, v_j)$ where $v_i, v_j \in V_1$ and $(f_1(v_i), f_1(v_j)) \in E\}$
2. $E_2 = \{(v_i, v_j)$ where $v_i, v_j \in V_2$ and $(f_2(v_i), f_2(v_j)) \in E\}$
3. $E_3 = \{(v_i, v_j)$ where $v_i, v_j \in V_3$ and $(f_3(v_i), f_3(v_j)) \in E\}$
4. $E_4 = \{(v_i, v_j), (v_i, v_k), (v_j, v_k)$ where $v_i \in V_1, v_j \in V_2, v_k \in V_3$ and $f_1(v_i) = f_2(v_j) = f_3(v_k)\}$

Let $V' = V_1 \cup V_2 \cup V_3$ and $E' = E_1 \cup E_2 \cup E_3 \cup E_4$. Then let $G' = (V', E')$. G' is 3-connected.

We now show that G' is k-colourable for $k \geq 3$ if and only if G is $k-$colourable.

First, assume that G' is k-colourable. Since G is isomorphic to an induced subgraph of G', then G is k-colourable. Next, assume that G is k-colourable. Let $C_0 ... C_{k-1}$ be the colour classes of G. Then we can colour G' as follows: For each vertex $v_i \in V_h$ where $h \in \{1, 2, 3\}$, let C_j be the colour class of $f_h(v_i)$. Then we assign v_i to colour class $C_{(j+(h-1)) \mod k}$.

It remains to prove that no two adjacent vertices in G' have been assigned the same colour.

We consider two cases: either two adjacent vertices are in the same one of V_1, V_2, V_3, or they are in different ones. Let $h \in \{1, 2, 3\}$. Let $v_i, v_j \in V_h$ such that $(v_i, v_j) \in E_h$. Then let C_g and C_l be the colour classes of of $f_h(v_i)$ and $f_h(v_j)$, respectively. Since $(f_h(v_i), f_h(v_j)) \in E$, it holds that $g \neq l$. We also know that $g < k$ and $l < k$. From the construction of the colour classes v_i is in colour class $C_{(g+(h-1)) \mod k}$ and v_j is in colour class $C_{(l+(h-1)) \mod k}$. Given the conditions on g and l, we know that $(g + (h-1)) \mod k \neq (l + (h-1)) \mod k$. Therefore v_i, v_j are in different colour classes.

Now we turn our attention to the case in which two adjacent vertices are in different ones of V_1, V_2, V_3. Let $h, g \in \{1, 2, 3\}$ but $h \neq g$. Let $v_i \in V_h$ and $v_j \in V_g$. By the edge set construction, if $(v_i, v_j) \in E_4$, it holds that $f_h(v_i) = f_g(v_j)$. Let C_l be the colour class of $f_h(v_i)$ (and $f_g(v_j)$).

Then the colour class of v_i is $C_{(l+(h-1)) \mod k}$ and the colour class of v_j is $C_{(l+(g-1)) \mod k}$. Since $g \neq h$ and $g \leq 3, h \leq 3$ and $k \geq 3$, these are different colour classes. □

□

2.4 Reduction Intuition

In this paper, we describe reductions from 3-CON-k-COLOURING to REC-SUB-T, REC-LEAFAGE-k, and REC-PMD-k.

All of our reductions work on the same intuition. We reduce from an instance of graph colouring to an instance of a representability problem.

We start with an instance of k-colouring in the form of a 3-connected graph G. We transform this graph into another graph G''. G'' will be our instance of the representability problem.

In the transformation from G to G'', the vertices and edges of G are used as vertices of G''. Other gadgets are added to G'' in such a way that for REC-SUB-T and REC-LEAFAGE-k, twigs in the host tree of the representation correspond to colours used for a k-colouring of graph G. That is, no two vertices corresponding to adjacent vertices in G are represented on the same twig of the representation of G''.

Similarly for REC-PMD-k, we start with an instance of k-colouring, G, and transform it to another graph G'' that will be our instance of REC-PMD-k. G'' includes the vertices and edges of G in its vertex set. Given a representation of G'', we use subtrees of the host tree that would form connected components of the forest that would result from removal of a fixed vertex v of degree k to correspond to colours used for a k-colouring of graph G. No two gadgets corresponding to adjacent vertices in G are represented on the same branch of the representation of G''. That is, their representing subtrees cannot contain v. We do this by forcing all vertices of G'' that are edges of G to contain v, and by using another class of gadget vertices.

The overarching idea is that for each of the reductions to the problems we are considering we associate sections of the host tree with one of k colours, and force all vertices of G'' that are also vertices of G to be represented in one of these sections.

To deal with a technical complication, the vertex set of G'' includes several copies of the vertex set of G as we show that a constant number of vertices may be represented in a bad way. By adding more copies than the number of possible bad exceptions, at least one copy must fulfill our requirements, allowing us to derive a colouring of G from a representation of G''.

We more formally describe our reductions in Sections 6 through 8 of this paper.

2.5 Discussion

There are two particularly interesting results here.

Intersection graphs of paths of a tree with maximum degree k (k-degree-PIG) are all overlap graphs of paths of a tree with maximum degree k (k-degree-POG). We justify this as follows: Let paths $\mathcal{P}$ of tree T with fixed maximum degree k be an intersection representation of graph $G = (V, E)$. We describe how to change the paths in $\mathcal{P}$ and the tree T such that a single path $P \in \mathcal{P}$ is not contained in any other path in $\mathcal{P}$, and every pair of paths intersect if and only if they did before

our change. Let p be a leaf node of path $P \in \mathcal{P}$ and q the neighbour of p not in $\mathcal{P}$. We add two new nodes r, s to T such that s subdivides the edge between p and q and r is adjacent only to s. We add s to every path in $\mathcal{P}$ that contains p and r only to P. Then P is contained in no other path in $\mathcal{P}$ and two paths in $\mathcal{P}$ intersect if and only if they did before our change. We can iteratively apply this change for every path in $\mathcal{P}$ that is contained in some other path. This will give an intersection representation in which the maximum degree of the tree is the same as the original tree, and no path is contained in any other. This representation is therefore also an overlap representation of the same graph. Therefore every intersection graph of paths in a tree with fixed maximum degree k is also the overlap graph of paths in a tree with fixed maximum degree k.

We present a polynomial time and space reduction from an instance of graph k-colouring to a graph that is in k-degree-PIG if and only if it is in k-degree-POG. Let $\mathcal{Z}$ be a graph class between k-degree-PIG and k-degree-POG. That is, k-degree-PIG $\subseteq \mathcal{Z} \subseteq k$-degree-POG. The graph produced by our reduction is in $\mathcal{Z}$ if and only if it is in k-degree-PIG if and only if it is in k-degree-POG. Therefore the recognition problem for every such graph class $\mathcal{Z}$ is also NP-complete.

We show that recognising the overlap graphs of subtrees of a tree with fixed integer $k \geq 3$ leaves is NP-complete. The overlap graphs of subtrees of a tree with fixed integer $k < 3$ leaves are circle graphs [10].

We show that recognising the overlap and intersection graphs of paths of a tree with maximum degree $k \geq 3$ is NP-complete. The overlap and intersection graphs of paths of a tree with maximum degree $k < 3$ leaves are exactly circle graphs and interval graphs, respectively. Spinrad [19] gives a linear-time recognition algorithm for circle graphs, and Booth and Lueker [1] gives a linear-time recognition algorithm for interval graphs.

We therefore have a dichotomy. For $k \geq 3$, the recognition problems for overlap graphs of subtrees of a tree with k leaves, and overlap and intersection graphs of paths in a tree with maximum degree k are NP-complete. For $k < 3$, we have polynomial-time solutions to these problems.

2.6 G_d^u-blocking Graphs

Let $d \geq 3$ and u where either $u \neq 1$ be two natural numbers. We use these as parameters in building a graph we call the G_d^u graph. The G_d^u graph consists of four named vertices, v_s, v_b, v_{s2}, v_{b2} connected by a number of paths of length three specified by the integers d and u.

Definition 2. *G_d^u is the graph shown in Figure 2.3a.*

Having defined the G_d^u graph, we will now describe the production, given an input graph G, of a new graph called the G_d^u-blocked graph of G. We will use these G_d^u-blocked graphs in our NP-completeness proofs as instances of REC-LEAFAGE-k , REC-SUB-T, and REC-PMD-k.

Let $G = (V, E)$ be a 3-connected graph. Let G' be the disjoint union of six copies of G. For later convenience, we refer to the vertices of the six isomorphic connected components of $G' = (V', E')$

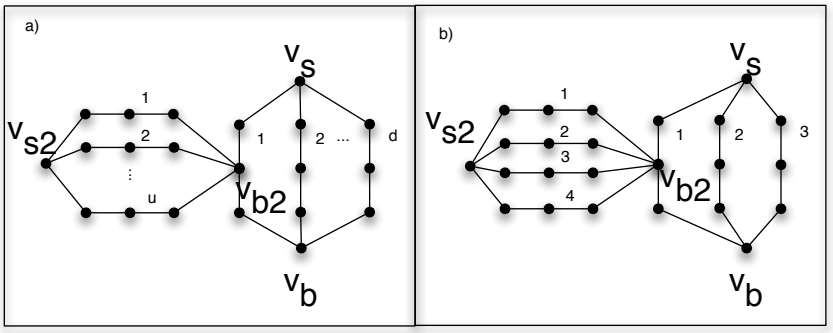

Figure 2.3: In a) the G_d^u graph - note the presence of d paths of three vertices between vertices v_s and v_b, and u paths of three vertices between v_{b2} and v_{s2}. In b) an example: the G_3^4 graph.

as $V_a, V_b, V_c, V_d, V_e, V_f$, or collectively as $V_a ... V_f$.

We describe the production of a graph G'' from G'. Given two natural natural numbers $d \geq 3$ and $u \neq 1$, we define four vertex sets:

1. $V_1 = V'$
2. $V_2 = E'$
3. $V_3 =$ a set of size $|V'|$ disjoint from $V_1 \cup V_2 \cup V_4$
4. $V_4 =$ the vertices of G_d^u (Figure 2.3)

Then let $V'' = V_1 \cup V_2 \cup V_3 \cup V_4$. Let f be a bijection from V_1 to V_3. Then let $E'' = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6$ where:

1. $E_1 = \{(v_i, v_j) | v_j \in E'$ and there exists v_k such that $v_j = (v_i, v_k)\}$
2. $E_2 = \{(v_i, v_j) | v_i, v_j \in V_3 \cup V_2\}$
3. $E_3 = \{(v_i, v_j) | v_j = f(v_i)\}$
4. $E_4 = \{(v_i, v_s) | v_i \in (V_3 \cup V_2)\}$ and v_s is as labelled in Figure 2.3
5. $E_5 = \{(v_i, v_b) | v_i \in (V_3 \cup V_2)\}$ and v_b is as labelled in Figure 2.3
6. $E_6 =$ the edges of G_d^u

Let $G'' = (V'', E'')$. Due to symmetry in the graph, we can label v_s, v_b, v_{s2} and v_{b2} in V_4 as in Figure 2.3. We call G'' the *G_d^u-blocked graph* of G, and G_d^u the *blocker* of G''. For a simple example of this production, see Figure 2.4.

In our NP-completeness reductions, we will take an instance of 3-CON-k-COLOURING in the form of a graph G, and produce its G_d^u-blocked graph G''. We will then use G'' as an instance of REC-LEAFAGE-$k(G)$, REC-SUB-T(G), or REC-PMD-$k(G)$. We will investigate a number of properties of overlap representations of G''.

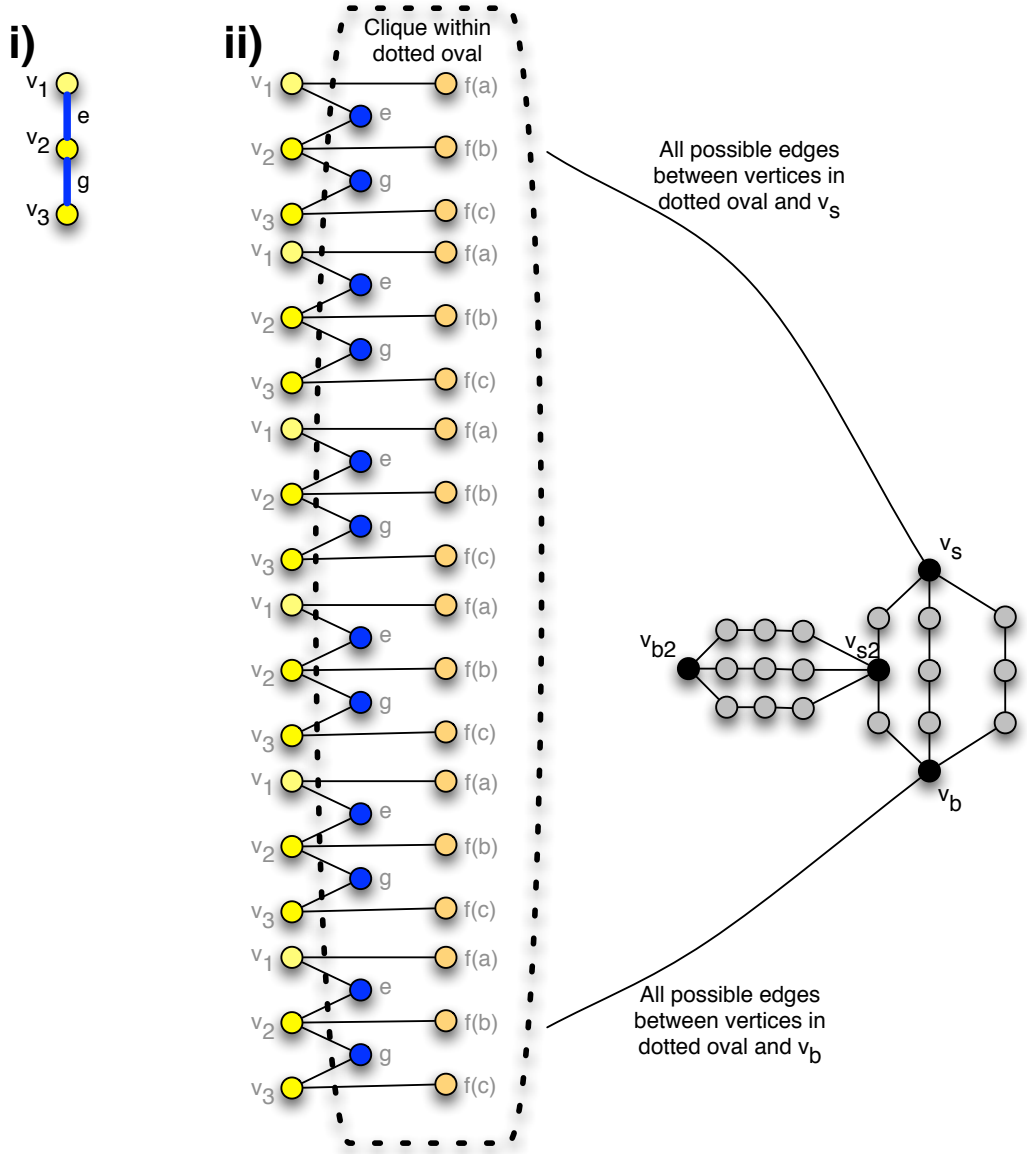

Figure 2.4: A simplified example of the transformation performed on a graph $G = (V, E)$ in i to its G_3^3-blocked graph G'' in ii. The first step of creating G_3^3-blocked graph is to make graph G' that is the union of six disjoint copies of G. Note that the vertex set of the G_3^3-blocked graph on the right consists of the vertex set of the graph on the left, the edge set of the graph on the left, an additional vertex for each vertex in V, and the vertices of G_d^u. The vertices within the dotted oval form a clique. The edges between the dotted oval and v_s, v_b indicate that all vertices within the dotted circle are adjacent to both of v_s, v_b. In this example, in ii), V_1 consists of the vertices in the first column of ii, V_2 the vertices in the second, and V_3 the vertices in the third.

2.7 Representing G_d^u-blocked graphs and colouring

Enright [5] showed that in any subtree overlap representation of G_d^0 where $d \geq 3$ either $t_s \subset t_b$ or $t_b \subset t_s$. Because of the symmetry of G_d^u, we will always assume that $t_s \subset t_b$.

Then in every subtree overlap representation of G_d^u where $d \geq 3$, either $t_s \subset t_b$ or $t_b \subset t_s$, and in every subtree overlap representation of G_d^u in which $u \geq 3$ either $t_{s2} \subset t_{b2}$ or $t_{b2} \subset t_{s2}$. If $u = 0$, then we can effectively disregard t_{s2} as it will be an isolated vertex and can be represented by any subtree containing a single vertex.

Observation 3. *In every subtree overlap representation of G_d^u where $d \geq 3$ and $u \geq 3$, either $t_s \subset t_b$ and $t_{s2} \subset t_{b2} \subset t_b$, and $t_{s2}|t_s$ and $t_{b2}|t_s$, or $t_b \subset t_s$ and $t_{s2} \subset t_{b2} \subset t_s$ and $t_{s2}|t_b$ and $t_{b2}|t_b$.*

Lemma 10. *In every subtree overlap representation $\mathcal{T}$ of G_d^u, where $d \geq 3$, t_s and t_b have at least d boundary nodes and if $u \geq 3$ then t_{s2} and t_{b2} have at least u boundary nodes.*

Proof. Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation of graph G_d^u. We know that either $t_s \subset t_b$ or $t_b \subset t_s$ by Statement 3. Without loss of generality, let $t_s \subset t_b$.

Vertex v_s has d neighbours which induce an independent set in G_d^u. We show by contradiction that the subtrees corresponding to these neighbours are disjoint. Assume that v_i, v_j are in the neighbourhood of v_s, and $t_i \subset t_j$. Then, by Lemma 8, $t_b \subset t_j$, and therefore $t_s \subset t_j$ - a contradiction. Hence all subtrees corresponding to neighbours of v_s are disjoint in $\mathcal{T}$.

Observe that if two subtrees overlap, then each contains a boundary node of the other. Then each subtree overlapping t_s must contain a boundary node of t_s. There are d neighbours of v_s, and therefore at least k subtrees that must overlap t_s, all of which are pairwise disjoint. Then by the pigeonhole principle, t_s has at least d boundary nodes.

The same argument gives us that t_{s2} has at least u boundary nodes.

□ □

It follows that:

Corollary 2. *In every subtree overlap representation of G_3^u where $u \geq$ both t_s and t_b contain a node of degree at least three in T.*

Lemma 11. *In every subtree overlap representation of graph G_d^u where $d \geq 3$ as subtrees $\mathcal{T}$ of tree T, T has at least d leaves.*

Proof. Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation of graph G_d^u. By Lemma 10, t_s and t_b have at least d boundary nodes. As both t_s and t_b are contained in T, then by Lemma 2, T has at least d boundary nodes. As all boundary nodes of T are leaves, T has at least d leaves. □ □

Then by Observation 3, Lemma 10 and Corollary 1 we have:

Lemma 12. *In every subtree overlap representation of G_d^u as subtrees $\mathcal{T}$ of tree T in which $t_s \subset t_b$ if $d \geq 3$ and $u \geq 3$ then $t_s \subset t_b$ has at least d boundary nodes, t_{s2} and t_{b2} have at least u boundary nodes, and T has at least $u + d - 2$ leaves.*

We now use the preliminary lemmas on representations of the G_d^u graph to prove lemmas on the colourability of G given representability of G''.

Definition 3. *Let $G = (V, E)$ be a graph and G'' the G_d^u-blocked graph of G. Let subtrees $\mathcal{T}$ of tree T be an overlap representation of G''. Let $v_i, v_j \in V_1$ be such that $(v_i, v_j) \in E'$. Vertices v_i, v_j are an* illegal pair *if t_i, t_j are contained in the same twig of T.*

Definition 4. *An* illegal subtree *is a subtree corresponding to a vertex in an illegal pair.*

Definition 5. *A* legal neighbour *of an illegal subtree t_j is a subtree t_i such that t_i and t_j are on different twigs of T'', and $(v_i, v_j) \in E'$.*

Lemma 13. *Let $G'' = (V'', E'')$ be the G_d^u-blocked graph of 3-connected graph $G = (V, E)$ with $G', V_a, V_b, V_c, V_d, V_e, V_f$, G_d^u, v_s, v_b as defined previously. Let subtrees $\mathcal{T}''$ of tree T'' be a subtree overlap representation of G''. Then at most one of $V_a, V_b, V_c, V_d, V_e, V_f$ has an illegal pair, and there is at most one illegal pair in each of $V_a, V_b, V_c, V_d, V_e, V_f$.*

Proof. Let v_x, v_y be an illegal pair in V_1, and v_w, v_z another. Assume that the pair t_x, t_y are on a different twig than t_w, t_z are. Let $e_{xy} = (v_x, v_y) \in E'$ and $e_{wz} = (v_w, v_z) \in E'$. Then t_{xy} is contained in one twig and t_{wz} another - they are therefore disjoint, a contradiction. Therefore all illegal pairs must be on the same twig.

Because all illegal subtrees are on the same twig, they are paths contained in a path of T''. In any three separate members of illegal pairs, at least one is disjoint from the interpath of some other illegal pair by Lemma 1. Let t_i be an illegal subtree with a legal neighbour such that t_i is disjoint from the interpath of an illegal pair t_j, t_k. Let t_h be a legal neighbour of t_i. Consider the vertex $v_l = (v_j, v_k) \in E'$ and the vertex $v_m = (v_i, v_h) \in E''$. Then subtree t_l must be contained in the interpath of t_j, t_k. If that path is disjoint from t_i, then t_m is either disjoint from or contains t_l, a contradiction with the construction of G'', where $V_3 \cup V_4$ induce a clique. Any illegal subtree disjoint from the interpath of any illegal pair cannot have a legal neighbour. Because one of every three illegal subtrees is disjoint from the interpath of at least one illegal pair, at most two illegal subtrees can have legal neighbours.

Let t_x, t_y, t_z be illegal subtrees corresponding to vertices in the same one of $V_a, V_b, V_c, V_d, V_e, V_f$. By the previous argument, at most two of them have legal neighbours. Then in G' consider a vertex partition into the vertices that correspond to illegal subtrees in $\mathcal{T}''$, and all other vertices. If only two of the vertices with illegal subtrees have legal neighbours, then those two constitute a two-vertex cutset in G' - a contradiction to the components of G' being 3-connected within each component. If there are at least three illegal vertices in the same one of $V_a, V_b, V_c, V_d, V_e, V_f$, then at least three

illegal subtrees have legal neighbours, a contradiction to the previous argument. Therefore there is at most one illegal pair in each of $V_a, V_b, V_c, V_d, V_e, V_f$.

Assume that two of $V_a, V_b, V_c, V_d, V_e, V_f$ contain illegal pairs. Since each of $V_a, V_b, V_c, V_d, V_e, V_f$ has at least 4 vertices, and there is no vertex of degree one in G', then every illegal subtree has a legal neighbour, but we know this cannot be the case because at most two illegal subtrees have legal neighbours. Therefore at most one $V_a, V_b, V_c, V_d, V_e, V_f$ has an illegal pair. □ □

Lemma 14. *Let $G = (V, E)$ be a 3-connected graph. Let G'' be the G^0_d-blocked graph of G. If G'' is the overlap graph of subtrees $\mathcal{T}$ of a tree T with $k = d$ leaves and a single vertex of degree greater than two, then G is k-colourable.*

Proof. Let $G = (V, E)$, $G'' = (V'', E'')$, T, $\mathcal{T}$, d and k be as described in the lemma statement.

The main idea of our proof is that in any representation by subtrees $\mathcal{T}$ of tree T at least one of $V_a...V_f$ is nicely represented with respect to G'. Since T has k twigs and each of $G'[V_a]...G'[V_f]$ is isomorphic to G, by Observation 2, if any of $V_a...V_f$ are nicely represented with respect to G', then G is k-colourable. If a vertex set V_i is not nicely represented with respect to G' then by definition, there is either a pair of illegal vertices in V_i, or a vertex in V_i that corresponds to a subtree not contained in a twig of T.

We prove a number of properties of any representation of G'', and then derive a colouring of G from this representation.

T has a single node p of degree greater than two - that node has degree d. Let $\mathcal{T}_1$, $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$ be the families of subtrees representing vertices in V_1, V_2, V_3, and V_4, respectively.

By Lemma 12, subtree t_s must have at least d boundary nodes. Since $d \geq 3$ this means that t_s must contain p. Since $t_s \subset t_b$ node p also contained in t_b. Every subtree $t_i \in \mathcal{T}$ that contains p must intersect both t_s and t_b. Since v_i is adjacent to neither v_s nor v_b in G'', t_i must contain or be contained in each of t_s and t_b. Assume that neither $t_i \subset t_s$ nor $t_i \supset t_b$. Then $t_i \supset t_s$ and $t_i \subset t_b$. However, by Lemma 8 if $t_i \supset t_s$ then $t_i \supset t_b$, a contradiction. Therefore every subtree $t_i \in \mathcal{T}_1$ that contains node p is either contained in both t_s and t_b, or contains both t_s and t_b. We show that at most one subtree in $\mathcal{T}_1$ contains t_b and at most one is contained in t_s.

Assume there are two vertices $v_x, v_y \in V_1$ such that $t_x \supset t_b$ and $t_y \supset t_b$. Without loss of generality assume $t_x \subset t_y$. Consider the subtree t_{yh} corresponding to the vertex $v_{yh} = f(v_y)$. Since t_{yh} must overlap t_b and t_y, it overlaps t_x, a contradiction.

Assume there are two vertices $v_x, v_y \in V_1$ such that $t_x \subset t_s$, $t_y \subset t_s$, and both t_x and t_y contain node p. Without loss of generality assume $t_x \subset t_y$. Consider the subtree t_{xh} corresponding to the vertex $v_{kh} = f(t_x)$. Since t_{xh} must overlap t_s and t_x, it therefore overlaps t_y as well, a contradiction. Then, because at most one subtree in $\mathcal{T}_1$ contains t_b and at most one is contained in t_s and we know that every subtree that contains p is either contained in t_s or contains t_b, there are at most two subtrees in $\mathcal{T}_1$ that contain p, and at most two subtrees in $\mathcal{T}_1$ that are not contained in a

twig of T''.

By Lemma 13 at most one of $V_a, V_b, V_c, V_d, V_e, V_f$ contains an illegal pair, and there is a most one illegal pair in each of $V_a, V_b, V_c, V_d, V_e, V_f$. Then there is at most one illegal pair in V_1. We showed that at most two of $V_a, V_b, V_c, V_d, V_e, V_f$ contain vertices that correspond to subtrees that are not contained in twigs of T. Therefore at least three of $V_a ... V_f$ are nicely represented with respect to G'.

Combining this with Observation 2, and the fact that T'' has k leaves, we have that at least three of $G'[V_a], G'[V_b], G'[V_c], G'[V_d], G'[V_e], G'[V_f]$ are k-colourable. Since all of $G'[V_a]$, $G'[V_b]$, $G'[V_c]$, $G'[V_d]$, $G'[V_e]$, $G'[V_f]$ are isomorphic to G, it holds that G is $k-$colourable. □ □

By a proof similar to that for Lemma 14, but somewhat more technically complicated, we have:

Lemma 15. *Let G be a 3-connected graph and $d, u \geq 3$ two natural numbers. If the G_d^u-blocked graph of G is the overlap graph of subtrees of a tree with $k = d + u - 2$ leaves and no lastbranch node of degree greater than two but less than d, then G is $k-$colourable.*

Proof. Let T be a tree with $k = (d + u - 2)$ leaves and no lastbranch node of T with degree greater than two but less than d, such that G'' is the overlap graph of a family of subtrees $\mathcal{T}$ of T. Because of the symmetric nature of G_d^u-blocked graphs, we will assume that $t_s \subset t_b$.

For convenience let $\mathcal{T}_1$ be the subtrees representing vertices in V_1, $\mathcal{T}_2$ be the subtrees representing vertices in V_2, $\mathcal{T}_3$ be the subtrees representing vertices in V_3, and $\mathcal{T}_4$ be the subtrees representing vertices in V_4, where V_1, V_2, V_3, V_4 are as defined in the construction of G_d^u-blocked graphs.

We claim that one node of T of degree d is contained in t_s, all other nodes of degree greater than two are contained in t_{s2} (which is contained in t_{b2} by Observation 3) and therefore by Observation 3 all nodes of T of degree greater than two are contained in t_b. First, we show that one node of degree d is contained in t_s, and that all other nodes of degree greater than two are contained in t_{s2}.

From Lemma 10 we have that t_s must have at least d boundary nodes. Since $d > 2$ this means that t_s must contain at least one node of degree greater than two. Since there are no lastbranch nodes of T of degree greater than two but less than d, t_s contains a node of at least degree d.

By Lemma 6 and Observation 3 if t_s has more than d boundary nodes, t_{b2} has at most $u - 1$ boundary nodes, a contradiction to the construction of G_d^u and Lemma 12. Therefore t_s has exactly d boundary nodes.

We claim that t_s must contain a lastbranch node of T. We justify this by contradiction: assume that t_s does not contain a lastbranch node. Since $t_s | t_{b2}$, by the definition of lastbranch, there must be a lastbranch node x such that the path from x to the closest node of t_s does not intersect t_{b2}. Then consider the subtree t_x formed by the union of t_s and that path including x. By the definition of lastnode, t_x has at least one more boundary node than t_s. That is t_x has at least $d + 1$ boundary nodes. Since t_{b2} has u boundary nodes (Lemma 12), and $t_{b2} | t_x$, then by Corollary 1, T must have at least $(u) + (d + 1) - 2 = k + 1$ leaves, a contradiction. Then t_s contains a lastbranch node.

Since there are no lastbranch nodes of degree greater than two but less than d, it holds that t_s contains exactly one lastbranch node of degree d. If t_s contains any other node of degree greater than two, then it must have at least $d+1$ boundary nodes. However, by $t_s|t_{b2}$, and the fact that t_{b2} has u boundary nodes then T must have at least $d+u+1-1 = d+u$ leaves, a contradiction to the assumed number of leaves of T.

Then, since t_{b2} has $u = k-d+2$ boundary nodes (Lemma 12), by Lemma 5, all nodes of degree greater than two that are not in t_s are in t_{b2}. Since $t_s \subset t_b$ and $t_{b2} \subset t_b$, all nodes of degree greater than two are contained in t_b. Let $t_i \in \mathcal{T}_1$ such that t_i contains a node of degree greater than two. Assume that $t_i \subset t_{b2}$. Consider the subtree $t_j \in \mathcal{T}_3$ such that $v_j = f(v_i)$. $t_j \not\setminus t_i$ and $t_j \not\setminus t_s$. Since $t_s|t_{b2}$ and $t_i \subset t_{b2}$ then $t_j \not\setminus t_{b2}$, a contradiction to the fact that $(v_{b2}, v_i) \notin E''$. Therefore no subtree in $\mathcal{T}_1$ is contained in t_{b2}.

We have established that there is one node of degree $d \geq 3$ in t_s, and all others are in $t_{s2} \subset t_{b2}$. Because $t_i \not\subset t_{b2}$, $t_i|t_{b2}$, so it must be that t_i contains the node of degree d in t_s. Therefore t_i intersects both t_s and t_b. Since v_i is adjacent to neither v_s nor v_b in G'', t_i must contain or be contained in each of t_s and t_b.

Assume that neither $t_i \subset t_s$ nor $t_i \supset t_b$. Then $t_i \supset t_s$ and $t_i \subset t_b$. However, by Lemma 8 if $t_i \supset t_s$ then $t_i \supset t_b$, a contradiction. Then for every subtree $t_i \in \mathcal{T}_1$, if t_i contains a node of T of degree greater than two, either $t_i \subset t_s$ or $t_i \supset t_b$.

Let $V_z \subset V_1$ be the set of vertices in V_1 such that their subtrees in $\mathcal{T}$ are not contained in a twig of T and do not contain a node of degree greater than two in T. Let $\mathcal{T}_z \subset \mathcal{T}_1$ be the subtrees corresponding to these vertices.

Any subtree in $\mathcal{T}_1$ that is not contained in a twig of T and does contain a node of degree greater than two in T is either contained in t_s or contains t_b. We will show that at most one such subtree contains t_b and at most one is contained in t_s.

Assume there are two vertices $v_x, v_y \in V_1$ such that $t_x \supset t_b$ and $t_y \supset t_b$. Without loss of generality assume $t_x \subset t_y$.

Then consider the subtree t_{yh} corresponding to the vertex $v_{yh} = f(v_y)$. Since t_{yh} must overlap t_b and t_y, it therefore overlaps t_x, a contradiction.

Assume that there are two vertices $v_x, v_y \in V_1$ such that $t_x \subset t_s$, $t_y \subset t_s$, and both t_x and t_y contain the node p of degree greater than two in t_s. Without loss of generality assume $t_x \subset t_y$.

Then consider the subtree t_{xh} corresponding to the vertex $v_{xh} = f(t_x)$. Since t_{xh} must overlap t_s and t_x, it therefore overlaps t_y as well, a contradiction.

By Lemma 13 at most one of $V_a, V_b, V_c, V_d, V_e, V_f$ has an illegal pair, and there is at most one illegal pair in each of $V_a, V_b, V_c, V_d, V_e, V_f$. At most three of $V_a \backslash V_z, V_b \backslash V_z, V_c \backslash V_z, V_d \backslash V_z, V_e \backslash V_z, V_f \backslash V_z$ contain vertices that correspond to subtrees that are not contained in twigs of T. That is, at most four of $V_a \backslash V_z, V_b \backslash V_z, V_c \backslash V_z, V_d \backslash V_z, V_e \backslash V_z, V_f \backslash V_z$ are not nicely represented with respect to G'.

Therefore at least two of $V_a \backslash V_z, V_b \backslash V_z, V_c \backslash V_z, V_d \backslash V_z, V_e \backslash V_z, V_f \backslash V_z$ are nicely represented

with respect to G'. Combining this with Observation 2, and the fact that T'' has k leaves, we have that at least two of $G'[V_a \backslash V_z]$, $G'[V_b \backslash V_z]$, $G'[V_c \backslash V_z]$, $G'[V_d \backslash V_z]$, $G'[V_e \backslash V_z]$, $G'[V_f \backslash V_z]$ are $k-$colourable. Let V_x, V_y where $x, y \in \{a, b, c, d, e, f\}$ be the two vertex sets such that $G'[V_x \backslash V_z], G'[V_y \backslash V_z]$ are $k-$colourable. Let $C_1 ... C_k$ be the colour classes of $(V_x \cup V_y) \backslash V_z$ as described in Observation 2 - recall that each colour class corresponds to a twig of T''. Since there are no edges in G' between V_x and V_y we can apply these colour classes to both $V_x \backslash V_z$ and $V_y \backslash V_z$ and still have a correct colouring in $G'[(V_x \cup V_y) \backslash V_z]$.

Now we turn our attention to V_z. Let the union of the $d-1$ twigs of T'' attached to the node of degree d in t_s be designated Q_g. Let the union of the other twigs be designated Q_o. Let $V_g \subset V_1$ be the subset of vertices corresponding to subtrees contained in Q_g. Let $V_o \subset V_1$ be the subset of vertices corresponding to subtrees contained in Q_o.

Let $\mathcal{T}_z$ be the subtrees corresponding to vertices in V_z. Recall that all members of $\mathcal{T}_1$, including those in $\mathcal{T}_z$ are not contained in t_{b2}. Because no member of $\mathcal{T}_z$ contains a node of degree greater than two, no member of $\mathcal{T}_z$ contains t_{b2}. Therefore all members of $\mathcal{T}_z$ are are disjoint from t_{b2}.

Since all nodes that are not on twigs of T'' are contained in t_b, and v_b is adjacent to no member of V_z in G'', it holds that every member of $\mathcal{T}_z$ is contained in t_b. To be not contained in a twig of T'', and not containing a node of degree greater than two, all members of $\mathcal{T}_z$ must be contained in a path between two nodes of degree greater than two. Recall that all nodes of T of degree $d \geq 3$ are contained in $t_s \subset t_b$ or in t_{b2}. Since all members of $\mathcal{T}_z$ are disjoint from t_{b2} and all but one node of degree greater than two is contained in t_{b2}, the only possible such path is between the node of degree d contained in t_s and a node in t_{b2}. Let the segment of this path between the node of degree d and the first node of t_{b2} be called P.

First, we claim that no node of P is contained only in t_b, and not in any other member of $\mathcal{T}_4$. We justify this by contradiction: since t_{b2} and t_s are disjoint, but each intersects opposite ends of P, if there were a node of P contained only in t_b and no other member of $\mathcal{T}_4$, then t_b would be a cutset separating t_s and t_{b2} in $G''[V_4]$ (Lemma 3). This is not the case, hence contradiction.

Therefore every node of P is contained in at least one subtree in $\mathcal{T}_4$ other than t_b. Let $t_i \in \mathcal{T}_z$ be such that $t_i \not\subset t_s$. Since v_i is nonadjacent in G'' to every member of V_4, t_i does not overlap any subtree in $\mathcal{T}_4$.

The subtree t_i does not contain any member of $\mathcal{T}_4$ by Lemma 8, and the fact that $t_i \subset t_b$. Therefore t_i is either disjoint from or contained in every member of $\mathcal{T}_4$. Since we have assumed that $t_i \not\subset t_s$, it must be that $t_i | t_s$. Then, since $t_i \subset P$ and every node of P is contained in a member of $\mathcal{T}_4$ that is not t_b, there exists $t_k \in \mathcal{T}_4$ such that $t_k \neq t_s$, $t_k \neq t_b$ and $t_i \subset t_k$. Now consider the subtree t_j such that $v_j = f(v_i)$. Since $(v_j, v_b) \in E''$ and $(v_j, v_s) \in E''$, t_j must overlap both t_s and t_b.

$t_j \not| t_i$ and $t_i \subset t_k$, but t_j does not overlap t_k so it must be that $t_j \subset t_k$. Since t_j must overlap both t_s and t_b, then t_k must also overlap t_s and t_b. However, there is no vertex in V_4 that is adjacent

in G'' to both v_s and v_b, a contradiction. Therefore all members of $\mathcal{T}_z$ are contained in t_s.

We claim that $G'[V_z]$ is an independent set. We proceed by contradiction. Let $v_i, v_j \in V_z$ be adjacent in G'. Then consider subtree $t_k \in \mathcal{T}_2$ such that $v_k = (v_i, v_j) \in V_2$.

Since t_i, t_j and the path between them contain no nodes of degree greater than two in T'' (because they are contained in P), t_k is a path with its leaves in t_i and t_j. Since $t_i \subset t_s$ and $t_j \subset t_s$, then $t_k \subset t_s$. This is a contradiction to $(v_s, v_k) \in E''$. Therefore $G'[V_z]$ is an independent set.

We claim that at most one member of V_z has neighbours in both V_g and V_o. Let t_i and t_j be two subtrees in $\mathcal{T}_z$.

Note that $t_i | t_j$ for reasons of representing $f(v_i)$ and $f(v_j)$ correctly and because they are non-adjacent in G''.

We now show by contradiction that at most one of v_i, v_j has neighbours in G' in both V_g and V_o. Assume that both v_i and v_j have neighbours in G' in both V_g and V_o. Let :

- v_g be a neighbour of v_i in V_g
- v_h be a neighbour of v_i in V_o
- v_k be a neighbour of v_j in V_g
- v_l be a neighbour of v_j in V_o

Then we consider further the vertices in V_2:

- $v_{g,i} = (v_g, v_i) \in E'$
- $v_{h,l} = (v_h, v_i) \in E'$
- $v_{k,j} = (v_k, v_j) \in E'$
- $v_{l,j} = (v_l, v_j) \in E'$

Note that $v_{g,i}, v_{h,l}, v_{k,j}, v_{l,j}$ induce a clique in G'', as all are members of V_2. This means that by the Helly property $t_{g,i}, t_{h,l}, t_{k,j}, t_{l,j}$ must have a node in common. Let us assume that node is not in t_i. If that node is between t_i and the node of degree d contained in t_s, then $t_{h,l} \supset t_i$, a contradiction. If that node is between t_i and a node of t_{b2} then $t_{g,i} \supset t_i$, a contradiction. Therefore that node must be in t_i. However, symmetrically, it must also be in t_j, a contradiction to $t_i | t_j$. At most one of v_i, v_j has neighbours in G' in both of V_g, V_o. Since v_i, v_j were arbitrary members of V_z, we have that at most one member of V_z has neighbours in both V_g and V_o.

Then, given the derivation of the colour classes $C_1 ... C_k$ from the twigs of T'', at most one member of V_z is adjacent in G' to members of every colour class. All other members of V_z are non-adjacent in G' to all members of at least one colour class. We can then simply extend the colour classes $C_1 ... C_k$ to colour all but at most one member of V_z, maintaining a correct colouring in G'. Since that one member can be in either V_x or V_y but not both, it holds that $C_1 ... C_k$ contains all members of at least one of V_x or V_y.

Therefore at least one of $G'[V_x], G'[V_y]$ is $k-$colourable. Since all of $G'[V_a], G'[V_b]$, $G'[V_c]$, $G'[V_d]$, $G'[V_e]$, $G'[V_f]$ are isomorphic to G, it holds that G is $k-$colourable. □ □

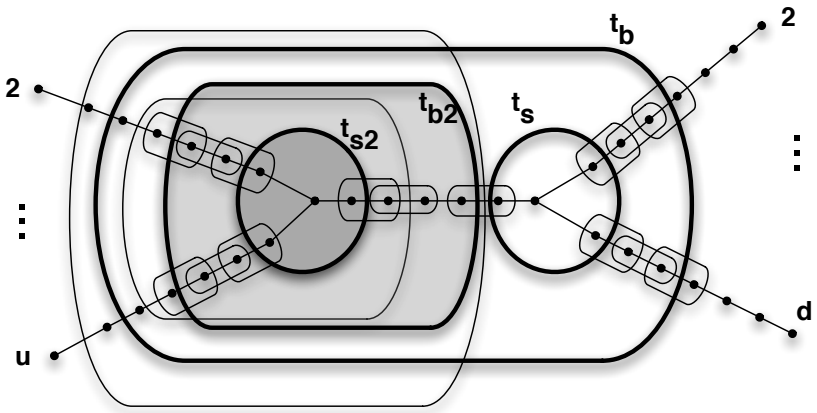

Figure 2.5: A generalised overlap representation of the G_d^u graph on a tree with a node such that the forest created by removing that node has two connected components: a tree with d leaves and a node of degree d and a tree with at least $u+1$ leaves. The interior of t_{b2} and t_{s2} are darkened to indicate that the structure of the tree there is somewhat irrelevant - only the number of boundary nodes is important. In the darkened region could be a single vertex of degree $u+1$ with many leaves, or any other tree with u attached twigs. There is exactly one node of degree greater than two contained in t_s and that node is contained only in t_s and t_b, and all other nodes of degree greater than two are contained in t_{b2}. The representation is on the left, and the G_d^u graph is on the right. Vertex labels and corresponding subtrees are colour coded.

We now move on to showing that a k-colouring of a graph G implies the existence of a representation for the G_d^u-blocked graph of G.

Definition 6. *Let k, d, u be natural number such that either $u = 0$ and $k = d \geq 3$ or $d, u \geq 3$ and $k = d + u - 2$. Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation of G_d^u. We say that $\mathcal{T}$ is a convenient representation of G_d^u if:*

- *T has k leaves*
- *There is a node q with degree d in t_s and t_b*
- *There are no other nodes of degree greater than two in t_s*
- *Node q is not in any subtrees other than t_s and t_b*
- *All other nodes of degree greater than two are in t_{s2} and t_{b2}*
- *$t_s | t_{b2}$*
- *The forest created by removing q from T has at most one non-path component*

Lemma 16. *Let T be a tree and k, d, u be natural numbers such that either $u = 0$ and $k = d \geq 3$ or $d, u \geq 3$ and $k = d + u - 2$.*

- *If $u = 0, k = d \geq 3$ and T has a single node with degree greater than two and has k leaves, then there is a convenient representation of G_d^u on a host tree in SUB(T).*

- *If $d, u \geq 3$ and $k = d + u - 2$ and there is a lastbranch node of T of degree d and T has k leaves, then there is a convenient representation of G_d^u on a host tree in SUB(T).*

Proof. We will proceed in two cases. In the first, let $u = 0, k = d \geq 3$ and T have a single node q with degree greater than two and k leaves. Then let T' be the tree in $SUB(T)$ with a single node q of degree d and no other nodes of degree greater than two, and let there be d paths of minimum five vertices pendant from q.

Then let t_s contain exactly q and all vertices adjacent to q. Let t_b contain exactly q and all vertices within edge-distance three of q. Associate each of the d paths pendant from q with one of the d paths between t_s and t_b. For each of the d paths between t_s and t_b let the first vertex (adjacent to t_s) be assigned a subtree consisting of the first two vertices (from q) on the associated path pendant from q. Let the second vertex on the path be assigned a subtree consisting of the second and third vertex on the associated path. Let the third vertex (the one adjacent to t_b) be assigned a subtree consisting of the third and fourth vertices on the associated path. These subtrees are a convenient representation of G_d^u.

We proceed to the second case. Let $d, u \geq 3$ and $k = d + u - 2$ and let q be a lastbranch node of T degree d and let T have k leaves. Let T be a subdivision of T.

Because paths can be lengthened by subdivision, we assume that every path in T' is as long as we require.

Let t_s contain q and all its neighbours. Let t_{s_2} contain all other nodes of degree greater than two and all their neighbours. Then let all the rest of the subtrees be as in Figure 2.5.

This representation satisfies the requirements, and is convenient.

□

Lemma 17. *Let G be a 3-connected graph, k, d, u be natural numbers such that $k \geq d + u$, and $G'' = (V'', E'')$ the G_d^u-blocked graph of G. Let T be a tree with $k \geq 3$ leaves.*

- ***If** G is k-colourable and G_d^u can be represented on a tree T' in SUB(T) such that the representation is convenient,*

- ***then** G'' can be represented on a tree in $SUB(T)$ such that there is exactly one node of degree greater than two contained in t_s and that node is contained only in t_s and t_b, and all other nodes of degree greater than two are contained in t_{b2}.*

Proof. Let G, G'', T be as described in the lemma statement, and let $C_1 ... C_k$ be a k-colouring of G.

Let subtrees $\mathcal{T}_4$ of tree $T' \in SUB(T)$ be a convenient representation of G_d^u.

We define several sets of nodes of T'. By Definition 6 there is a vertex q of T' such that at most one connected component of the forest $T' \backslash \{q\}$ is not a path. Let N_1 be the set of nodes of that non-path component - this will include all nodes of T_2. If all components are paths, then let N_1 be

an arbitrary component of $T' \backslash \{q\}$. Let $N_2 ... N_d$ be the nodes of the other components. We will also refer to the nodes of N_1 as the *organ nodes* of T' and the subtree they induce as the *organ subtree*.

We will create a tree T'' from tree T' by adding paths to the leaves of T'. Let the leaves of T' be labeled $l_1 ... l_k$

Let $\mathcal{W}$ be a set of k paths. Let each $W_i \in \mathcal{W}$ be length $2(|C_i|) - 1$, and the nodes of W_i be labeled as $r^i_1 ... r^i_{2(|C_i|)}$.

We then create T'' by adding an edge between $r^i_{2(|C_i|)}$ of path W_i to leaf l_i.

Having described tree T'', we now construct the family of subtrees $\mathcal{T}''$ such that G'' is the overlap graph of $\mathcal{T}''$.

We will construct $\mathcal{T}''$ in four parts: $\mathcal{T}_1$ will represent V_1, $\mathcal{T}_2$ will represent V_2, $\mathcal{T}_3$ will represent V_3, and $\mathcal{T}_4$ will represent V_4, where V_1, V_2, V_3, V_4 are as described in the section on constructing the G^u_d-blocked graph of G. The subtrees in $\mathcal{T}_4$ are already given.

Let $v_i \in V_1$ be the i^{th} vertex in colour class C_j. Then t_i is the subpath of W_j induced by nodes r^j_{2i-1}, r^j_{2i} of W. For later convenience, we refer to r^j_{2i-1} as $Up(t_i)$ and r^j_{2i} as $Down(t_i)$. Let subtree family $\mathcal{T}_1$ be the subtrees defined this way for all vertices in V_1. Note that all subtrees in $\mathcal{T}_1$ are pairwise disjoint.

We now turn to the construction of $\mathcal{T}_2 \cup \mathcal{T}_3$. We do this in two parts: first we describe an intermediate form of the subtrees in these subtree families, and then we alter the subtrees to their final form to ensure that they all pairwise overlap.

Let v_i be a vertex in V_2. There are two vertices v_h, v_j in V_1 such that $v_i = (v_h, v_j)$. If the path between $Down(t_h)$ and $Down(t_j)$ contains an organ node of T'', let t_i be the union of that path and the organ subtree of T''. If the path between $Down(t_h)$ and $Down(t_j)$ does not contain an organ node of T, let t_i be that path. Let subtree family $\mathcal{T}_2$ be the subtrees defined this way for all vertices in V_2.

Let v_i be a vertex in V_3. There is exactly one subtree $v_j \in V_1$ such that $v_i = f(v_j)$. If the path between $Down(t_j)$ and q contains an organ node of T', let t_i be the union of that path and the organ subtree of T'. If the path between $Down(t_j)$ and q does not contain an organ node of T', let t_i be that path. Let subtree family $\mathcal{T}_3$ be the subtrees defined this way for all vertices in V_3. Observe that each subtree in $\mathcal{T}_2 \cup \mathcal{T}_3$ contains at most two neighbours of p.

For this to be a valid representation, we need all subtrees in $\mathcal{T}_2 \cup \mathcal{T}_3$ to pairwise overlap. This is a problem, as they may not at the moment. However, because each subtree in $\mathcal{T}_2 \cup \mathcal{T}_3$ contains at most two neighbours of p, and no subtree in $\mathcal{T} \backslash \mathcal{T}_2 \cup \mathcal{T}_3$ has p as a boundary node, we can apply Lemma 9.

Then there exists a subtree overlap representation on a tree that can be derived from T by subdivision of edges such that all there is a one-to-one correspondence between subtrees in this new representation and the subtrees in $\mathcal{T}$ and two subtrees in the new representation have the same relationship as their corresponding subtrees in $\mathcal{T}$ unless they are both in $\mathcal{T}_2 \cup \mathcal{T}_3$, in which case they

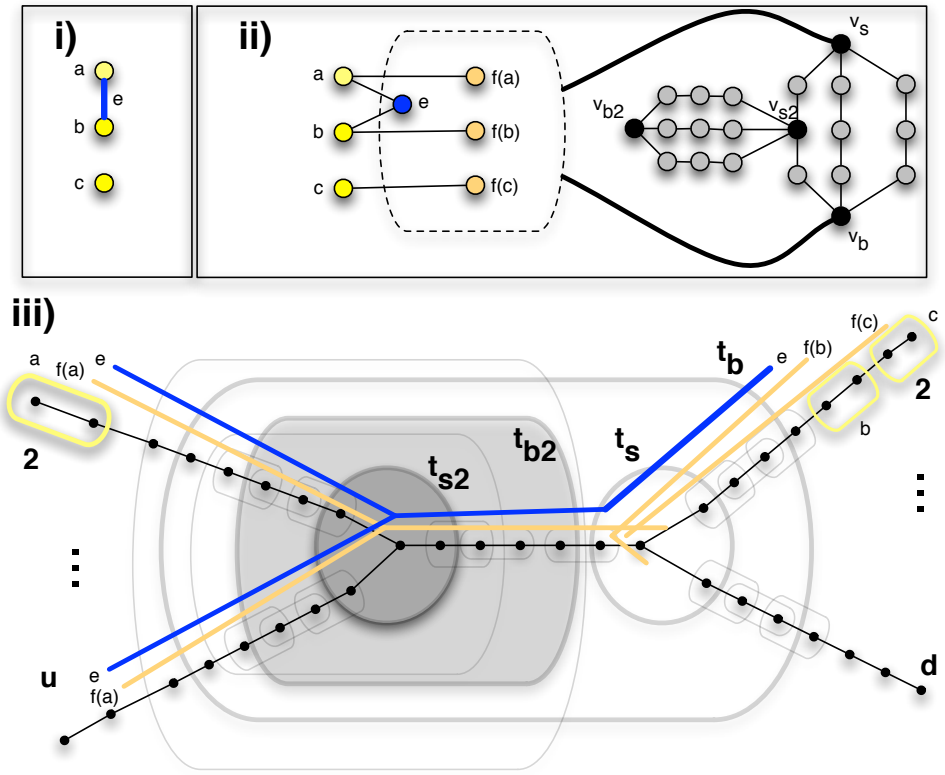

Figure 2.6: In i) a part of a graph, in ii) the corresponding part of that graph in a G_3^3-blocked graph and in iii) a representation of the vertices shown in ii) on a tree as described in Lemma 17. Note that the subtrees corresponding to vertices in ii) that were vertices in i) (that is, a, b, c) are represented on the twigs of the tree. The subtree corresponding to an edge (e) and the subtrees corresponding to $f(a), f(b), f(c)$ are indicated using lines instead of ovals, and all pairwise overlap.

overlap.

Therefore, without loss of generality, we will assume that all subtrees in $\mathcal{T}_2 \cup \mathcal{T}_3$ pairwise overlap.

Let $\mathcal{T}$ be $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3 \cup \mathcal{T}_4$. For a quite intuition on the placement of these subtrees, see Figure 2.6, which shows a very small example.

It remains to show that G'' is the overlap graph of $\mathcal{T}$.

We will show this by proving that for every two vertices $v_i, v_j \in V''$ it holds that $t_i \between t_j$ if and only if v_i is adjacent to v_j.

Assume that $t_i \between t_j$. Because all subtrees in $\mathcal{T}_1$ are disjoint from each other and from the subtrees of $\mathcal{T}_4$ we have the following cases:

1. $v_i, v_j \in V_2 \cup V_3$
2. $v_i, v_j \in V_4$
3. $v_i \in V_1, v_j \in V_2$

4. $v_i \in V_1, v_j \in V_3$

5. $v_i \in V_2 \cup V_3, v_j \in V_4$

Case 1: Let $v_i, v_j \in V_2 \cup V_3$. Then $(v_i, v_j) \in E_4 \subset E''$.

Case 2: Let $v_i, v_j \in V_4$. These are correct by the assumption that G_d^u is the overlap graph of $\mathcal{T}'$.

Case 3: Let $v_i \in V_1, v_j \in V_2$. Since t_i is a path, and none of its nodes have degree greater than two, it holds that if $t_i \between t_j$, then a leaf of t_i must be contained in t_i. By the construction of $\mathcal{T}_2$, then there exists a vertex $v_k \in V_1$ such that $v_j = (v_i, v_k)$. Since such a v_k exists, then $(v_i, v_j) \in E_2 \subset E''$.

Case 4: Let $v_i \in V_1, v_j \in V_3$. Since t_i is a path, and none of its vertices have degree greater than two, it holds that if $t_i \between t_j$, then a leaf of t_i must be contained in t_j. By the construction of $\mathcal{T}_3$, $v_j = f(v_i)$. Then $(v_i, v_j) \in E_3 \subset E''$.

Case 5: $v_i \in (V_2 \cup V_3), v_j \in V_4$. Here we need to consider two subcases: either v_j is v_s or v_b or it is not. In the former case $(v_i, v_j) \notin E''$, in the latter case, $(v_i, v_j) \in E''$. If v_j is in V_4 and is not one of v_s, v_b, then t_j is a subtree of exactly one of $N_1 ... N_d$. If v_j is one of v_s or v_b, then t_j contains nodes of at least three of $N_1 ... N_d$. Because t_i contains all nodes of exactly two of $N_1 ... N_d$ and no nodes of any others, t_i therefore overlaps t_j if and only if t_j is one of v_s, v_b.

This proves that if two subtrees overlap, their corresponding vertices are adjacent. We now prove the converse.

Assume that v_i is adjacent to v_j. By the construction of G'' we know that at least one of v_i, v_j is not in V_1. We then have the following cases:

1. $v_i, v_j \in V_2 \cup V_3$

2. $v_i, v_j \in V_4$

3. $v_i \in V_1, v_j \in V_2$

4. $v_i \in V_1, v_j \in V_3$

5. $v_i \in V_2 \cup V_3, v_j \in \{v_s, v_b\} \subset V_4$

Case 1: Let $v_i, v_j \in V_2 \cup V_3$. We earlier observed that $t_i \between t_j$.

Case 2: Let $v_i, v_j \in V_4$. These are correct by the assumption that G_d^u is the overlap graph of $\mathcal{T}'$.

Case 3: Let $v_i \in V_1, v_j \in V_2$. Then there exists a vertex $v_k \in V'$ such that $v_j = (v_i, v_k)$. Then by the construction of the subtrees $t_i \between t_j$.

Case 4: Let $v_i \in V_1, v_j \in V_3$. Then $v_j = f(v_i)$ (and so $(v_i, v_j) \in E''$), by the construction $Down(t_i)$ is a leaf of t_j. Therefore $t_i \between t_j$.

Case 5: $v_i \in (V_2 \cup V_3), v_j \in V_4$. We know that t_i contains a node also in a subtree corresponding to a member of V_1, and therefore that node is outside both t_b and t_s. Since t_i also contains nodes inside both t_s and t_b, $t_i \between t_b$, and $t_i \between t_s$. Therefore $t_i \between t_j$.

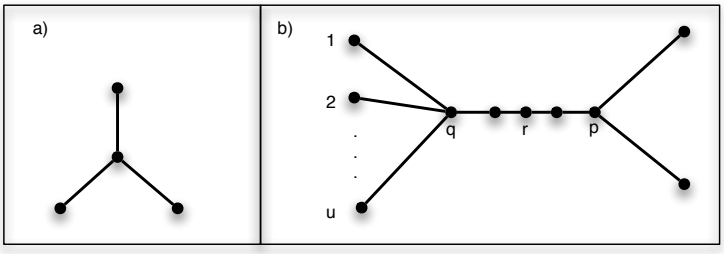

Figure 2.7: Examples of the trees required if a) $k = d = 3$ and $u = 0$ or b) $d = 3$ and $u = k - d + 2$ in the reduction from 3-CON-k-COLOURING to REC-T-k(G), depending on the value of k.

This concludes the proof that G'' is the overlap graph of $\mathcal{T}''$, and therefore that G is k-colourable and G_d^u can be represented on a tree T' in SUB(T) such that the representation is convenient then G'' can be represented on a tree in $SUB(T)$. □

□

2.8 Reductions using nicely represented G_d^u-blocked graphs

We can now show that both REC-LEAFAGE-k and REC-SUB-T are NP-complete.

We start with REC-LEAFAGE-k , proving a reduction from an arbitrary instance of 3-CON-k-COLOURING with at least four vertices to an instance of REC-LEAFAGE-k.

Let G be a 3-connected graph with at least 4 vertices, and $k \geq 3$ a fixed natural number. Let $d = 3$, and if $k = 3$ then $u = 0$, otherwise $u = k - d + 2$. Note that $d \geq 3$ and either $u = 0$ or $u \geq 3$. Let G'' be the G_d^u-blocked graph of G. We show that the answer to 3-CON-k-COLOURING on G is exactly the answer to REC-LEAFAGE-k on G''.

Let T be a tree with k leaves. The form of T depends on u: if $u = 0$ then let T have a single node of degree three, and no other nodes of degree greater than two, as shown in Figure 2.7a. If $u \geq 3$ then let T have exactly two nodes with degree greater than two: the node p with degree 3, and the node q with degree u as shown in Figure 2.7b. This tree has a node r (Figure 2.7) such that the forest created by removing r from T has two connected components: a tree with d leaves and a node of degree d, and a tree with u leaves. Then by Lemma 17 if G is k-colourable, then the graph G'' can be represented on a tree in $SUB(T)$. Since all trees in $SUB(T)$ have k leaves:

Statement 1. *If G is k-colourable, G'' can be represented on a tree with k leaves.*

Recall that $d = 3$ and u equals either 0 (only if $d = k = 3$) or $k - d + 2$. Since there is no node with degree greater than two but less than three, we have from Lemma 15 that:

Statement 2. *If G'' can be represented on a tree with k leaves, then G is k-colourable.*

Combining Statements 1 and 2, and the facts that k-colouring a 3-connected graph is NP-complete and REC-LEAFAGE-k is in NP:

Theorem 4. *REC-LEAFAGE-k is NP-complete.*

We now move on to REC-SUB-T, proving a reduction from an arbitrary instance of 3-CON-k-COLOURING with at least four vertices to an instance of REC-SUB-T for an arbitrary tree T with more than two leaves.

Let T be a fixed tree with $k \geq 3$ leaves and G be a 3-connected graph with at least four vertices. Let q be a lastbranch of T with smallest degree d.

If $d = k$, then let the integer $u = 0$, otherwise let $u = k - d + 2$. Note that either $u = 0$ or $u \geq 3$. Let G'' be the G_d^u-blocked graph of G.

By Lemma 16, in either case there is a convenient representation of G_d^u on a subdivision of T.

Then by Lemma 17:

Statement 3. *If G is k-colourable, G'' can be represented on a tree in $SUB(T)$.*

By the computation of d, T has no lastbranch node of degree greater than two but less than d. Any tree that can be derived from T by subdivision also has no node of degree greater than two but less than d, and still has k leaves. Then by Lemma 15 if G'' is the overlap graph of any family of subtrees of any tree in $SUB(T)$, it holds that G is k-colourable, and therefore:

Statement 4. *If G'' can be represented on a tree in $SUB(T)$, then G is k-colourable.*

Combining Statements 3 and 4, and the facts that k-colouring a 3-connected graph is NP-complete and REC-SUB-T is in NP, we have:

Theorem 5. *REC-SUB-T is NP-complete.*

2.9 Paths in a Tree of Fixed Maximum Degree

We now turn to the complexity of REC-PMD-k.

First consider a polynomial size certificate - a family of paths of a polynomially-sized tree with maximum degree k. Čenek [2] showed that if there is such a representation, there is one that is polynomial-sized in the size of the graph. Since there are n subtrees, we can check to see if G is the overlap graph of $\mathcal{T}$ in polynomial time. Hence the problem is in NP.

We now proceed to show hardness. We describe a reduction from 3-CON-k-COLOURING.

Let $G = (V, E)$ be a 3-connected graph with at least four vertices, and let $k \geq 3$ be a fixed integer. We will produce a graph $G'' = (V'', E'')$ such that G'' is the overlap graph of paths in a tree with maximum degree k if and only if G is k-colourable.

First, we produce the graph $G' = (V', E')$ such that G' is the disjoint union of three copies of G. For later convenience, we will refer to the vertices of the three connected components of G' (each of which is isomorphic to G) as V_a, V_b, V_c.

We now define vertex sets:

- $V_1 = V'$
- $V_2 = E'$
- $V_3 =$ a set of vertices of size $|V_1|$ such that there exists a bijection f from V_1 to V_3

Let $V'' = V_1 \cup V_2 \cup V_3$.

We now define edge sets:

- $E_1 = (v_i, v_j)$ where $v_i \in V_1, v_j \in V_2$ and there exists $v_k \in V_1$ such that $v_j = (v_i, v_k) \in E'$
- $E_2 = (v_i, v_j)$ where $v_i \in V_1$ and $v_j \in V_3$ and $v_j = f(v_i)$
- $E_3 = (v_i, v_j)$ where $v_i, v_j \in (V_2 \cup V_3)$

Let $E'' = E_1 \cup E_2 \cup E_3$. Let $G'' = (V'', E'')$.

For an example of this construction, see Figure 2.8.

Lemma 18. *If G is k-colourable, then G'' is the overlap graph of subpaths of a tree with maximum degree k such that no subpath contains any other.*

Proof. Assume G is k-colourable. Then G' is k-colourable. We construct subpaths $\mathcal{T}$ of tree T with maximum degree k such that G'' is the overlap graph of $\mathcal{T}$.

Let $C_1...C_k$ be the colour classes of a k-colouring of G, and therefore of G'. We will define a tree R_i of maximum degree 3 for each colour class C_i, and then compose these together to produce a tree of maximum degree k.

Consider a colour class C_i. Let R_i be a long path of length $|C_i| - 1$ with a path of 3 nodes for each node p of that long path such that one of the leaves of the 3-node path is adjacent to p. Let the 3-vertex shorter paths be labelled $r_1...r_{|C_i|}$ for convenience. Figure 2.9 provides an illustration.

Then let q be a node adjacent to one end node of the long path in each R_i. Figure 2.10 provides an illustration. Let this constructed tree be T. Note that T has maximum degree k.

We then define several classes of subpaths, representing the vertex sets V_1, V_2, V_3. First we define $\mathcal{T}_1$ to represent the vertices in V_1.

Note that each vertex in V_1 is in one of the colour classes $C_1...C_k$. Let $C_i \in \{C_1...C_k\}$ be in some arbitrary order, such that we can refer to the vertices within it as $v_1...v_{|C_i|}$. Then for vertex $v_j \in C_i$, we define the subtree t_j as the three-vertex path r_j in R_i. For each path t_j, we also label its nodes $t_{j_1}, t_{j_2}, t_{j_3}$ from leaf of T to non-leaf of T. Let $\mathcal{T}_1$ be these subpaths corresponding to vertices in V_1.

We now turn our attention to producing subpaths $\mathcal{T}_2$ that correspond to the vertices in V_2. Let $v_i \in V_2$. Then there exist $v_j, v_k \in V_1$ such that $v_i = (v_j, v_k) \in E'$. Then we define subtree t_i as

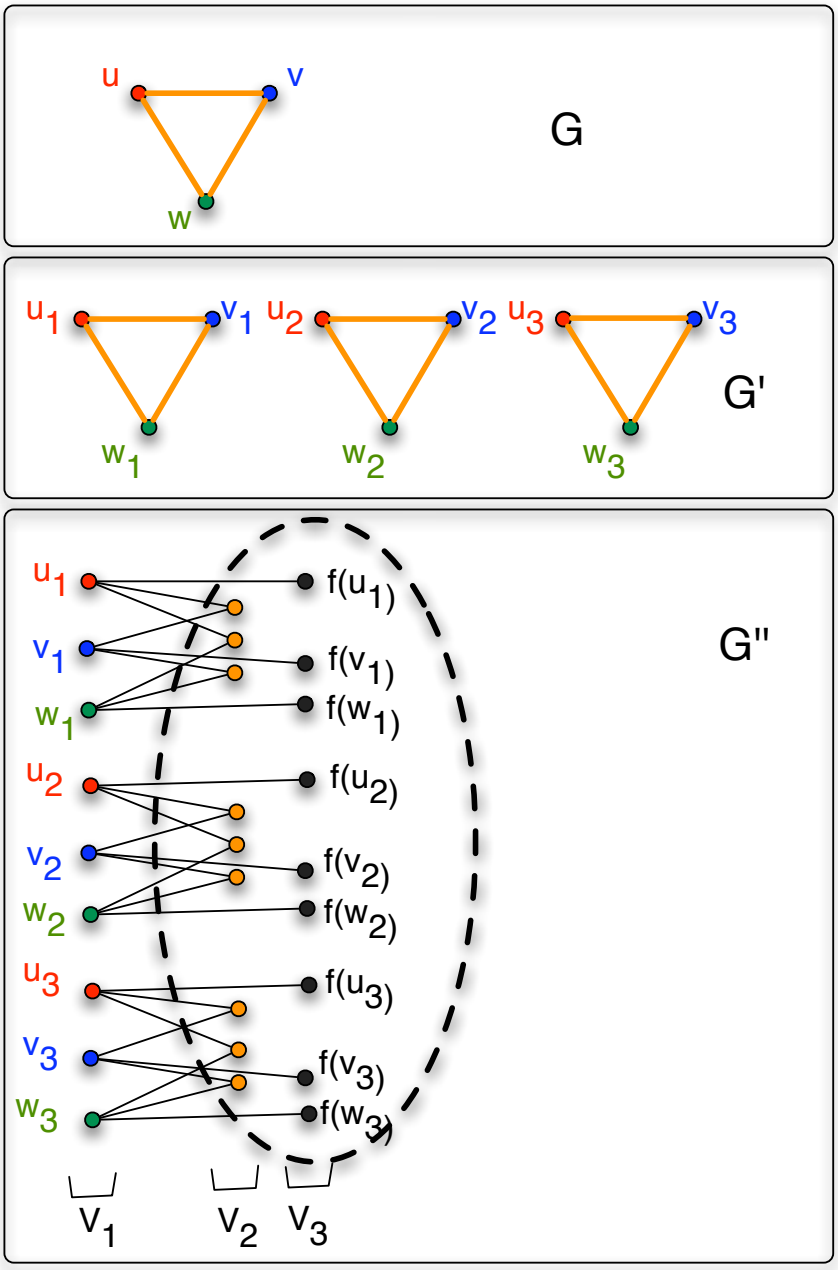

Figure 2.8: An example of the construction of graphs G' and G'' from the graph G. G' is simply the disjoint union of three copies of G. G'' consists of three vertex sets, V_1, V_2, V_3 as outlined in the text. V_1 is the vertex set of G', V_2 is the edge set of set of G', and V_3 is a new set of vertices of the same size as V_1. Please note that the vertices inside the dashed oval should induce a clique - these edges were omitted for legibility.

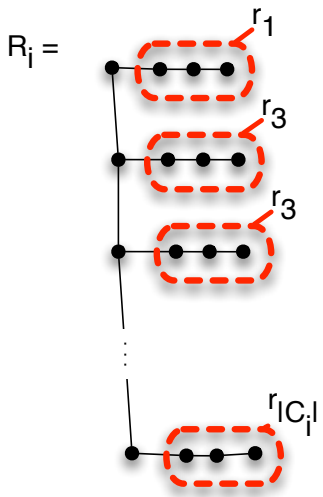

Figure 2.9: Caterpillar R_i associated with colour class C_i with labeled short paths r_1 to $r_{|C_i|}$

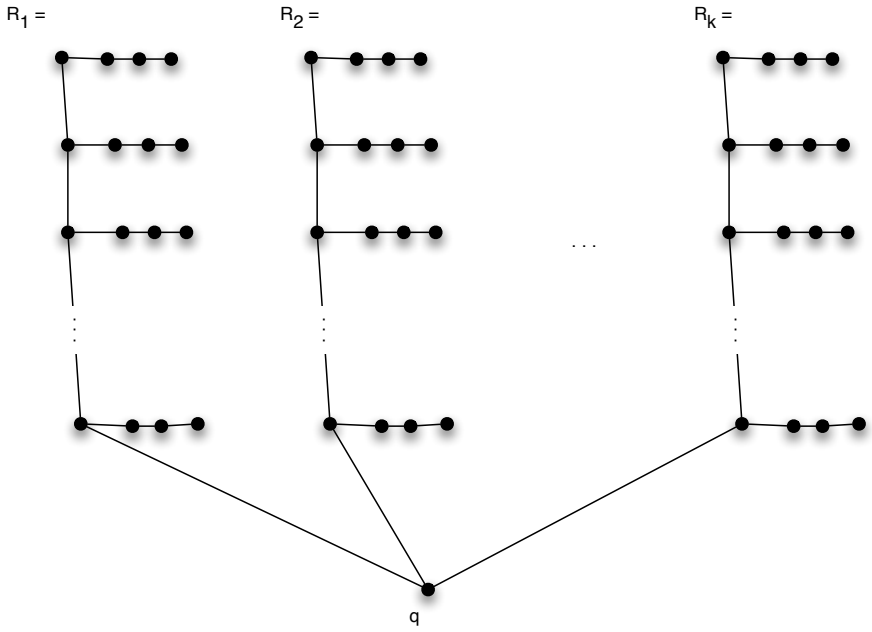

Figure 2.10: An illustration of the construction of tree T to represent the k-colourable graph G. Note that q has degree k, and all other nodes have degree of three or less.

the path in T with endpoints t_{j_3} and t_{k_3}. Let $\mathcal{T}_2$ be the family of these subpaths corresponding to vertices in V_2.

Finally, we produce the subpaths $\mathcal{T}_3$ corresponding to the vertices in V_3. Let $v_i \in V_3$. Then there exists $v_j \in V_1$ such that $v_i = f(v_j)$. Then we define subpath t_i as the path of T connecting the nodes t_{j_2} and q, inclusive. Let $\mathcal{T}_3$ be the family of these subpaths corresponding to vertices in V_3.

Observation 4. *Observe that by this construction, for every pair of subtrees t_i, t_j such that $t_i \in \mathcal{T}_1$ and $t_j \in \mathcal{T}_2 \cup \mathcal{T}_3$, either $t_i \between t_j$ or $t_i | t_j$ - that is, neither contains the other.*

Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3$.

Having defined T and $\mathcal{T}$, it remains to prove that G'' is the overlap graph of $\mathcal{T}$. By the construction, we have a subpath corresponding to each vertex in V''. We show that for two vertices $v_i, v_j \in V''$, $t_i \between t_j$ if and only if $(t_i, t_j) \in E''$.

First, assume that $t_i \between t_j$. Then we consider cases on v_i, v_j. Observe that it is not the case that $v_i, v_j \in V_1$, as by the construction all subtrees in $\mathcal{T}_1$ are pairwise disjoint. Then we are left with the cases:

1. Case 1: $v_i \in V_1$, $v_j \in V_2$. Then let C_k be the colour class that v_i is in. Then t_i is on R_k, and in particular let r_h be the three-vertex path of R_k that t_i is contained in. Then t_j must contain some nodes of r_h. By the construction of $\mathcal{T}_2$, it must be that t_j is an edge in E' such that v_i is one of its endpoints. Hence $(v_i, v_j) \in E''$

2. Case 2: $v_i \in V_1$, $v_j \in V_3$. Then let C_k be the colour class that v_i is in. Then t_i is on R_k, and in particular let r_h be the three-vertex path of R_k that t_i is contained in. Then t_j must contain some nodes of r_h. By the construction of $\mathcal{T}_3$, it must be that $v_j = f(v_i)$, and therefore $(v_i, v_j) \in E''$.

3. Case 3: $v_i, v_j \in (V_2 \cup V_3)$. Then $(v_i, v_j) \in E''$.

Now, assume that $(v_i, v_j) \in E''$. We consider cases on v_i, v_i to show that $t_i \between t_j$.

1. Case 1: $v_i \in V_1$, $v_j \in V_2$. By the construction of E'', it must be that v_j is an edge in E' such that v_i is one of its endpoints. Then by the construction of $\mathcal{T}$, $t_i \between t_j$.

2. Case 2: $v_i \in V_1$, $v_j \in V_3$. By the construction of E'', it must be that $v_j = f(v_i)$. Then by the construction of $\mathcal{T}$, $t_i \between t_j$.

3. Case 3: $v_i, v_j \in (V_2 \cup V_3)$. First, observe that all members of $\mathcal{T}_2 \cup \mathcal{T}_3$ contain q. Therefore it remains to show that neither $t_i \subset t_j$ not $t_j \subset t_i$. If v_i, v_j are both in V_2, then, as their endpoints in E' cannot be the same, they each overlap members of $\mathcal{T}_1$ from which the other is disjoint (Observation 4), therefore they overlap.

   Similarly, if they are both from V_3, as it cannot be that there exists $v_k \in V_1$ such that both $v_i = f(v_k)$ and $v_j = f(v_k)$ (because f is one-to-one), then they each overlap members of $\mathcal{T}_1$

from which the other is disjoint (Observation 4), and therefore overlap. If one is from V_2 and one is from V_3, and it is not the case that there exists $v_k \in V_1$ such that both $v_i = f(v_k)$, and v_k is an endpoint of v_j in E', then they each overlap members of $\mathcal{T}_1$ from which the other is disjoint (Observation 4), and therefore overlap.

If one is from V_2 and one is from V_3, and there exists $v_k \in V_1$ such that both $v_i = f(v_k)$, and v_k is an endpoint of v_j in E', then by the construction of $\mathcal{T}$ t_{k_2} is contained in v_i but not v_j, and t_j overlaps some member of $\mathcal{T}_1$ from which t_i is disjoint (Observation 4), therefore $t_i \not\between t_j$.

This concludes the proof that G'' is the overlap graph of $\mathcal{T}''$, and therefore that if G is $k-$colourable, then G'' is the overlap graph of subpaths of a tree with maximum degree k.

□

Let subpaths $\mathcal{T}$ of tree T with maximum degree k be an overlap representation of G''. Since $V_2 \cup V_3$ induce a clique, by the Helly property there is a node q that is in every subpath corresponding to a member of $V_2 \cup V_3$.

Node q has at most degree k. Consider the connected components $C_1....C_h$ of T created by removing q from T. Certainly $h \leq k$.

Lemma 19. *Let $V_z \subset V_1$. Let $\mathcal{T}_z \subset \mathcal{T}$ be the subpaths representing the vertices in V_z. If no subtree in $\mathcal{T}_z$ contains q, then $G'[V_z]$ is k-colourable.*

Proof. If no member of $\mathcal{T}_z$ contains q, then every member of $\mathcal{T}_z$ is contained in exactly one of $C_1....C_h$. If there are no two subpaths corresponding to adjacent vertices that are contained in the same one of $C_1...C_h$, then $C_1...C_h$ can be used as colour classes.

We show by contradiction that if $t_i, t_j \in \mathcal{T}_z$ and $(v_i, v_j) \in E'$, then t_i, t_j are not contained in the same one of $C_1....C_h$.

Assume that $t_i, t_j \in \mathcal{T}_z$ and $(v_i, v_j) \in E'$, and t_i, t_j are both contained in component $C_l \in \{C_1....C_h\}$. Since $(v_i, v_j) \in E'$, there exists a vertex $v_k = (v_i, v_j) \in V_2$.

By the definition of q, the subtree t_k contains q. Since v_k is adjacent in G'' to both v_i and v_j, the subtree t_k overlaps both t_i and t_j. Recall that t_k is a path.

Without loss of generality assume that the distance along t_k between q and t_i is greater than the distance along t_k between q and t_j.

Because $t_k \not\between t_j$, every path that intersects both t_i and q overlaps t_j.

Then consider the subtree t_h corresponding to the vertex $f(v_i)$. Since it overlaps t_i and contains q, it must also overlap t_j - a contradiction to the fact that $f(v_i)$ is not adjacent to v_j in G''. □ □

If one of V_a, V_b, or V_c is k-colourable in G', that implies a k-colouring of the vertices of G. Therefore it suffices to show that at least one of V_a, V_b, V_c is represented by a set of subpaths in $\mathcal{T}$

such that none of those subpaths contains q. From the previous Lemma, this implies a k-colouring of that member of $\{V_a, V_b, V_c\}$.

Lemma 20. *At least one of V_a, V_b, V_c, is represented by a set of subpaths in $\mathcal{T}$ such that none of those subpaths contains q.*

Proof. Assume that every one of V_a, V_b, V_c contains a vertex for which the subpath contains q. Then there are are vertices v_i, v_j, v_k such that t_i, t_j, t_k all contain q. Since v_i, v_j, v_k are an independent set in G'', all are in pairwise containment relationships. Without loss of generality, let $t_i \subset t_j \subset t_k$. Consider the vertex $v'_i \in V_3$ such that $v'_i = f(v_i)$. Since $t'_i \between t_i$, but t'_i must not overlap t_j, it holds that $t'_i \subset t_j$.

Consider $v'_k \in V_3$ such that $v'_k = f(v_k)$. Then, since t'_k contains q, and overlaps t'_i, but does not overlap t_j, it holds that $t'_k \subset t_j$, and then $t'_k \subset t_k$ - a contradiction with the fact that $(v_k, v'_k) \in E''$. □

Then, since $G'[V_a], G'[V_b], G'[V_c]$ are isomorphic to G, by Lemma 19:

Lemma 21. *If G'' is the overlap graph of subpaths of a tree with maximum degree k, then G is k-colourable.*

Finally, as we have shown that REC-PMD-$k(G)$ is in NP, and that for $k \geq 3$, G is k-colourable if and only if the answer to REC-PMD-$k(G'')$ is yes, then we have:

Theorem 6. *REC-PMD-$k(G)$ is NP-complete.*

Observe that the proof of Lemma 21 is based on the Helly property of subtree -in fact, the proof works equally well if for every occurrence of the word "overlap" we substitute the word "intersect". Then, because the representation guaranteed by Lemma 18 has no containment, and is therefore also an intersection representation of G'', we have that:

Theorem 7. *Recognising the intersection graphs of paths in a tree with maximum degree $k \geq 3$ is NP-complete.*

2.10 Conclusion and Future Work

We have shown that recognising a number of subclasses of subtree overlap graphs is NP-complete. This is surprising because of the enforced simplicity of the representations. Our ultimate goal continues to be resolving complexity of the recognition problem for subtree overlap graphs in general. This is currently an open problem.

Other related open problems include tighter bounds on the leafage of subtree overlap graphs, as well as investigation of other geometric overlap and intersection classes.

Bibliography

[1] Kellogg S. Booth and George S. Lueker. Linear algorithms to recognize interval graphs and test for the consecutive ones property. In *STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing*, pages 255–265, New York, NY, USA, 1975. ACM Press.

[2] Eowyn Čenek. Subtree overlap graphs and the maximum independent set problem. Master's thesis, University of Alberta, Department of Computing Science, 1998.

[3] Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Lbfs orderings and cocomparability graphs. In *SODA '99: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 883–884, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[4] Cornelia Dangelmayr and Stefan Felsner. Chordal graphs as intersection graphs of pseudosegments. In Michael Kaufmann and Dorothea Wagner, editors, *Graph Drawing*, volume 4372 of *Lecture notes in Computer Science*, pages 208–219. Springer, 2007.

[5] Jessica Enright. Subtree overlap graphs - towards recognition. Master's thesis, University of Alberta, Department of Computing Science, 2006.

[6] Jessica Enright and Lorna Stewart. Subtree filament graphs are subtree overlap graphs. *Inf. Process. Lett.*, 104(6):228–232, 2007.

[7] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory (B)*, 16:47–56, 1974.

[8] Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.*, 73(5-6):181–188, 2000.

[9] P.C. Gilmore and A.J. Hoffman. A characterization of comparability graphs and of interval graphs. *Canadian Journal of Mathematics*, 16:539 – 548, 1964.

[10] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland, 2004.

[11] Martin Charles Golumbic and Robert E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151–159, 1985.

[12] Martin Charles Golumbic, Marina Lipshteyn, and Michal Stern. Equivalences and the complete hierarchy of intersection graphs of paths in a tree. *Discrete Applied Mathematics*, 156(17):3203–3215, 2008.

[13] Robert E. Jamison and Henry Martyn Mulder. Constant tolerance intersection graphs of subtrees of a tree. *Discrete Mathematics*, 290(1):27–46, 2005.

[14] Clyde L. Monma and Victor K.-W. Wei. Intersection graphs of paths in a tree. *J. Comb. Theory, Ser. B*, 41(2):141–181, 1986.

[15] Martin Pergel. Recognition of polygon-circle graphs and graphs of interval filaments is NP-complete. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *WG*, volume 4769 of *Lecture Notes in Computer Science*, pages 238–247. Springer, 2007.

[16] Donald J. Rose, R. Endre Tarjan, and George S. Leuker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 5(2):266–283, 1976.

[17] William Rosgen. Set representations of graphs. Master's thesis, University of Alberta, Department of Computing Science, 2005.

[18] Alejandro A. Schäffer. A faster algorithm to recognize undirected path graphs. *Discrete Appl. Math.*, 43(3):261–295, 1993.

[19] Jeremy Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16:264–282, 1994.

[20] Juraj Stacho and Michel Habib. Polynomial-time algorithm for the leafage of chordal graphs. *17th Annual European Symposium on Algorithms*, 2009.

Chapter 3

Filament characterisations of overlap graphs

Introduction

Subtree overlap graphs are the overlap graphs of subtrees in a tree. Given a subtree overlap representation for a graph, we can solve maximum independent set and maximum clique on that graph [1]. Subtree filament graphs and interval filament graphs were introduced by Gavril in [3]. Pergel [4] showed that recognising interval filament graphs is NP-complete. Enright and Stewart [2] showed that subtree overlap graphs are exactly subtree filament graphs. We present a generalisation of that result that also implies that caterpillar overlap graphs are exactly interval filament graphs, and that therefore recognising caterpillar overlap graphs is NP-complete.

Overlap graphs are not nearly as well studied as intersection graphs. Resolving the complexity of recognising overlap graphs has proven to be more difficult than for intersection graphs. The complexity of recognising subtree overlap graphs remains open. The characterisation in this paper provides an alternate way of looking at some overlap graphs as intersection graphs. This may lead to new insights into these classes.

We consider only graphs that are finite, simple and loopless. Let A and B be two sets. A and B are *disjoint*, denoted $A|B$ if they do not intersect. A *overlaps* B, denoted $A \between B$ if $A \cap B$ but neither $A \subseteq B$ nor $B \subseteq A$.

Let A, B, A', B' be four sets. We say that A, B and A', B' are *similarly related*, denoted $A, B \sim A'B'$ if $A|B$ if and only if $A'|B'$, $A \between B$ if and only if $A' \between B'$, $A \subseteq B$ if and only if $A' \subseteq B'$ and $B \subseteq A$ if and only if $B' \subseteq A'$.

Let $G = (V, E)$ be a graph. A family of sets $\mathcal{S}$ is an intersection (overlap, disjointness) representation of G if there is a bijection between the vertices in V and the sets in $\mathcal{S}$ such that two vertices are adjacent if and only if their corresponding sets intersect (overlap, are disjoint).

Subtrees $\mathcal{T}$ of tree T are a *subtree overlap representation* of graph $G = (V, E)$ if there is a bijection between the vertices in V and the subtrees in $\mathcal{T}$ such that two vertices are adjacent if and

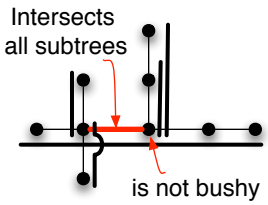

Figure 3.1: An example of a subtree overlap representation that is minimally-S-covered if S is the edge that intersects all subtrees indicated by the arrow. The underlying tree is shown in dots and thin lines, the representing subtrees in thicker lines. The node on the right of the covering edge is bushy with respect to that edge because all of its neighbours that are not in S are leaves, whereas the node on the right of the edge is not, as it has neighbours that are neither in S nor a leaf.

only if their corresponding subtrees overlap. For convenience, we will use the notation that vertex $v_i \in V$ corresponds to subtree $s_i \in \mathcal{S}$. We call the subtrees corresponding to vertices *representing subtrees*.

A *subdivision* of an edge (u, v) in a graph G is an operation in which we remove the edge between u and v, and add a new vertex adjacent to exactly u and v. We say graph H is a *subdivision* of graph G if H can be produced from G by a single or repeated edge subdivisions, or is isomorphic to such a graph.

Let subtrees $\mathcal{T}$ of tree T be a subtree overlap representation for graph $G = (V, E)$. Let S be a tree. We say that this representation is *S-covered* if there is a subtree R of T isomorphic to a subdivision of S that intersects all subtrees in $\mathcal{T}$. We say that the representation is *minimally-S-covered* by subtree R of T if R is isomorphic to a subdivision of S that intersects all subtrees in $\mathcal{T}$, and there is no subtree of R isomorphic to a subdivision of S that intersects all subtrees in $\mathcal{T}$. We say that the representation is *minimally-S-covered* if there exists a subtree R of T that is a subdivision of S and the representation is minimally-S-covered by R.

We say a graph $G = (V, E)$ is *S-covered* if there exists a S-covered representation of G.

Let S be a tree, $G = (V, E)$ a graph, and subtrees $\mathcal{T}$ of tree T a minimally-S-covered representation of G with R being the covering subtree of T that is a subdivision of S. A node p of R is *bushy with respect to* R if every neighbour of p not in R is a leaf. The representation of G is minimally-S-bushy if there is a subtree R of T that is a subdivision of S that intersects all subtrees in $\mathcal{T}$, and there is no subtree of R that is a subdivision of S and intersects all subtrees in T, and every node in R is bushy with respect to R.

For examples of bushy and non-bushy nodes, and a minimally-S-covered representation, see Figure 3.1.

Let $G = (V, E)$ be a graph and $\mathcal{G}$ a graph class. G is $\mathcal{G}$-mixed if there is a partition of its edges into E_1 and E_2 such that:

- $G_1 = (V, E_1)$ is a comparability graph and

- $G_2 = (V, E_2)$ is in $\mathcal{G}$ and
- there is a transitive orientation $\overrightarrow{E_1}$ of E_1 such that for every pair of edges $(u \to w) \in \overrightarrow{E_1}$ and $(v, w) \in E_2$, we have $(u, v) \in E_2$.

Filaments are curves in a surface above some geometric object, as defined by Gavril [3]. *Subtree filaments* are curves in a surface above a tree. Let $\mathcal{T}$ be a family of subtrees of a tree T that is embedded in a plane P. The *filament surface* defined by T is the surface orthogonal to P that intersects P at exactly T. This surface can be imagined to be formed by drawing T upwards from P to form a surface. Filaments $\mathcal{F} = \{f_1...f_n\}$ on the elements of $\mathcal{T} = \{t_1...t_n\}$ are then curves in the filament surface where each $f_i, 1 \leq i \leq n$ connects the leaves of t_i, and for two filaments $f_i, f_j \in \mathcal{F}$ corresponding to $t_i, t_j \in \mathcal{T}$:

- if $t_i | t_j$ then $f_i | f_j$ and
- if $t_i \between t_j$ then f_i intersects f_j.

Gavril [3] showed that the subtree filament graphs are exactly the complements of cochordal-mixed graphs, and the interval filament graphs (the intersection graphs of filaments on intervals on a line) are the complements of cointerval-mixed graphs.

Let T be a tree. If there is a subtree filament representation of graph G such that the host tree embedded in the plane is isomorphic to a subdivision of T, then we say that G is a *T-filament* graph. Similarly, if $\mathcal{G}$ is a class of trees, and there is a subtree filament representation of graph G such that the host tree embedded in the plane is a subdivision of a tree in $\mathcal{G}$, then we say that G is a *$\mathcal{G}$-filament graph*. By this notation, the interval filament graphs are path-filament graphs, as well as the P_2-filament graphs.

Let *disjointness of T* graphs be the graphs that have disjointness representations with a subdivision of tree T as the host tree. Similarly, if $\mathcal{G}$ is a class of trees, let the *disjointness of $\mathcal{G}$* graphs be the graphs that have disjointness representations with a subdivision of a tree in $\mathcal{G}$ as the host tree.

3.1 Results

We now give several lemmas that allow us to transform a subtree overlap representations of graphs into representations with convenient properties. We will use these lemmas in our eventual theorem on the relationship between T-filament graphs and T-covered graphs.

Lemma 22. *Let subtrees $\mathcal{T}$ of tree T be a minimally-S-covered by R subtree overlap representation of graph $G = (V, E)$. There exists a minimally-S-bushy representation of G.*

Proof. Starting with a minimally-S-covered representation $\mathcal{T}$, T, we show how to create a representation of G that is minimally-S-covered and has at least one fewer non-bushy node with respect to R than $\mathcal{T}$, T. Applied inductively, this gives us a representation of G that is minimally-S-bushy.

Let p be a node of R that is not bushy with respect to R. Let $\mathcal{T}_p \subseteq \mathcal{T}$ be the set of subtrees that contain p, and let V_p be the corresponding vertices of G. Let $q_1...q_h$ be the h neighbours of p in $T \backslash R$. Because R hits all subtrees in $\mathcal{T}$, there is no subtree in $\mathcal{T}$ such that removing any $(p, q_i), 1 \leq i \leq h$ edge would disconnect that subtree from p. That is, every subtree that contains a vertex that is reachable from p only through one of $q_1...q_h$ also contains p.

Let T' be a tree that consists of the component of $T \backslash \{q_1...q_h\}$ that contains p, with $|V_p|$ extra leaves added, each adjacent to only p. We call these leaves $l_1...l_{|\mathcal{T}_p|}$, and associate each leaf with a member of $\mathcal{T}_p$.

Let subtree $t'_i, 1 \leq i \leq |\mathcal{T}|$ of T' be defined as follows:

- if $t_i \in \mathcal{T}$ does not contain p, then $t'_i = t_i$
- if $t_i \in \mathcal{T}$ does contain p, then let L_{t_i} be the set of leaves in $l_1...l_{|\mathcal{T}_p|}$ that are associated with either t_i or a member of $\mathcal{T}_p$ that is contained in t_i. Then $t'_i = (t_i \cap T') \cup L_{t_i}$.

Let $\mathcal{T}'$ be composed of all these t'_i.

We claim that for every $1 \leq i, j \leq |V|$ the subtrees $t_i, t_i \sim t'_i, t'_j$, and therefore $\mathcal{T}'$ is a representation for G. Subtree t'_i only differs from t_i if t_i contains p. Therefore if $t_i | t_j$, then at most one of t'_i, t'_j differs from its original version and $t'_i | t'_j$. If subtree $t_i \subseteq t_j$ than every node in t'_i that is not in t_i is also in t'_j, therefore $t'_i \subseteq t'_j$. If $t_i \between t_j$, then t_i intersects t_j, and because this representation is in a tree and they both intersect R, they must intersect at a node in R. No nodes in R were removed from subtrees, so t'_i and t'_j intersect. We now proceed by contradiction: without loss of generality assume that $t'_i \subseteq t'_j$. If $t_i \notin \mathcal{P}$ then $t_i = t'_i$ does not contain p, and the vertex $z \in t_i \backslash t_j$ must be in $T \cap T'$, and therefore is not in t'_j, a contradiction. If t_i is in $\mathcal{T}_p$, we know that there is a leaf in $l_1...l_{|\mathcal{T}_p|}$ that is in only t'_i, and not t'_j because $t_i \not\subseteq t_j$, a contradiction.

The vertex p is now bushy with respect to R, and every other node in T' that is not bushy with respect to R is also not bushy with respect to R in T. R intersects every member of $\mathcal{T}$, and is minimal by its previous minimality in T. We have described a representation of G that is minimally-S-covered, and that has at least one fewer non-bushy node than $\mathcal{T}, T$. □

A *boundary node* of a subtree t of tree T is a node of t that is either a leaf of both t and T, or has a neighour in T that is not in t.

We have showed how to produce a bushier representation. It simplifies our later proof to use a representation in which no two representing subtrees share a boundary node, and no boundary node of a representing subtree is of high degree in the underlying tree. We therefore show how to produce a representation with these convenient properties.

Lemma 23. *If there exists a minimally-S-covered subtree overlap representation of graph G, then there exists a minimally-S-covered subtree overlap representation of G in which there are no boundary nodes of degree greater than two in the underlying tree, and no vertex is a boundary node of more than one representing subtree.*

Proof. Let subtrees $\mathcal{T}$ of tree T be a minimally-S-covered by R representation of graph $G = (V, E)$.

We describe iterative methods for decreasing the number of boundary nodes of degree greater than two and the number of boundary nodes shared between subtrees while preserving all the properties we require.

Let $p \in t_i \in \mathcal{T}$ be a boundary node of subtree t_i with degree greater than two. Let $N_o(p) = \{r_1...r_{|N_o(p)|}\}$ be the neighbourhood of p outside of t_i in T. Then let tree T' be T with each edge between p and $r_j \in N_o(p)$ subdivided with a new vertex q_j. Then for each $t_j \in \mathcal{T}$, let t'_j be defined as follows: if $p \notin t_j$, then $t'_j = t_j$, if $p \in t_j$, then $t'_j = t_j \cup \{q_1...q_{|N_o(p)|}\}$.

We claim that for every pair of subtrees $t_i, t_j \in \mathcal{T}$, $t_i, t_j \sim t'_i, t'_j$, that there are fewer boundary nodes of degree greater than two in $\mathcal{T}', T'$ than in $\mathcal{T}, T$, and that $\mathcal{T}', T'$ is minimally-S-covered such that a subdivision of R intersects all subtrees in $\mathcal{T}'$.

Recall that $\mathcal{T}', T'$ is minimally-S-covered if there is a minimal subtree R of T that is a subdivision of S and intersects all subtrees in $\mathcal{T}$. Because for every subtree $t_i \in \mathcal{T}$, we have $t_i \cap R = t'_i \cap R$ and the new nodes either subdivide R, or R is a connected subtree of T', the subtrees in $\mathcal{T}', T'$ are minimally-S-covered. Node p is no longer a boundary node, no new nodes of degree greater than two have been introduced, and every node in $T' \cap T$ that is not a boundary node of any subtree in $\mathcal{T}$ is not a boundary node of any subtree in $\mathcal{T}$. Therefore there are fewer boundary nodes of degree greater than two in $\mathcal{T}', T'$ than in $\mathcal{T}, T$.

A subtree $t'_i \in \mathcal{T}'$ contains the new nodes if and only if t_i contains p. It follows that every pair of subtrees satisfies $t'_i, t'_j \sim t_i, t_j$.

Applied iteratively, this gives us a method for producing a minimally-S-covered subtree overlap representation with no boundary nodes of degree greater than two. Then let subtrees $\mathcal{T}_2$ of tree T_2 be a minimally-S-covered subtree overlap representation of G with no boundary nodes of degree greater than two.

We now describe a method of decreasing the number of shared boundary nodes between two subtrees in a representation that, if used on a minimally-S-covered representation with no boundary nodes of degree greater than two (such as $\mathcal{T}_2, T_2$), produces a minimally-S-covered representation with no boundary nodes of degree greater than two with fewer shared boundary nodes.

Let R_2 be a subdivision of S that is a subtree of T_2 and intersects all members of $\mathcal{T}_2$. Let p be a node in T_2 that is the boundary node of more than one subtree in $\mathcal{T}_2$. If p is not a leaf, then let q and r be the neighbours of p in T_2 - recall that there are no boundary nodes in $\mathcal{T}_2, T_2$ of degree greater than two. If p is a leaf, let q be its neighbour. Let $\mathcal{T}_q \subset \mathcal{T}_2$ be the family of subtrees for which p is a leaf that also contain q, and $\mathcal{T}_r \subset \mathcal{T}_2$ be the family of subtrees for which p is a leaf that also contain r. If p is a leaf, let $\mathcal{T}_r$ be the empty set. Let both $\mathcal{T}_q$ and $\mathcal{T}_r$ be sorted by increasing size. Let $n_q = |\mathcal{T}_q|$ and $n_r = |\mathcal{T}_r|$.

Let T_3 be the tree T_2 with the edge between p and q subdivided by a path of new vertices $s^q_1...s^q_{n_q}$ such that s^q_1 is adjacent to p, $s^q_{n_q}$ adjacent to q, and the edge between p and r (if p is not a

leaf) subdivided by a path of new vertices $s_1^r ... s_{n_r}^r$ such that s_1^r is adjacent to p, $s_{n_r}^r$ adjacent to r.

For each subtree $t_i \in \mathcal{T}_2$, let subtree t_i' be defined as follows:

- If $t_i \in \mathcal{T}_q$ then $t_i' = t_i \cup \{s_j^q$ where j is less than the position of t_i in $\mathcal{T}_q\}$.
- If $t_i \in \mathcal{T}_r$ then $t_i' = t_i \cup \{s_j^r$ where j is less than the position of t_i in $\mathcal{T}_r\}$.
- If $q \in t_i$ and $r \in t_i$, then $t_i' = t_i \cup \{s_1^q ... s_{n_q}^q\} \cup \{s_1^r ... s_{n_r}^r\}$.
- If $p \notin t_i$ then $t_i' = t_i$.

Note that, if p is a leaf, the above subtree constructions do not include any vertices in the nonexistent subdividing path $s_1^r ... s_{n_r}^r$.

Let $\mathcal{T}_3$ be all such t_i'.

We claim that for every pair of subtrees $t_i, t_j \in \mathcal{T}_2$, $t_i, t_j \sim t_i', t_j'$, that there are fewer boundary nodes in $\mathcal{T}_3, T_3'$ that are boundary nodes for more than one subtree than there are in $\mathcal{T}_2, T_2$, that $\mathcal{T}_3, T_3$ is minimally-S-covered, and that $\mathcal{T}_3, T_3$ has no boundary nodes of degree greater than two.

Every subtree $t_i' \in \mathcal{T}_3$ has the same intersection with R_2 that t_i does, and the new nodes either subdivide R_2, or R_2 is a subtree of T_3, therefore $\mathcal{T}_3, T_3$ is minimally-S-covered. Because $\mathcal{T}_2, T_2$ has no boundary nodes of degree greater than two, we have not added any nodes of degree greater than two, and every node in T_3 that was not a boundary node in $\mathcal{T}_2, T_2$ is not a boundary node in $\mathcal{T}_3, T_3$, representation $\mathcal{T}_3, T_3$ has no boundary nodes of degree greater than two.

It remains to show that for every pair of subtrees $t_i, t_j \in \mathcal{T}_2$, $t_i, t_j \sim t_i', t_j'$. A subtree $t_i' \in \mathcal{T}_3$ contains nodes not in t_i if and only if t_i contains p. Therefore $t_i | t_j$ implies $t_i' | t_j'$. Because only new nodes on the subdividing paths were added to subtrees, and no nodes were removed, $t_i \between t_j$ implies $t_i' \between t_j'$. If $t_i \subseteq t_j$, then any node in $t_i' \backslash t_i$ is also in $t_j' \backslash t_j$, therefore $t_i' \subseteq t_j'$.

Applied iteratively, this gives us a method for producing from $\mathcal{T}_2, T_2$ a subtree overlap representation with no boundary nodes of degree greater than two that is minimally-S-covered and has no nodes that are boundary nodes of more than one representing subtree.

□

Theorem 8. *Let $G = (V, E)$ be a graph and S a nontrivial tree. The following statements are equivalent:*

1. *G is minimally-S-covered.*
2. *There is an overlap representation of G which is minimally-S-bushy*
3. *G is a S-filament graph*
4. *G is the complement of a (disjointness of S)-mixed graph.*

Proof. **From 1 to 2**: By Lemma 22

From 2 to 3:

By Lemma 23 there is a minimally-S-covered subtree overlap representation of $G = (V, E)$ with no shared boundary nodes and no boundary nodes of degree greater than two. Let subtrees $\mathcal{T}$ of

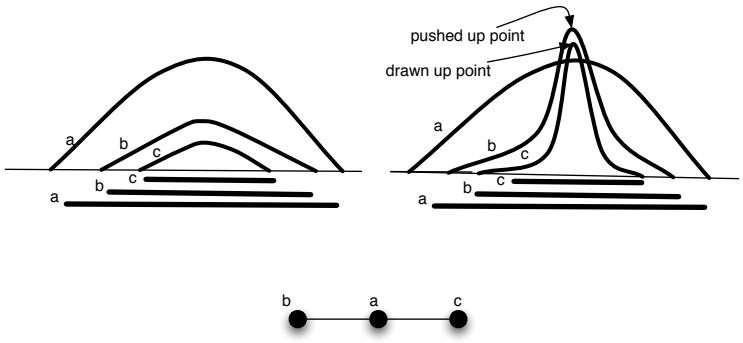

Figure 3.2: A graph on the bottom and the initial stage of construction of filaments for that graph on the left. Note that the filaments for c and d do not intersect the filament for a, so these filaments are not yet an intersection representation for the graph. On the right, we have drawn up a point of the filament for c to intersect the filament for a. During the drawing up, we encountered the filament for b, and because c and b are not adjacent, a point of the filament for b was pushed up.

tree T be such a representation, with R being a minimal subtree of T that is a subdivision of S and intersects all subtrees in $\mathcal{T}$.Because there are no shared boundary nodes, no two subtrees in $\mathcal{T}$ are equal.

First, we embed R in the plane P. Let Q_L be the filament surface above R. Now we index all members of $\mathcal{T} = \{t_1..t_n\}$ by nondecreasing size.

First, starting at t_1 and proceeding to t_n, we construct a filament f_i for each subtree $t_i \in \mathcal{T}$ such that every point of f_i is above some point of t_i, f_i joins the endpoints of t_i, f_i is above every point of every f_j where $j < i$ and $t_j \subset t_i$, and f_i is a function. Let $\mathcal{F}$ be all such filaments.

For every two subtrees in $\mathcal{T}$ that are disjoint in T, their corresponding filaments in $\mathcal{F}$ do not intersect. For every two subtrees t_i, t_j in $\mathcal{T}$ such that $t_i \subset t_j$, and therefore $t_i \cap R \subseteq t_j \cap R$, their filaments do not intersect.

However, consider two overlapping subtrees in $\mathcal{T}$. Their corresponding filaments may or may not intersect. We therefore modify the filaments in $\mathcal{F}$ such that filaments corresponding to overlapping subtrees do intersect.

Let $t_i, t_j \in \mathcal{T}$ be two subtrees that overlap, but f_i, f_j do not intersect. First, notice that $t_i \cap R$ intersects $t_j \cap R$, and because we are assuming that no two subtrees share a boundary node, $t_i \cap t_j \cap R$ contains at least an edge. Let (p, q) be that edge. Assume without loss of generality that $i < j$, and therefore f_i is entirely below f_j. We take a point of f_i above (p, q) and draw it upwards until f_i intersects f_j, stretching f_i so that it remains a function. If, in drawing up f_i, we encounter another filament f_k such that $t_k \not\subset t_i$, we cross f_k with f_i. However, if we encounter a filament f_k such that $t_i \subset t_k$, then we draw up the point of f_k that we encounter with f_i such that f_i does not intersect f_k. We say that f_k modified in this manner has been *pushed up*. For an example of drawn up and pushed up points, see Figure 3.2

We perform this drawing up operation for every pair of filaments that do not intersect but corre-

spond to subtrees that overlap in $\mathcal{T}, T$.

We claim that now two filaments in $\mathcal{F}$ intersect if and only if their corresponding subtrees in $\mathcal{T}$ overlap, and therefore these filaments form an intersection representation for G.

Certainly filaments corresponding to overlapping subtrees now intersect, and filaments corresponding to disjoint subtrees do not intersect.

It remains to show that filaments $f_i, f_j \in \mathcal{F}$ such that $t_i \subset t_j$ do not intersect. If f_i was neither drawn up nor pushed up, this is the case as in the original set of filaments.

If f_i was drawn up then, if f_j was encountered during the drawing up, it was pushed up, so f_j remains entirely above f_i.

If f_i was pushed up, let f_k be the filament that was being drawn up when f_j was pushed. $t_k \subset t_i$, and therefore by transitivity of containment $t_k \subset t_j$. Therefore t_j was pushed up as well, and t_j remains entirely above t_i.

From 3 to 4:

This proof closely follows ones given in [3]. Let filaments $\mathcal{F}$ on subtrees $\mathcal{S}$ of tree S be a subtree filament representation of G. We describe a partition of the nonedges of G to show that G is the complement of a (disjointness-of-S)-mixed graph. Let (v_i, v_j) be an edge in $\overline{E}$. We know that f_i, f_j do not intersect. Let s_i, s_j be the subtrees of S induced by the endpoints of t_i, t_j. If $s_i | s_j$ we place (v_i, v_j) in E_1. If $s_i \subset s_j$ then we place $(v_i \to v_j)$ in $\overrightarrow{E_2}$. If $s_i = s_j$, then without loss of generality assume that f_i is entirely below f_j. Then we place $(v_i \to v_j)$ in $\overrightarrow{E_2}$. If $s_j \subset s_i$ then we place $(v_j \to v_i)$ in $\overrightarrow{E_2}$. We claim that $E_1, \overrightarrow{E_2}$ is a (disjointness of S)-mixed partition of the edges of $\overline{G}$.

First, $G_1 = (V, E_1)$ is a disjointness-of-S graph because two vertices v_i, v_j are adjacent if and only if s_i, s_j are disjoint. Therefore $\mathcal{S}$ is a disjointness representation of G_1.

Secondly, $G_2 = (V, E_2)$ is a transitive orientation of a comparability graph, by the transitivity of containment.

It remains to show that for every three vertices $v_i, v_j, v_k \in V$, if $(v_i \to v_j) \in E_2$ and $(v_j, v_k) \in E_1$, then $(v_i, v_k) \in E_1$. If $(v_i \to v_j) \in E_2$ then $s_i \subset s_j$, and if $(v_j, v_k) \in E_1$ then $s_j | s_k$. Geometry then forces s_i to be disjoint from s_k, and therefore $(s_i, s_k) \in E_1$.

From 4 to 1:

This proof closely follows ones given in [3]. Let E_1, E_2 be a (disjointness-of-S)-mixed partition of the edges in $\overline{E}$. Let subtrees $\mathcal{S}$ of tree S be a disjointness representation of $G_1 = (V, E_1)$.

Let $v_i \to v_j$ be in E_2. Is it the case that $s_i \subset s_j$? Assume it is not. Because s_i, s_j are nondisjoint, either $s_i \between s_j$ or $s_j \subset s_i$. Consider a third subtree s'_j such that $s'_j = s_j \cup s_i$. We claim that s'_j is a valid representing subtree for v_j. Assume that it is not - then s'_j must intersect some subtree s_k such that $(v_j, v_k) \in E_1$. Because $s_j | s_k$, it must be that s_i intersects s_k. Then we have that $(v_i \to v_j) \in E_2$, $(v_j, v_k) \in E_1$, but $(v_i, v_k) \notin E_1$, a contradiction. Then s'_j is a valid representing subtree for v_j, we could use it instead of s_j. We therefore assume that for every pair of vertices v_i, v_j such that $(v_i \to v_j) \in E_1$, $s_i \subset s_j$.

Let v_i, v_j be two vertices adjacent in $G = (V, E)$. Their corresponding subtrees are not disjoint in $\mathcal{S}, S$. Without loss of generality either $s_i \subset s_j$ or $s_i \between s_j$. Assume that $s_i \subset s_j$. We show how to produce subtrees $\mathcal{S}'$ of tree S' such that $s'_i \between s'_j$, and all other pairs of subtrees are similarly related. Let p be a vertex of S that is contained in both s_i and s_j. Let tree S' be the tree S with two new leaves q, r adjacent to p.

Let s'_i be $s_i \cup q$ and s'_j be $s_j \cup r$. For every other subtree $s_k \in \mathcal{S}$, we define s'_k as:

- If s_k does not contain p, then $s'_k = s_k$
- If s_k contains p, then $s'_k = s_k \cup q$ (if $s_i \subset s_k$) $\cup r$ (if $s_j \subset s_k$)

Let $\mathcal{S}'$ be all such subtrees.

Then $s'_i \between s'_j$, and for every other pair $s_k, s_l \in \mathcal{S}$, $s_k, s_l \sim s'_k, s'_l$. Observe that $S \subset S'$ intersects all subtrees in $\mathcal{S}'$. Then by iterated application, we can get a minimally-S-covered overlap representation of G.

□

The caterpillar overlap graphs are overlap graphs of subcaterpillars in a caterpillar. It follows from Theorem 8:

Corollary 3. *Caterpillar overlap graphs are exactly interval filament graphs and path-covered subtree overlap graphs.*

Because Pergel [4] showed that recognising the interval filament graphs is NP-complete, recognising the caterpillar overlap graphs is also NP-complete.

Bibliography

[1] Eowyn Čenek and Lorna Stewart. Maximum independent set and maximum clique algorithms for overlap graphs. *Discrete Appl. Math.*, 131(1):77–91, 2003.

[2] Jessica Enright and Lorna Stewart. Subtree filament graphs are subtree overlap graphs. *Inf. Process. Lett.*, 104(6):228–232, 2007.

[3] Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.*, 73(5-6):181–188, 2000.

[4] Martin Pergel. Recognition of polygon-circle graphs and graphs of interval filaments is NP-complete. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *WG*, volume 4769 of *Lecture Notes in Computer Science*, pages 238–247. Springer, 2007.

Chapter 4

List colouring permutation and interval graphs with a fixed colour bound[1]

4.1 Introduction

In the vertex colouring problem, we try to assign each vertex in a graph a colour such that no two adjacent vertices are assigned the same colour using the minimum number of colours.

In the vertex list colouring problem, each vertex has a list of colours, and we try to assign each vertex a colour from its list such that no two adjacent vertices are assigned the same colour. Determining if this is possible is a decision problem and is NP-complete, as it is a generalization of vertex colouring [5]. List colouring remains hard even on interval graphs [1], as well as split graphs, cographs, and bipartite graphs [4]. It is solvable in polynomial time on trees [4].

List colouring with fixed colour bound of natural number $k \geq 3$ is a generalization of k-vertex colouring, and so is NP-complete. It remains NP-complete on planar bipartite graphs [6], but is solvable in polynomial time on graphs of fixed treewidth [3]. We give a polynomial-time algorithm for solving list colouring with fixed colour bound k on a class of graphs that does not have fixed treewidth.

2-list colouring a graph is solvable in polynomial time. Let $G = (V, E)$ be the graph we wish to 2-list colour and $\mathcal{P}$ be the list mapping for vertices in V. Let C_1, C_2 be the two colours that occur in lists in $\mathcal{P}$. We need only check to see if there is a bipartition of the graph in which all vertices with only C_1 in their lists are on one side of the bipartition and all vertices with only C_2 in their lists are on the other. Graph G with list mapping $\mathcal{P}$ is 2-list colourable if and only if there exists such a bipartition.

Permutation graphs are exactly comparability cocomparability graphs - the graphs that admit transitive orientations of both their edges and their nonedges. In this paper we give a polynomial algorithm for list colouring with a fixed colour bound of $k \geq 3$ on permutation graphs.

[1] This is joint work with Lorna Stewart and Gabor Tardos.

Our algorithm uses the layers of a breadth-first search rooted at a particular vertex in what we call a multi-chain ordering. This ordering is closely related to the ordering used by Heggernes et al. [2] to compute the bandwidth of bipartite permutation graphs in polynomial time. Our ordering, which applies to the larger class of permutation graphs, is expressed in terms of the layers of a breadth first search. This sort of ordering gives insight into the structure of permutation graphs, and may lead to algorithms for other problems on permutation graphs.

4.2 Definitions and Preliminaries

A graph $G = (V, E)$ is a pair of vertex set V and edge set E composed of subsets of V of size two. All graphs that we consider are connected, finite, simple, and loopless. A directed graph $G = (V, E)$ is a pair of vertex set V and edge set E composed of ordered pairs of V.

A transitive orientation of the edges of a graph is an orientation such that the presence of edges $(u \rightarrow v)$ and $(v \rightarrow w)$ implies edge $(u \rightarrow w)$. A comparability graph is a graph that admits a transitive orientation of its edges. A cocomparability graph is a graph that admits a transitive orientation of its nonedges.

Let $G = (V, E)$ be a graph. A *list mapping* of G is a mapping that assigns a list of colours to each vertex in G. A colouring of G *obeys* a list mapping $\mathcal{P}$ of G if every vertex is assigned a colour that is in that vertex's list in $\mathcal{P}$. A *proper colouring* of a graph is a colouring in which no two adjacent vertices are assigned the same colour. A k-list colouring is a proper colouring that obeys a list mapping in which at most k distinct colours occur in the lists assigned to vertices, and at most k colours are therefore used in the colouring.

We might sometimes say that a list mapping *precolours* a vertex. By this we mean that the list mapping assigns a list with only a single colour to that vertex.

Let $G = (V, E)$ be a graph and $\mathcal{L} = [L_0 ... L_z]$ be the layers of a breadth-first traversal of G with vertex v_0 as the starting point. We call $\mathcal{L}$ a *multi-chain ordering* of G if for every two vertices u, v in layer $L_i, 0 \leq i \leq z$ the neighbourhood of u in L_{i-1} (if $i - 1 \geq 0$) is a subset of the neighbourhood of v in L_{i-1} (or vice versa) and the neighbourhood of u in L_{i+1} (if $i + 1 \leq z$) is a subset of the neighbourhood of v in L_{i+1} (or vice versa).

Lemma 24. *Let $\overrightarrow{G} = (V, \overrightarrow{E})$ be a transitive orientation of a comparability graph $G = (V, E)$ in which v_0 is a source or a sink, and $[L_0 ... L_z]$ be the layers of a breadth first search traversal of G starting at v_0. For every two consecutive layers L_i, L_{i+1} for $0 \leq i < z$, either all edges in $\overrightarrow{E}$ between vertices of L_i and L_{i+1} are directed toward L_i or all edges in $\overrightarrow{E}$ between vertices of L_i and L_{i+1} in are directed toward L_{i+1}.*

Proof. This follows from the fact that v_0 is a source or sink in $\overrightarrow{G}$, and the observation that there are no edges between nonconsecutive layers of a breadth first search. □

Lemma 25. *Let $\overrightarrow{\overline{G}} = (V, \overrightarrow{\overline{E}})$ be a transitive orientation of the complement of a comparability graph $G = (V, E)$ in which v_0 is a sink and $\mathcal{L} = [L_0...L_z]$ are the layers of a breadth-first search traversal of G starting at v_0. Then for every pair of layers L_i, L_j where $0 \leq i < j \leq z$ all nonedges between L_i and L_j are directed toward L_i.*

Proof. We proceed by induction. First, consider the level L_0. Because the only vertex on L_0 is a sink in $\overline{G}$, all nonedges of G between another level and the vertex on level L_0 are directed toward L_0.

Assume that for every layer L_h such that $h \leq i$, all nonedges between L_h and a layer of index greater than h are directed toward L_h. Then consider L_{i+1}.

Now we continue by contradiction. Assume that there is a nonedge between vertex v_{i+1} in L_{i+1} and vertex v_j in some layer L_j where $j > i + 1$ that is directed toward L_j. Let v_i be a neighbour of v_{i+1} in layer L_i. Because the layers are produced by a breadth first traversal, there is a nonedge between v_j and v_i, and by the inductive assumption, it is directed toward v_i. Then we have in G a nonedge directed from v_{i+1} to v_j and from v_j to v_i, but no nonedge between v_i and v_{i+1}, a contradiction.

□

Lemma 26. *Let $G = (V, E)$ be a permutation graph and let $\overrightarrow{G}$ be a transitive orientation of G in which v_0 is a source or a sink, and $\overrightarrow{\overline{G}}$ a transitive orientation of the complement of G in which v_0 is a sink. Let $\mathcal{L} = [L_0...L_z]$ be the layers of a breadth-first search traversal of G rooted at v_0. Then $\mathcal{L}$ is a multi-chain ordering.*

Proof. Let u, v be two vertices on L_i. We consider two cases: if u, v are adjacent, and if they are not adjacent.

We proceed by contradiction. Assume that u, v are adjacent and that there are vertices x, y in L_{i+1} such that x is adjacent to u but not v and y is adjacent to v but not u. By Lemma 24, the edges between u and x and between v and y are either both directed toward L_i, or both directed toward L_{i+1}. In either case there is no transitive orientation of the edge between u and v, a contradiction. Similarly, if x, y are in L_{i-1} by Lemma 24, the edges are either both directed toward L_i, or both directed toward L_{i+1}. In either case there is no transitive orientation of the edge between u and v, a contradiction.

Now assume that u, v are not adjacent and that there are vertices x, y in L_{i+1} such that x is adjacent to u but not v and y is adjacent to v but not u. By Lemma 25, these nonedges are directed as $(x \to v)$ and $(y \to u)$. Then there is no transitive orientation of the nonedge between u and v, a contradiction. Similarly, if x, y are in L_{i-1} then by Lemma 25, these nonedges are directed as $(v \to x)$ and $(u \to y)$. Then there is no transitive orientation of the nonedge between u and v, a contradiction.

We have shown that whether or not two vertices u, v on layer L_i are adjacent, the neighbourhood of one in L_{i-1} is a subset of the neighbourhood of the other in L_{i-1}, and the neighbourhood of one in L_{i+1} is a subset of the neighbourhood of the other in L_{i+1}. □

Theorem 9. *Every permutation graph has a multi-chain ordering.*

Proof. This follows from the comparability cocomparability orderings of permutation graphs (as described in [8]) and the previous three lemmas. □

But also observe that:

Observation 5. *Not every graph with a multi-chain ordering is a permutation graph.*

By similar reasoning to that for permutation graphs, we can also show that interval graphs have multi-chain orderings.

What does this neighbourhood containment mean for our list colouring? In each layer there is a vertex with a maximal neighbourhood in the previous layer, and a vertex with a maximal neighbourhood in the following layer. Because each vertex in a layer has at least one neighbour in the previous layer, this means that there is a vertex in each layer adjacent to all vertices in the next layer. Therefore:

Observation 6. *No single layer in a multi-chain ordering can have vertices of more than $k-1$ colours in it in a valid k-list-colouring.*

We can generate the layers of a breadth-first search in $O(n+m)$ time. We can then sort the vertices on each layer by neighbourhood above, and in a separate ordering by neighbourhood below using bucket sort in $O(n)$ time. Once these are sorted, we can check for the neighbourhood nesting required for a multi-chain ordering in $O(n+m)$ time. We can therefore generate a breadth-first search and check to see if it gives us a multi-chain ordering in $O(n+m)$ time.

As naive algorithm to check if a graph has a multi-chain ordering, and generate it if it does, we can start a breadth-first search from each vertex, and check to see if that search has given us a multi-chain ordering in $O(n(n+m))$ time overall. In some classes, for example permutation graphs, this can be done more quickly. In the case of permutation graphs, we can use the output of the recognition algorithm provided by McConnell and Spinrad [7] to identify a vertex that is both a source or a sink in some transitive orientation of the graph and a sink in a transitive orientation of the complement of the graph. We can then generate a multi-chain ordering in $O(n+m)$ time by virtue of Lemma 26.

4.3 Layer Configurations

We use the layers of a multi-chain ordering to devise a polynomial-time algorithm for fixed colour bound list colouring for graph classes in which all induced subgraphs have multi-chain orderings.

A *configuration* B_i for a layer L_i in a multi-chain ordering is an ordered k-tuple $[B_i^1 ... B_i^k]$ such that every entry is a natural number between 0 and $|L_i|$. When we use these configurations, entry B_i^j will indicate the number of vertices on layer L_{i+1} that are adjacent to a vertex of colour C_j on layer L_i, or equivalently the maximum size of the neighbourhood on L_{i+1} of a vertex of colour C_j on L_i.

We say a colouring $C_1 ... C_k$ of a graph with multi-chain ordering $[L_0 ... L_z]$ *admits* a configuration B_i of layer L_i if for $1 \leq j \leq k$ there are exactly B_i^j vertices in L_{i+1} adjacent to vertex in layer L_i of colour C_j.

Let B_i be a configuration for layer L_i of a multi-chain ordering of graph $G = (V, E)$ with list mapping $\mathcal{P}$. B_i is *good* if there is a list colouring of the vertices in $[L_0 ... L_i]$ that obeys $\mathcal{P}$ and admits B_i.

Let B_i be a configuration for layer L_i and B_{i+1} be a configuration for layer L_{i+1} of a multi-chain ordering of graph $G = (V, E)$ with list mapping $\mathcal{P}$. B_{i+1} is *consistent* with B_i if, if there is a list colouring of the vertices in $L_0 ... L_i$ that obeys $\mathcal{P}$ and admits B_i, then there is a list colouring of the vertices in $L_0 ... L_{i+1}$ that obeys $\mathcal{P}$ and admits B_i and B_{i+1}.

Let $L_0 ... L_z$ be a multi-chain ordering of graph $G = (V, E)$. Let $C_1 ... C_k$ be a proper colouring of the vertices in $L_0 ... L_i$ that obeys colour list mapping $\mathcal{P}$ and admits configuration B_i for L_i. Consider the vertices on L_i, and what their colours might be in a colouring admitting B_i.

Observation 7. *At least one vertex with a neighbourhood in L_{i+1} of size equal to entry B_i^j must be coloured with C_j in any proper colouring of vertices in $L_0 ... L_i$ that admits B_i and obeys $\mathcal{P}$.*

Proof. This follows from the neighbourhood nesting in a multi-chain ordering. □

Observation 8. *Any vertex with a neighbourhood in L_{i+1} of size larger than B_i^j must not be coloured with C_j in any proper colouring of vertices in $L_0 ... L_i$ that admits B_i and obeys $\mathcal{P}$*

Proof. Let $v \in L_i$ be a vertex in L_i with d neighbours in L_{i+1}. If v were coloured with C_j, then there would be d vertices in L_{i+1} adjacent to a vertex of colour C_j in B_i, a contradiction.

□

Lemma 27. *Let $[L_0 ... L_z]$ be a multi-chain ordering of graph $G = (V, E)$. Let $C_1 ... C_k$ be a proper colouring of the vertices in $L_0 ... L_{i+1}$ that obeys colour list mapping $\mathcal{P}$ and admits configuration B_i for L_i and B_{i+1} for L_{i+1}. Any vertex v that occurs h^{th} in a nonincreasing sort of the vertices of L_{i+1} by size of neighbourhood in L_i must not be coloured C_j where $B_i^j > h$.*

Proof. Because the neighbourhoods in L_i of vertices in L_{i+1} are nested, if there is a vertex on layer L_i of colour C_j adjacent to at least h vertices on L_{i+1}, it must be adjacent to v. In any proper colouring admitting B_i there is a vertex of colour C_j on L_i adjacent to $B_i^j > h$ vertices on L_{i+1}, and therefore adjacent to v. □

Assume that we had a list of all good configurations for a layer L_i of a multi-chain ordering of graph $G = (V, E)$ with list mapping $\mathcal{P}$. How could we produce all good configurations for layer L_{i+1}? We will produce and check all configurations consistent with each known good configuration for L_i.

Let B_i be a good configuration for L_i. Let B_{i+1} be a configuration for L_{i+1}. We will check whether B_{i+1} is a good configuration consistent with B_i.

For each entry B^j_{i+1} in B_{i+1}, consider the set of vertices in L_{i+1} that have B^j_{i+1} neighbours in L_{i+2}. By Observations 8 and 7 at least one of these must be coloured C_j, and no vertex with a neighbourhood larger than B^j_{i+1} in L_{i+2} can be coloured C_j.

Let $V^1_{i+1}...V^k_{i+1}$ be sets of vertices such that for $1 \leq j \leq k$ the vertex set V^j_{i+1} contains all vertices on L_{i+1} that have B^j_{i+1} neighbours on L_{i+2}. Let x be the number of combinations of vertices such that one is chosen from each V^j_{i+1}, and let vertex sets $W_1...W_x$ be these combinations. The natural number x is $O(n^k)$. For each W_l where $1 \leq l \leq x$, we create a list mapping $\mathcal{P}'_l$ that assigns lists **only to the vertices in** L_{i+1} such that every vertex not in W_l has the same list as it does in $\mathcal{P}$, but for each vertex $v \in W_l$, $\mathcal{P}'_l$ assigns v colour C_j where $v \in V^j_{i+1}$.

We now modify these colour mappings further. For every vertex v in L_{i+1}, let it be in position h in a sorting of the vertices of L_{i+1} by nonincreasing neighbourhood in L_i. For vertex v, we remove from $v's$ list in $\mathcal{P}'_l, 1 \leq l \leq x$ every colour j such that $h > B^j_i$.

These modified lists in $\mathcal{P}'_1...\mathcal{P}'_x$ contain at most $k-1$ colours in the lists for vertices on L_{i+1}. We know this because there is at least one entry $B^j_i \in B_i$ that is equal to $|L_{i+1}|$. Then C_j does not occur in any list in any of $\mathcal{P}'_1...\mathcal{P}'_x$.

There is a valid $k-1$ list colouring of the vertices on L_{i+1} that obeys at least one of the list mappings in $\mathcal{P}'_1...\mathcal{P}'_x$, if and only if B_{i+1} is a good configuration for L_{i+1} and is consistent with B_i. We prove this in the next two lemmas.

Lemma 28. *If there is a valid $k-1$ list colouring of the vertices on L_{i+1} that obeys at least one one of the list mappings in $\mathcal{P}'_1...\mathcal{P}'_x$, then B_{i+1} is a good configuration for L_{i+1} and is consistent with B_i.*

Proof. Assume that B_i is a good configuration for L_i, and there is a $k-1$ colouring of the vertices on L_{i+1} that obeys list mapping $\mathcal{P}'_l$. Without loss of generality, assume that it is the k^{th} colour that is not used in the $k-1$ colouring of L_{i+1} - that is, that $B^k_i = |L_{i+1}|$. Let $D_1...D_{k-1}$ be the colour classes of the $k-1$ colouring of vertices on L_{i+1}.

There is a colouring of the vertices in $L_0...L_i$ that obeys $\mathcal{P}$ and admits B_i. Let $C_1...C_k$ be the colour classes of this colouring.

We will extend $C_1...C_k$ to give a proper colouring of $L_0...L_{k+1}$ that obeys $\mathcal{P}$ and admits B_{i+1}. For $1 \leq h < k$, let $C'_h = C_h \cup D_h$, and let $C'_k = C_k$.

We claim that $C'_1...C'_k$ is a proper colouring of the vertices in $L_0...L_{i+1}$ that obeys $\mathcal{P}$ and admits both B_i and B_{i+1}.

We proceed by contradiction. Assume that $C'_1...C'_k$ is not a proper colouring. Then there must be two adjacent vertices in the same colour class. Because $C_1...C_k$ and $D_1...D_{k-1}$ are proper colourings, one of these vertices must be in L_i and one in L_{i+1}. Let $u \in L_i$ and $v \in L_{i+1}$ be these adjacent vertices, and C'_h their shared colour class.

Let q be the position of v in a sort of the vertices in L_{i+1} by nonincreasing neighbourhood in L_i. Because $C_1...C_k$ admits B_i, we know that $B_i^h < q$. If B_i^h were at least q, than $C'_h = C_h$ would have been removed from the list of colours possible for v when creating $\mathcal{P}'_l$.

However, the neighbourhood of u in L_{i+1} contains at least q vertices, and therefore we know that in every colouring of $L_0...L_i$ that admits B_i, u is not coloured C_h, a contradiction. □

Lemma 29. *If B_{i+1} is a good configuration for L_{i+1} and is consistent with B_i, then there is is a valid $k-1$ list colouring of the vertices on L_{i+1} that obeys at least one of the list mappings in $\mathcal{P}'_1...\mathcal{P}'_x$.*

Proof. This is a consequence of trying all configurations for L_{i+1}, and all of the list mappings generated.

Let B_{i+1} be a good configuration that is consistent with B_i and assume that there is no $k-1$ list colouring of the vertices in L_{i+1} with any colour mapping in $\mathcal{P}'_1...\mathcal{P}'_x$.

There must be a colouring $C_1...C_k$ of the vertices in $L_0...L_{i+1}$ that obeys $\mathcal{P}$ and admits B_{i+1} and B_i because B_{i+1} is good and consistent with B_i.

Let $W_l \in W_1...W_x$ be any vertex set such that every vertex v in that W_l is assigned C_j by $C_1...C_k$ where $v \in V_{i+1}^j$. Existence of W_l is guaranteed by Observation 8. Then for every vertex $v \in W_l$, $\mathcal{P}'_l(v)$ consists of only the colour assigned to v by $C_1...C_k$.

$C_1...C_k$ gives a $k-1$ colouring of the vertices on L_{i+1}. It must be that at least one vertex on L_{i+1} is assigned a colour by $C_1...C_k$ that is not in its list in $\mathcal{P}'_l$. Let v be this vertex and C_y the colour.

Why is C_y in $\mathcal{P}(v)$ but not in $\mathcal{P}'_l(v)$? It must have been removed because: $h < B_i^y$ where h is the position of v in a nonincreasing sort of the vertices of L_{i+1} by size of neighbourhood in L_i (a contradiction to Observation 7), or the size d of v's neighbourhood in L_{i+2} is larger than B_{i+1}^y (a contradiction to Lemma 27) or $v \in W_l$ and d is equal to some entry in B_{i+1} other than B_{i+1}^j. In this last case we have a contradiction to the fact that every vertex $v \in W_l$, $\mathcal{P}'_l(v)$ consists of only the colour assigned to v by $C_1...C_k$.

□

We then have a polynomial time method, given all good configurations for a layer L_i of a multi-chain ordering of graph $G = (V, E)$ with list mapping $\mathcal{P}$, to generate all good configurations for layer L_{i+1}, provided there is a fixed bound k on the number of colours, and every induced subgraph of G has a multi-chain ordering. Applied recursively, this gives us an algorithm for determining whether such a graph, with its list mapping, is k-list colourable, as given in Algorithm 1.

Algorithm 1 listColouring(Graph $G = (V, E)$, list mapping $\mathcal{P}$, natural number k)

if k = 2 **then**
  check for a bipartition of G such that all vertices with only one colour in their lists are on one side of the bipartition and all vertices with only the other are on the other side.
  if there is such a bipartition **then**
    return true
  else
    return false
  end if
end if
Find a multi-chain ordering $L_0...L_z$
Make $z + 1$ empty lists: $Configurations_0...Configurations_z$
Let v be the single vertex in L_0
for each colour C_i in $\mathcal{P}(v)$ **do**
  add a configuration to $Configurations_0$ that contains a 0 in every entry except the ith one, which is $|L_1|$
end for
for layers L_i where $0 \leq i < z$ **do**
  for each possible configuration B_{i+1} for L_{i+1} **do**
    for each configuration B_i in $Configurations_i$ **do**
      let $V_{i+1}^1...V_{i+1}^k$ be the sets of vertices such that V_{i+1}^h contains all vertices on L_{i+1} that have B_{i+1}^h neighbours in L_{i+2}
      Let $W_1...W_x$ be all possible combinations of vertices with one chosen from each of $V_j^1...V_j^k$
      for $W_l \in W_1...W_x$ **do**
        make list mapping $\mathcal{P}_l'$ for vertices in L_{i+1} such that all entries in $\mathcal{P}_l'$ are the same as in $\mathcal{P}$ except for the vertices in W_l
        for vertex $v \in W_l$ where v is in V_{i+1}^j **do**
          set v's list in P_l' to the single colour C_j
        end for
      end for
      for each list mapping $\mathcal{P}_l'$ in $\mathcal{P}_0'...\mathcal{P}_x'$ **do**
        for each vertex v in L_{i+1} **do**
          Let h be the position of v in a sort of vertices on L_{i+1} by nonincreasing neighbourhood in L_i
          remove from v's list in $\mathcal{P}_l'$ all colours C_j such that $h > B_i^j$
        end for
      end for
      for list mapping $\mathcal{P}_l'$ in $\mathcal{P}_0'...\mathcal{P}_x'$ **do**
        boolean hasAColouring $\leftarrow$ listColouring($G[L_{i+1}]$, $\mathcal{P}_l'$, $k - 1$)
        if hasAColouring **then**
          add B_{i+1} to the list $Configurations_{i+1}$
          exit the for loop
        end if
      end for
    end for
  end for
  if $Configurations_{i+1}$ is empty **then**
    return false
  end if
end for
return true

Roughly, the recursive algorithm works as follows: first, if the input is a 2-list colouring instance, we solve it. We return true if there is a valid colouring, false if not. Otherwise, we produce a multi-chain ordering. We produce configurations for the first layer, as it consists of only a single vertex. For each layer, we then produce all good configurations given the good configurations for the previous layer, as outlined in the text. This is where the recursive call is made.

The recursion tree of this algorithm is of depth at most k. Overall, there will be $O(n^k)$ recursive calls. How much work is done in each call? We generate a multi-chain ordering in $O(n(n+m))$ time. Then for each layer we generate all possible configurations - there are $O(n^k)$ of these for each layer, and it takes constant time to generate each. For each possible configuration, we generate $O(n^k)$ list mappings. Therefore the overall complexity of the algorithm is $O(n^{3k})$

4.4 Conclusion

We have given a polynomial-time algorithm for determining whether a graph $G = (V, E)$ with list mapping $\mathcal{P}$ has a colouring that admits $\mathcal{P}$, if every induced subgraph of G has a multi-chain ordering and there is a fixed colour bound of k. Every induced subgraph of a permutation graph has a multi-chain ordering, so this algorithm will work for permutation graphs. Similarly, every induced subgraph of a interval graph has a multi-chain ordering, so this algorithm will also work for interval graphs.

The multi-chain ordering may be useful finding future polynomial-time algorithms for these classes.

Bibliography

[1] Milos Biro, Mihaly Hujter, and Zsolt Tuza. Precoloring extension. i. interval graphs. *Discrete Mathematics*, 100(1):267–279, 1992.

[2] Pinar Heggernes, Dieter Kratsch, and Daniel Meister. Bandwidth of bipartite permutation graphs in polynomial time. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, pages 216–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] Mihaly Hujter and Zsolt Tuza. Precoloring extension 3: Classes of perfect graphs. *Combinatorics, Probability & Computing*, pages 35–56, 1996.

[4] Klaus Jansen and Petra Scheffler. Generalized coloring for tree-like graphs. *Discrete Applied Mathematics*, 75(2):135 – 155, 1997.

[5] T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, NY, USA, 1994.

[6] Jan Kratochvil. Precoloring extension with fixed color bound. *Acta Math. Univ. Comen.*, 62:139–153, 1994.

[7] Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189 – 241, 1999.

[8] Amir Pnueli, Abraham Lempel, and Shimon Even. Transitive orientation of graphs and identification of permutation graphs. *Canadian Journal of Mathematics*, 23:160–175, 1971.

Chapter 5

Set representation games[1]

5.1 Introduction

Combinatorial games played on graphs provide an area of active and exciting research. Node-Kayles, which we will refer to as Kayles, is an extensively-studied game played on a graph [3, 6, 10] and is a generalization of a game in which players knock over pins in a row. In a game of Kayles, players alternate turns, choosing vertices of the graph. A player may not choose a vertex already chosen, or a vertex adjacent to a vertex that has already been chosen. The last player to make a legal move wins (the *normal play* condition).

While solving an arbitrary Kayles position is PSPACE-complete [10], positions on trees with only one vertex of degree greater than two [6], graphs with bounded asteroidal number [3], and cocomparability, permutation, interval, and circular arc graphs [2] can be solved in polynomial time. The complexity of resolving Kayles positions on trees remains open.

In this paper, we describe games using set representations of graphs. These games generalise Kayles.

Sets have relationships with each other. Sets may intersect, overlap, contain each other, or they may be disjoint. One way to capture these interactions is with a graph. We can create one vertex for each set, and make two vertices adjacent if their corresponding sets have a relationship that we specify. We call the sets a set representation of this graph.

Many graph classes have characterizations based on particular types of set representations, including permutation graphs [11], interval graphs [9], chordal graphs [7], and filament graphs [8].

A *combinatorial game* is a two player game with no chance or hidden information.

In this work, we deal with *impartial* combinatorial games - games in which from a given position the same moves are available to either player.

We will use the terms *game* and *position* through this work. It is common in the literature to use the term *game* both to mean a set of rules and the underlying structure used to play, as in the case of *chess* or *hex* being a game, as well as to mean a particular position of a game. We will use it only

[1] This is joint work with Lorna Stewart, submitted to the Integers Journal on August 31, 2011

in the first sense. We will not explicitly use the formal definition of a game as a recursive set of all games reachable from that game in one move.

A game position S' is *reachable* from game position S if there is a legal sequence of moves from S that results in S'.

A *winning move* is a move that, with perfect play, will lead to a win for the player making that move. A *losing move* is a move that, with perfect play, will lead to a loss for the player making the move. A *winning position* or $\mathcal{N}$*-position*, is a game position from which there is a winning move. A *losing position* or $\mathcal{P}$*-position* is a game position from which there is no winning move.

A *end position* is a game position from which there are no legal moves. The *outcome class* of a game position is who will win with perfect play. An *instance* of a game is a position of that game, along with the question: *"Is there a winning move for the next player from this position"*.

There are a couple of special games that we will use. The game $0 = \{|\}$ is the game in which there are no legal moves for either player, and so is a $\mathcal{P}$-position and an end position. The game $* = \{0|0\}$ is the game in which the next player can move the game to a 0 game, and so is a $\mathcal{N}$ position.

Nim is a classic combinatorial game in which players take turns removing any number of stones from any one of a number of piles of stones. The last player to make a legal move (that is, take a stone) wins. Nim has been very important to the analysis of impartial combinatorial games.

We will use the notions of game addition and nim-sum as described by Sprague-Grundy theory and Bouton [4], in particular the following two theorems:

Theorem 10. *A nim position is a win for the previous player (and a loss for the next player) if and only if the nim-sum of its components (the number of stones in each of the piles) is zero.*

Two games F and G are *equivalent* if for every other game H, the outcome class of $F + H$ is the same as the outcome class of $G + H$.

Theorem 11. *Every impartial combinatorial game position is equivalent to a nim position.*

The number of stones in the single-pile nim position that is equivalent to a position G is called the *nimber* or *nim-value* of G. So, for example, the 0 position has a nimber of 0, and the position $*$ has nimber 1.

The *minimum excluded value* of a set of nimbers is the smallest nimber not in that set. Sprague-Grundy theory tells us that the nimber of a position is the minimum excluded value of the nimbers of its successor positions. For example, consider a game G in which the only move takes us to $*$. The nimber of $*$ is 1, and the minimum excluded value of the set $\{1\}$ is 0. Therefore the nimber of the G is 0.

For each position of a game, the nimber can be calculated in time linear in the number of successor positions. Then, given a game tree completely enumerating all positions of a game, we can

calculate the nimber of the initial position. Sometimes the nim-sum of k positions of $*$ is abbreviated as $k*$.

In this paper we describe several combinatorial games on graphs, all of which generalise Kayles.

We define the *set representation game*, in which players pick sets from a provided pool to build a set representation of a given graph. At each step, the set chosen must be consistent with the set representation already partially built. The last player to make a legal move wins.

While showing that resolving positions of these games is, in general, PSPACE-hard, we give algorithms for resolving the outcomes of positions of cases of these games in polynomial time.

5.2 Definitions and Notation

A graph G is a pair (V, E) such that V is a set and is called the *vertex set* of G, and E is a set of subsets of size two of V and is called the *edge set* of G.

Let S be a set of sets. We say that a set family $\mathcal{S}$ is a subfamily of S if every element of $\mathcal{S}$ is an element of S. Note that this is not quite the same as a subset because $\mathcal{S}$ may contain the same element more than once.

Let $G = (V, E)$ be a graph. A *set representation* of G is a tuple $(S, \mathcal{S}, f, \Phi)$ where:

- S is a set of sets
- $\mathcal{S}$ is a subfamily of S
- f is a bijection from V to $\mathcal{S}$
- Φ is a set relationship
- for every pair of vertices $v_i, v_j \in V$, v_i is adjacent to v_j if and only if $f(v_i)\Phi f(v_j)$

For convenience, when we are only talking about a particular type of set relationship, we may drop the full notation and not explicitly refer to f. For example, we may call a subfamily of S and S itself an intersection representation of a graph without referring to $(S, \mathcal{S}, f, \cap)$. We call S the *host* of the representation.

Let $G = (V, E)$ be a graph. A *graph representation* of G is a tuple $(H, \mathcal{H}, f, \Phi)$ where:

- H is a graph
- $\mathcal{H}$ is a family of subgraphs of H
- f is a bijection from V to $\mathcal{H}$
- Φ is a graph relationship (for example, subtree intersection)
- for every pair of vertices $v_i, v_j \in V$, v_i is adjacent to v_j if and only if $f(v_i)\Phi f(v_j)$

For convenience, when we are only talking about a particular type of graph relationship, we may drop the full notation and not explicitly refer to f. That is, we may call a family of subgraphs of H and H itself an intersection representation of a graph without referfing to $(H, \mathcal{H}, f, \cap)$. We call H the *host* of the representation.

5.3 Set representation games - Selecting from a given set of objects

Let S be a set of sets, Φ a set relationship that is a symmetric binary relation on S, and $G = (V, E)$ a graph.

We define a game based on these. Informally: on his turn a player selects a vertex v of G and a set s in S and assigns that set to v such that for every already-selected vertex u assigned set s_u, $s \Phi s_u$ if and only if v and u are adjacent in G. The last player with a legal move wins.

A position of this set representation game comprises a set representation $(S, \mathcal{S}, f, \Phi)$ of a subgraph of G. An instance of the game is a position of the game along with the question: is this an $\mathcal{N}$-position?

More precisely, a game position of the S-set representation game using graph $G = (V, E)$ with set relationship Φ is a tuple $(\mathcal{Q}, V', f)$ where:

- $\mathcal{Q}$ is a subfamily of S
- V' is a subset of V
- and f is a bijection from V' to $\mathcal{Q}$ such that $(S, \mathcal{Q}, f, \Phi)$ is a set representation of $G[V']$

A legal move of the S-set representation game using $G = (V, E)$ with set relationship Φ is the production of a position $S_2 = (\mathcal{Q}_2, V_2', f_2)$ from position $S_1 = (\mathcal{Q}_1, V_1', f_1)$ such that there is a single vertex v in $V_2' \backslash V_1'$, a single set s in $\mathcal{Q}_2 \backslash \mathcal{Q}_1$, $f_2(v) = s$, and:

- $V_2' = V_1' \cup \{v\}$
- $\mathcal{Q}_2 = \mathcal{Q}_1 \cup \{s\}$
- $f_2 = f_1 \cup \{(v, s)\}$

When we restrict the set S and the set relationship Φ, we may redefine the game positions slightly for compactness. Also, we will often omit Φ in the name of set representation games. Unless stated otherwise, assume that we are playing the games with intersection.

Separability

Let S be a set and $\mathcal{Q}$ a subset of S, Φ a set relationship, and d a function from the elements of $\mathcal{Q}$ to $\{true, false\}$. A set $s \in S$ is *consistent* with $\mathcal{Q}$, d if s is related by Φ to exactly the sets in $\mathcal{Q}$ for which d yields $true$.

Let S be a set, Φ a set relationship. We say that S, Φ are *separable* if for every natural number k we can identify in time polynomial in k: k disjoint subsets $S_1 ... S_k$ of S and a function $d_i : S_i \rightarrow \{true, false\}$ for each i such that for each set S_i where $1 \leq i \leq k$ there is at least one set in S that is consistent with S_i, d_i and for every i, j where $1 \leq i < j \leq k$ no set in S consistent with S_i, d_i is Φ with any set in S that is consistent with S_j, d_j. $S_1 ... S_k$ are said to be *separating* for S and Φ.

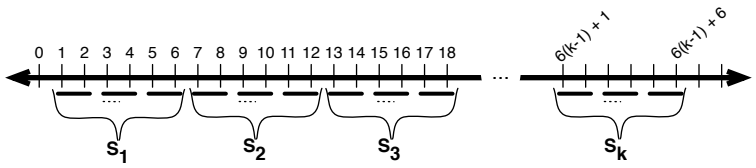

Figure 5.1: An example of k separating sets for intervals on the real number line and intersection. The solid black intervals are the members of the separating sets, the dotted intervals are examples of sets that are consistent with the functions specified in the text and the separating sets.

We now present an example of a separable family of sets and set relationship. Let the family of sets S be intervals on a real number line, and the set relationship Φ be intersection.

We show how to produce, in time polynomial in k, k separating sets with functions as required, for any natural number k. Let the set S_i for $1 \leq i \leq k$ be the set of three intervals t_i^1, t_i^2, t_i^3 defined by:

- t_i^1 spans $6(i-1)+1$ to $6(i-1)+2$
- t_i^2 spans $6(i-1)+3$ to $6(i-1)+4$
- t_i^3 spans $6(i-1)+5$ to $6(i-1)+6$

Let the function $d_i : S_i \rightarrow \{true, false\}$ yield :

- $d_i(t_i^1) = false$
- $d_i(t_i^2) = true$
- $d_i(t_i^3) = false$

We claim that: for each i where $1 \leq i \leq k$ there is at least one set in S that is consistent with S_i, d_i, and for every two i, j where $1 \leq i < j \leq k$ no set in S consistent with S_i, d_i that intersects any set in S that is consistent with S_j, d_j.

For each set S_i where $1 \leq i \leq k$, the interval in S that contains only $6(i-1)+3$ is consistent with S_i, d_i, as it intersects only the set t_i^2, and no other set in $(\cup S_j$ for $1 \leq j \leq k)$. For an example of this set construction, see Figure 5.1.

Let s_i be an interval that is consistent with S_i, d_i, and s_j an interval that is consistent with S_j, d_j.

We claim that $s_i | s_j$. We proceed by contradiction. Without loss of generality assume that $i < j$.

Then, because $s_i | t_i^3$ and s_i intersects t_i^2, the highest point in s_i strictly lower than $6(i-1)+5$. Because $s_j | t_j^1$ and s_j intersects t_j^2, the lowest point in s_j is at least $6(j-1)+2$. Since $6(i-1)+5 < 6(j-1)+2$, we have that s_i does not intersect s_j.

Therefore:

Lemma 30. *The family of intervals on the real line and the set relationship* intersection *are separable.*

Note that the construction of intervals above could easily be slightly modified so as to only use proper intervals. Then we also have that:

Lemma 31. *The family of proper intervals on the real line and the set relationship* intersection *are separable.*

5.3.1 PSPACE Hardness of the Set Representation Game

We provide a reduction from an arbitrary instance of Kayles to an instance of the set representation game on a S, Φ pair that are separable.

Let $G = (V, E)$ be a graph used in an instance of Kayles, and $U \subset V$ the set of vertices already selected.

For any separable family of sets S and set relationship Φ, we describe the production of a graph $G' = (V', E')$ that, along with an assignment f of some sets in S to some vertices in V' produces an instance of the set representation game that is next-player win if and only if G, U is a next-player win instance of Kayles.

First, because S, Φ are separable, we know that we can produce an arbitrary number of separating subsets of S. Let $S_1....S_n$ be $n = |V|$ separating subsets, with the functions d_i as specified in the definition of *separable* associated with each. We associate one of these subsets with each vertex $v_1, ..., v_n$ in V, with subset S_i associated with vertex v_i. We now construct the graph G' and a set representation for part of it.

Let V' be the union of V and new vertices created for each vertex $v \in V$.

Let $S_i = \{t_i^1 ... t_i^p\}$ be the subset associated with vertex v_i, and $p = |S_i|$ the size of S_i. Then we create p new vertices $v_i^1 ... v_i^p$ in V'. Let $f(v_i^j)$ be t_i^j.

If $v_i \in U$, then let $f(v_i)$ be any set consistent with S_i, d_i, the existence of which is guaranteed by the definition of separating. If v_i is not in U, we add no entry for $f(v_i)$.

Let edge set F_i contain exactly the edges between v_i and each one of the new p vertices for which d_i yields $true$.

Let s be any set in S that is consistent with d_i, S_i. Let edge set D_i contain exactly edges between v_i and each vertex v_h corresponding to a set s_h in $\cup S_j$ where $1 \leq j \leq k, j \neq i$ and $s_h \Phi s$.

Let edge set C contain an edge between every pair of vertices $v_i, v_j \in V'$ that are assigned sets by f such that $f(v_i) \Phi f(v_j)$. Let edge set E' be the union of C, E, and the edge sets D_i, F_i created for each vertex $v_i \in V$.

Then $G' = (V', E')$, S, Φ and f form an instance of the set representation game. Note that, by the construction of the instance, any vertex of V' that is not assigned a set is an element of V.

Given this reduction, we have the following:

Lemma 32. *A vertex $v \in V \subset V'$ can be picked and assigned a set by the next player in the set representation game using $G' = (V', E')$, in position $(\mathcal{S}, V' \backslash V, f)$ if and only if v can be picked by the next player in the Kayles game G, U.*

Proof. We proceed by contradiction. Assume that there is a legal set assignment s_i to vertex v_i, even though one of its neighbours v_j has already been selected and assigned a set s_j from S.

Then $s_i \Phi s_j$. However, s_i is consistent with S_i, d_i and s_j is consistent with S_j, d_j. This is a contradiction because $S_1 ... S_n$ are separating.

Assume that no neighbours in V of vertex v_i have been previously selected, but there is no set in S that can be assigned to v_i legally. This is a contradiction, because as part of the definition of separating, we required that there exists a set in $S \backslash (\cup S_j$ for $1 \leq j \leq k)$ that is consistent with S_i, d_i.

This holds not only in the set representation game position that we produce from a Kayles position, but also at any subsequent set representation game position produced by legal play. □ □

Notice that this is exactly the same constraint that would apply in a game of Kayles on $G = (V, E)$. That is, a vertex can be selected if and only if none of its neighbours have been previously selected.

It remains to show that there is a next-player win strategy for the constructed set representation instance if and only if there is a next-player win strategy for the source Kayles instance.

Let $S_0 = (\mathcal{S}_0, W_0, f_0)$ be an end position of a game tree of the set representation game using G' started at $(\mathcal{S}, W, f)$. Then by Lemma 32 the Kayles position G, W_0 is also an end position of that game.

Assume the outcome class of every position $S_k = (\mathcal{S}_k, W_k, f_k)$ of the game tree started at $(\mathcal{S}, W, f)$, that is of distance at most k from every leaf of the game tree below it, is the same as the outcome class of G, W_k as a position of Kayles. Then consider a position $S_{k+1} = (\mathcal{S}_{k+1}, W_{k+1}, f_{k+1})$ in the game tree started at $(\mathcal{S}, W, f)$ that is at most distance $k + 1$ from every leaf below it in the tree.

We claim that S_{k+1} is an $\mathcal{N}$-position if and only if G, W_{k+1} is an $\mathcal{N}$-position in Kayles. If S_{k+1} is an $\mathcal{N}$-position, then there is at least one child position of S_{k+1} that is a $\mathcal{P}$-position. By the inductive assumption, there is then at least one child position of G, W_{k+1} that is a $\mathcal{P}$-position, and so G, W_{k+1} is an $\mathcal{N}$-position. If S_{k+1} is a $\mathcal{P}$-position, then all child positions are $\mathcal{N}$-positions. By the inductive assumption, all child positions of G, W_{k+1} are $\mathcal{N}$-positions and therefore G, W_{k+1} is a $\mathcal{P}$-position.

Therefore there is a next-player win strategy for the constructed set representation instance if and only if there is a next-player win strategy for the source Kayles instance.

Then we have:

Theorem 12. *Resolving arbitrary positions of the set representation game using separable sets is PSPACE-hard.*

We present a specific case of the above reduction: a reduction from Kayles to the set representation game on intervals on the real line and the set relationship *intersection*.

Let $G = (V, E)$ be a graph, and $U \subset V, G$ an instance of Kayles on G, with U being the set of vertices already selected.

Let family of sets S be the intervals on the real line. We describe a graph $G' = (V', E')$, a family of sets $\mathcal{S}$, and a bijection f from a subset of V to $\mathcal{S}$ that form an $\mathcal{N}$-postion of the set representation game that is next-player win if and only if G, U is an $\mathcal{N}$-position of Kayles.

Let $S_1....S_n$ be $n = |V|$ separating subsets of S. Let S_1 be the union of all these S_i, $1 \leq i \leq n$. Let $S_i = \{t_i^1, t_i^2, t_i^3\}$ be the subset associated with vertex v_i. Then we create three new vertices v_i^1, v_i^2, v_i^3 in V'. Let $f(v_i^j)$ be t_i^j. If $v_i \in U$, then let $f(v_i)$ be the interval containing only the point $6(i-1)+3$. As in Section 5.3, this interval is consistent with S_i, d_i. Let S_2 be all these intervals.Let edge set F_i contain exactly edges between v_i and each vertex v_i^j for which $d_i(f(v_i^j)) = true$.

Let V' be the union of V and the three new vertices created for each vertex $v \in V$. Let edge set E' be the union of E and each edge set F_i. Then let $G' = (V', E')$ be our graph, $\mathcal{S} = S_1 \cup S_2$ our family of sets. $(\mathcal{S}, (V' \backslash V) \cup U, f)$ is a position of the set representation game on intervals.

As in the proof of Theorem 12, our position of the set representation game on intervals is an $\mathcal{N}$-position if and only if our Kayles position is.

Given Lemma 31, we have as a corollary to Theorem 12:

Corollary 4. *Resolving arbitrary positions of the set representation game on intervals on the real line with the set relationship* intersection *is PSPACE-hard.*

Permutation Graph Game

Permutation graphs are a well-studied intersection class. They are comparability cocomparability graphs [9]. They are also intersection graphs of line segments between two parallel lines and containment graphs of intervals on a line.

Given the amount of study on permutation representations, we consider the set representation game on a permutation representation.

Let the set S be all line segments between two infinite parallel lines I_1, I_2 indexed with the reals, and the set relationship Φ be intersection. We claim that S, Φ are separable.

Given a natural number k, we can produce in time polynomial in k, $S_1...S_k$ disjoint subsets of S and a function $d_i : S_i \to \{true, false\}$ for each $i, 1 \leq i \leq k$, such that the required conditions for separability hold.

Let the set S_i for $1 \leq i \leq k$ be the set of three line segments t_i^1, t_i^2, t_i^3 defined by:

- t_i^1 connects $6i$ on I_1 to $6i$ on I_2
- t_i^2 connects $6i + 2$ on I_1 to $6i + 2$ on I_2
- t_i^3 connects $6i + 4$ on I_1 to $6i + 4$ on I_2

Let the function $d_i : S_i \to \{true, false\}$ yield:

- $f_i(t_i^1) = false$

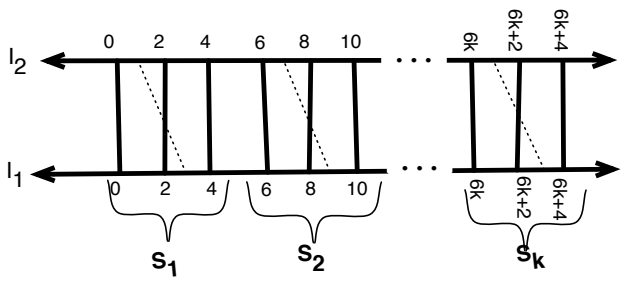

Figure 5.2: An example of k separating sets of line segments between two lines and the set relationship *intersection*. The solid black line segments are the members of the separating sets, the dotted line segments are examples of sets that are consistent with the functions specified in the text and the separating sets.

- $f_i(t_i^2) = true$
- $f_i(t_i^3) = false$

For each set S_i where $1 \leq i \leq k$ there is at least one set in S that is consistent with S_i, d_i: the line segment that connects $6i + 1$ on I_1 to $6i + 3$ on I_2.

For an example of this set construction, see Figure 5.2. Let s_i be a line segment that is consistent with S_i, d_i, and s_j a line segment that is consistent with S_j, d_j such that $i < j$.

Because $s_i | t_i^3$ and s_i intersects t_i^2, the highest endpoint of s_i on either of I_1, I_2 is at most $6i + 4$. Because $s_j | t_j^1$ and s_j intersects t_j^2, the lowest endpoint s_i on either of I_1, I_2 is at least $6j$. Since $6i + 4 < 6j$, s_i does not intersect s_j.

Therefore:

Lemma 33. *The family of line segments between two infinite lines and the set relationship* intersection *are separable.*

Then as a corollary to Theorem 12:

Corollary 5. *Resolving arbitrary positions of the set representation game on line segments between two infinite lines with the set relationship* intersection *is PSPACE-hard.*

Containment and Overlap Games

So far, we have played the set representation game only using intersection as the set relationship. In this section we consider playing the game with the set relationships *containment* and *overlapping*.

Lemma 34. *Set S and the set relationship* containment *are separating if there are in S an arbitrary number of pairs of sets such that:*

- *one of each pair is contained in the other,*
- *each set in a pair is disjoint from all sets in other pairs*

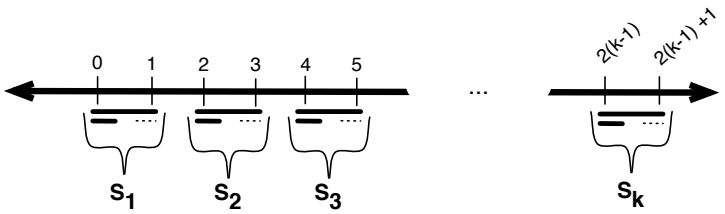

Figure 5.3: An example of k separating sets of intervals on the real number line with the set relationship *containment*. The solid black intervals are the members of the separating sets, the dotted intervals are examples of sets that are consistent with the functions specified in the text and the separating sets.

- *and for each pair $P_i = (p_i^1, p_i^2)$ there is a set in S that is not in any pair and is contained in only the maximal one of p_i^1, p_i^2.*

Proof. Let $P_1...P_k$ be an arbitrary number of disjoint pairs. We denote the two elements of $P_i, 1 \leq i \leq k$ as (p_i^1, p_i^2). For convenience, let $p_i^1 \subset p_i^2$. We will use $P_1...P_k$ as our disjoint subsets of S.

For pair P_i, let $d_i(p_i^1) = false$, $d_i(p_i^2) = true$.

For every pair P_i there is at least one set that is consistent with P_i, d_i. This follows from the requirement that for each pair (p_i^1, p_i^2) there is a set s in S that is not in any pair and is contained in only the maximal one of p_i^1, p_i^2.

We also claim that for every two pairs P_i, P_j, every set s_i consistent with P_i, d_i is disjoint from every set s_j consistent with P_j, d_j.

Set s_i must either contain or be contained in p_i^2, but must not contain p_i^1, as it is consistent with d_i. Then s_i must be contained in p_i^2, as $p_i^1 \subset p_i^2$. Following the same reasoning, s_j must be contained in p_j^2. Because p_i^2 is disjoint from p_j^2, set s_i is disjoint from set s_j. No set consistent with P_i, d_i contains or is contained in any set consistent with P_j, d_j.

We have shown that $P_1...P_k$, and our collection of d_i are separating for S and the set relationship *overlapping*. □ □

The set of intervals on the line and *containment* satisfy this lemma. We can produce the required pairs of intervals as follows: Let the pair P_i be composed of p_i^1 and p_i^2 where p_i^1 is the interval spanning only $2(i-1)$, and p_i^2 is the interval spanning $2(i-1)$ to $2(i-1)+1$. Observe that $p_i^1 \subset p_i^2$, and for every two P_i, P_j the sets in P_i are disjoint from the sets in P_j. Let $P_1...P_k$ be k of these pairs. Let the function $d_i(p_i^1) = false$ and $d_i(p_i^2) = true$.

For every P_i, every interval that is consistent with P_i, d_i is contained in p_i^2, and there is at least one interval that is consistent with P_i, d_i - the interval spanning only $2(i-1)+1$.

Because p_i^2 is disjoint from every set in every pair P_j for $j \neq i$, every set consistent with P_i, d_i is disjoint from every (and therefore not contained in any) set consistent with P_j, d_j for $i \neq j$.

For an example of this construction, see Figure 5.3.

Then as a corollary to Theorem 12:

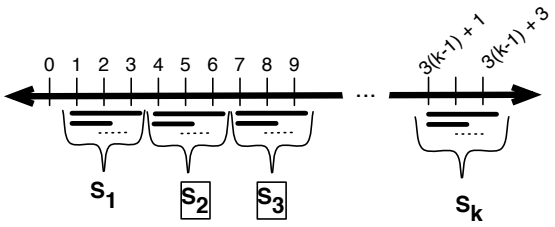

Figure 5.4: An example of k separating sets of intervals on the real number line under the set relationship *overlapping*. The solid black intervals are the members of the separating sets, the dotted intervals are examples of sets that are consistent with the functions specified in the text and the separating sets.

Corollary 6. *Resolving arbitrary positions of the set representation game on intervals on the real line with the set relationship* containment *is PSPACE-hard.*

Lemma 35. *Set family S and the set relationship* overlapping *are separable if there are in S an arbitrary number of disjoint pairs of sets such that:*

- *one of each pair is contained in the other,*
- *each set in a pair is disjoint from all sets in other pairs*
- *and for each pair $P_i = (p_i^1, p_i^2)$ there is a set in S that overlaps the minimal one of p_i^1, p_i^2, and is contained in the maximal one of p_i^1, p_i^2.*

Proof. Let $P_1 ... P_k$ be the arbitrary number of disjoint pairs. We will denote the two elements of P_i as (p_i^1, p_i^2). For convenience, let $p_i^1 \subset p_i^2$. We will use $P_1 ... P_k$ as our disjoint subsets of S.

For pair P_i, let $d_i(p_i^1) = true$, $d_i(p_i^2) = false$.

For every pair P_i there is at least one set that is consistent with P_i, d_i. This follows from the requirement that for each pair (p_i^1, p_i^2) there is a set s in S that overlaps the minimal one of p_i^1, p_i^2, and is contained in the maximal one of p_i^1, p_i^2, and the fact that sets in different pairs are disjoint.

We also claim that for every two pairs $P_i, P_j, i \neq j$, every set s_i consistent with P_i, d_i is disjoint from every set s_j consistent with P_j, d_j. Set s_i overlaps p_i^1, but not p_i^2, as s_i is consistent with d_i. Because $p_i^1 \subset p_i^2$, then $s_i \subseteq p_i^2$. By the same reasoning, s_j must be contained in p_j^2. Then, because p_i^2 is disjoint from p_j^2, set s_i is disjoint from set s_j. No set consistent with P_i, d_i overlaps any set consistent with P_i, d_i.

$P_1 ... P_k$ are separating, therefore S and the set relationship *containment* are separable. □ □

Figure 5.4 provides a graphical example of this overlapping lemma as applied to the intervals on the real line.

As shown in Figure 5.4 intervals on the real number line combined with the set relationship *overlapping* are separable. Then as a corollary to Theorem 12:

Corollary 7. *Resolving arbitrary positions of the set representation game on intervals on the real line with the set relationship* overlapping *is PSPACE-hard.*

5.3.2 Intervals and Permutation in PSPACE

In this section, we prove that resolving the interval and permutation games are in PSPACE, and are therefore PSPACE-complete.

We first observe that if A is a game of size k such that from any legal position of A the number of possible moves is polynomial in k, from any position of A the game is guaranteed to end after a number of moves polynomial in k, and every game position can be represented in space polynomial in k, then solving an arbitrary position of A is in PSPACE. We can justify this by considering a backtracking search for this game. Each position is polynomial in size to store, and each recursion branch is at most of polynomial depth. Therefore an arbitrary game position can be solved in space polynomial in k.

Lemma 36. *The set representation game played on intervals on the real line with any one of the set relationships* intersection, disjointness, overlapping, containment *is in PSPACE.*

Proof. We provide a polynomial-space reduction from the set representation game on intervals with the relationship *intersection* to an endpoint-ordering game that we show is in PSPACE.

$S_{EP} = (L, V', f_l, f_r)$ is a position of the *endpoint-ordering game* using graph $G = (V, E)$ with Φ where:

- L is an ordered set of endpoints
- $V' \subset V$
- f_l and f_r are functions from V' to the elements of L such that
  - the ranges of f_l andf_r partition the set of all elements of L and
  - for every vertex $v \in V'$ $f_l(v)$ occurs before $f_r(v)$ in L
  - for every two vertices $u, v \in V'$ the order of $f_l(v), f_r(v), f_l(u), f_r(u)$ in L is such that if they were placed in that order on the real line, the interval described by $[f_l(v), f_r(v)]$ would be Φ with the interval described by $[f_l(u), f_r(u)]$.

In this game the players alternate turns, and on a player's turn, he choses an unselected vertex v_i in V and adds two entries to L setting $f_l(v_i)$ to one of them and $f_r(v_i)$ to the other.

Let $G = (V, E)$ be a graph, S the set of all finite intervals on the real line, Φ one of the set relationships *intersection, disjointness, overlapping, containment*, and f an assignment of members of S to some members of V. Let $\mathcal{S}$ be the subfamily of S containing the intervals assigned to vertices in V. $G, \Phi, \mathcal{S}$ are an instance of the set representation game on intervals on the real line. We give a reduction from this instance to an instance of the endpoint-ordering game.

Let L be a list. For each vertex v_i in V already assigned an interval by f, we add v_i^1 and v_i^2 to L where v_i^1 is the left endpoint of the interval assigned to v_i and v_i^2 is the right endpoint of the interval

assigned to v_i. We then sort L and amend f_l and f_r such that $f_l(v_i) = v_i^1$ and $f_r(v_i) = v_i^2$ for all v_i. Note that the order of the members of L will be the same as their order on the real line.

Lemma 37. *Let $S_A = (\mathcal{Q}_A, V', f_A)$ be a position of the interval representation game, and $S_{EP} = (L, V', f_l, f_r)$ its corresponding transformed position of the endpoint ordering game. Then S_A is an $\mathcal{N}-$position if and only if S_{EP} is.*

Proof. Let $T_A = V_A, E_A$ be the directed acyclic graph that gives the game tree for the interval representation game played from S_A. We proceed by induction on the maximum distance of a position from a leaf of T_A.

Let S_A be a leaf position of T_A. Then there are no legal moves in the interval representation game from S_A, or from the endpoint-ordering transformation of it.

Assume that for every position S_A that is at most k moves from a leaf in T_A, S_A is an $\mathcal{N}-$position if and only if its endpoint-ordering transformed version S_{EP} is.

Now let $S_A = (\mathcal{Q}_A, V', f_A)$ be a position at most $k+1$ moves from a leaf of T_A, and let $S_{EP} = (L, V', f_l, f_r)$ be the transformation of S_A into the endpoint-ordering game.

Assume that S_A is an $\mathcal{N}$-position. Then there is a winning move from S_A to position S_B. But then note that the endpoint-ordering game transformation of S_B is a child position of S_{EP}, and by the inductive assumption is a $\mathcal{P}$-position. Therefore S_{EP} is an $\mathcal{N}$-position.

Assume that S_{EP} is an $\mathcal{N}$-position. Then there is a child position of S_{EP} that is a $\mathcal{P}$-position. Let $S_{EP2} = (L_2, V_2', f_{l2}, f_{r2})$ be this position. Let v be the single vertex in $V_2' \backslash V'$. Then L_2 differs from L only by the inclusion of v^1, v^2. Then there is a child position of S_A with v represented such that the endpoint-ordering transformation of that position is S_{EP2}. That child position is a $\mathcal{P}$-position, and therefore S_A is an $\mathcal{N}$-position.

□

Let $S_A = (\mathcal{Q}_A, V', f_A)$ and $S_B = (\mathcal{Q}_B, V', f_B)$ be two positions of the interval representation game. We say that S_A and S_B are *endpoint-equivalent* if they have the same transformed endpoint-ordering game position.

Corollary 8. *Let $S_A = (\mathcal{Q}_A, V', f_A)$ and $S_B = (\mathcal{Q}_B, V', f_B)$ be two endpoint-equivalent positions of the interval representation game. Then S_A is an $\mathcal{N}-$position if and only if S_B is.*

Recall that the endpoint ordering game with set relationship *intersection, disjointness, overlapping*, or *containment* is in PSPACE, because it can be solved with recursive game search.

Then, the set representation game played on intervals on the real line with any one of the the set relationships *intersection, disjointness, overlapping, containment* is in PSPACE. □

Consider a restriction of the interval containment game in which we specify a point z on the line and require that all intervals must include z. We call this game the interval containment game with

an equator, and call z the equator. This game is also in PSPACE because the interval containment game is.

Lemma 38. *The set representation game played on line segments between two parallel infinite lines with the set relationship* intersection *is in PSPACE.*

Proof. We prove a polynomial-space reduction from the set representation game on line segments between parallel lines with the relationship $intersection$ to the interval overlap game with an equator. Permutation graphs are exactly overlap graphs of intervals on a line with an equator as shown by Dushnik and Miller [5], so this reduction is not surprising.

Let $G = (V, E)$ be a graph, S the set of all line segments between two parallel lines I_1 and I_2, $\mathcal{S}$ a subfamily of S, and f a bijection from $V' \subset V$ to $\mathcal{S}$. $(\mathcal{S}, V', f)$ is a position of the permutation set representation game.

We give a reduction from this position to a position of the interval overlap set representation game. Let vertex $v \in V'$ and let $f(v)$ be a line with endpoints at x_1^v on I_1 and x_2^v on I_2 by f. Let x_{max1} be the largest-valued endpoint of a line on l_1 corresponding to a vertex. Let x_{max2} be the largest-valued endpoint of a line on l_2 corresponding to a vertex. Without loss of generality, assume that no endpoint on l_2 is placed at 0. Then let function $f_{intervals}$ assign v the interval from $[x_1^v, x_{max1} + (x_{max2} - x_2^v) + 1]$ on line I. Let $\mathcal{S}_{interval}$ be all these intervals, and $S_{interval}$ be the set of all intervals on I.

$(\mathcal{S}_{interval}, V', f_{interval})$ with equator x_{max1} form an position of the interval overlap set representation game with an equator.

Suppose that the next player has a winning strategy for the permutation set representation instance. If the permutation player would choose vertex v_i and assign it endpoints x on I_1 and y on I_2, then in the interval overlap game they choose v_i and assign it interval $[x, x_{max} + y]$ on line I. Applied inductively, this gives a strategy. A similar argument applies if there is a previous-player winning strategy on the permutation set representation instance.

Because solving the interval overlap game with an equator is in PSPACE, solving the permutation set representation game is. □ □

Combining the previous two Lemmas with Corollaries 4, 5, 6 and 7, we have:

Theorem 13. *The set representation game played on line segments between two parallel lines with* intersection*, as well as on intervals on the real line with any of* intersection, containment, disjointness, overlapping *is PSPACE-complete.*

5.4 Solving some connected representation games

We have given hardness results for the new types of games we've defined. When can we solve an initial position of these games? We give some special cases with algorithms for solving positions

in polynomial time. Our results are for the connected set representation game, and are all for when the game is played using trees. In this section, we will make use of nimbers to solve some of these games and their unions. As mentioned before, every impartial combinatorial game has a nimber, and is an $\mathcal{N}$-position if and only if that nimber is positive.

We will also use the idea of a fixed starting vertex for a game. What if we played the connected proper interval representation game using a tree $T = (V, E)$ but specified the first vertex u chosen, and specified that throughout play no interval may have any endpoint to the left of the left endpoint of the interval representing u? In this case we call u a *mandatory head*. We use a similar idea for the connected permutation game.

5.4.1 Connected proper interval representation game using a tree

Let $T = (V_T, E_T)$ be a tree, and $u \in V_T$ a mandatory head. Can we calculate the nimber of a game starting at u?

What do we know about the end positions of games that started with u as a mandatory head? First, we know that the vertices represented at the end of any game induce a path, because every proper interval graph that is a tree is a path. What about the endpoints of this path? If the end of the path that isn't u is not a leaf, then there are remaining moves, and the game is not over. Therefore one end of the path will be u, and the other will be a leaf of T.

Using these observations, we can calculate the nimber of a game starting at u. First we root T at u. Every game will end with a move that places a leaf of this rooted tree - no further moves are possible from a position with a leaf represented, and if there is no leaf represented then further moves are possible. Given a vertex v in T, we know exactly all moves previous to v given that u is the mandatory head, as there are unique paths in trees.

Recall that the minimum excluded ordinal of a set is the smallest ordinal that is not in that set. Then for each v the nimber of the game in which v has just been placed is the minimum excluded value of the nimbers of the games in which each of the children of v have been placed, by these positions being all the possible succesor positions, and the Sprague-Grundy theorem [1].

The nimber of the game after a leaf has been played is 0. Then a single leaf is the $*$ game and has nimber 1.

Given T and u we can then compute the nimber of the game position for each v in a bottom-up fashion using a postorder traversal of T.

Then, given a set of trees with a specified mandatory head in each, we can sum the nimbers to determine the overall nimber and therefore the winner of the union of these games.

We can use this to help us find the nimber of a game using a tree without a mandatory head. Let v be a vertex in tree $T = (V_T, E_T)$. Imagine if v were played as a first move in the connected proper interval representation game using a tree . If v is removed from T, we are left with a forest $\{T_1 ... T_k\}$ of subtrees of T. Because the eventual representation is a proper interval representation,

we know that vertices from two of $T_1...T_k$ will be represented. We also know that for each T_i the vertex u_i of T_i that is adjacent to v, if it is represented, will be represented with one endpoint strictly to the left or strictly to the right of all other intervals corresponding to vertices in T_i, and will be the first vertex of T_i to be represented. Then if u_i is played, we can treat it as a mandatory head with respect to T_i.

Then, vertex v is a winning first move for the first player if for every $T_i \in \{T_1...T_k\}$ there is a $T_j \neq T_i \in \{T_1...T_k\}$ such that the sum of the nimbers of u_i and u_j is zero.

We can use a method almost exactly the same as our method to solve normal play in the connected proper interval representation game using a tree when the first move is at a mandatory head to solve the misere version of the same game.

5.4.2 Improper Intervals

We now consider the set representation game with a mandatory head played using trees on intervals on a line.

A caterpillar is a tree such that if all leaves are removed we are left with a path. The spine of a caterpillar is that path.

Lemma 39. *In every position of every connected intervals using a tree game the represented vertices induce a caterpillar.*

Proof. This follows from the fact that all trees that are interval graphs are caterpillars. □

In an interval representation, let the line have an arbitrary left-right orientation. The *leftmost* interval is the one with an endpoint to the left of every other interval. The *rightmost* interval is the one with an endpoint to the right of every other interval. The leftmost and rightmost vertices are the vertices corresponding to these intervals.

Let u be the leftmost represented vertex in an interval representation of tree $G' = (V', E')$ and w the rightmost. G' is a caterpillar. We claim that the path between u and v is a spine of G'. First, observe that the interval from the leftmost point of u to the rightmost point of w contains all intervals in the representation. Now, observe that the union of the intervals representing vertices on the path from u to w, inclusive, contains that entire interval. Every vertex not on this path therefore is adjacent to at least one vertex on the path. Because G' is a tree, all vertices not on the path are therefore adjacent to exactly one vertex on that path, and not to any other vertices. Therefore the path is a spine of G'.

Definition 7. *The* placed-spine *of G is the path between the leftmost and rightmost represented vertices in V', inclusive.*

Lemma 40. *Let $V' \subset V$ be a represented subset in the connected intervals using a tree game that induces a caterpillar. Then, in the representation, an interval can be added to represent any vertex that is adjacent to a single vertex on that placed-spine.*

Proof. Let v_i be a vertex on the placed-spine of a caterpillar, corresponding to interval l_i. Because G is a tree, and because there is a mandatory head and therefore l_i is not contained in any other interval, there is a portion of l_i not contained in any other interval.

Then we can produce an interval small enough to fit in that portion that intersects only l_i, and so can add an interval to represent a leaf. □

Lemma 41. *A specification of the placed-spine of the caterpillar represented at the end of a connected intervals using a tree game is sufficient to allow us to calculate the winner of the game.*

Proof. By Lemma 40, at the end of the game all leaves attached to the specified spine will be represented. No other vertex can be added.Then the winner of the game is determined by the parity of the number of vertices in the spine plus the leaves. □

Let S be a game position of the connected interval representation game using a tree $T = (V_T, E_T)$ with mandatory head u. A represented vertex v is in the *spine-so-far* position if its interval is the rightmost interval (assuming that u is the leftmost).

Definition 8. *Let S be a game position of the interval game with mandatory head u and vertex v in the spine-so-far position. An* unrepresented leaf *is a leaf pendant from the path between u and v that is not yet represented in S.*

Let S be a game position of the connected interval representation game using a tree $T = (V_T, E_T)$ with mandatory head u and vertex v in the spine-so-far position. If a player chooses an unrepresented neighbour of v and represents it with a interval contained in the one assigned to v, we say that player has made a *foldunder* move. We add the restriction of forbidding foldunders in our game.

In the interval game, what information is actually important in a game position?

Let S_1 and S_2 be two positions of the interval game with the same mandatory head, the same spine-so-far vertex, and the same number of represented vertices. We claim that S_1 and S_2 are equivalent.

That is, it is only how many vertices are represented and not how they are represented or which vertices they are that is relevant to solving the game, with the exception of the mandatory head and spine-so-far vertices.

This follows from the fact that, by Lemma 40 at the end of the game all of the pendant leaves are represented. Each pendant leaf is effectively a $*$ in the game, as each can be played at any time, it doesn't matter how they are played, and they will all be played by the end of the game.

Then:

Lemma 42. *S_1 and S_2 have the same nimber.*

Game Graph for a Mandatory Head

An *abstracted game position* Q of the interval graph game using tree T with mandatory head u consists of a vertex v that is in the spine-so-far position and a number k indicating the number of unrepresented pendant leaves.

For a pair of vertices x, y let $leaves(x, y)$ give the number of leaves pendant from a path with endpoints x, y, excluding x and y even if they are leaves, but including the neighbours of x and y.

We use a graph, given a tree $T = (V, E)$ and mandatory head $u \in V$ to describe the entire game position space of abstracted positions. This graph is polynomial in size with respect to the size of T and can be produced in time polynomial in the size of T.

For each vertex $v \in V$ let W_v be a set of abstracted game positions $w = \{v, i\}$ for $0 \leq i \leq leaves(u, v)$. Abstract position $\{v, i\}$ represents a game position in which v is in the spine-so-far position, and i leaves pendant on the u to v path are unrepresented. Then let vertex set W consist of the union of W_v over all $v \in V$.

We define a directed edge set F as follows: directed edge $\{v_1, h\} \to \{v_2, k\}$ is in F if:

- $v_1 = v_2$ and $k = h - 1$, or
- v_1 and v_2 are adjacent in T, but v_2 is not on the path between v_1 and u and $k = leaves(u, v_2) - (leaves(u, v_1) - h)$

Then let $\overrightarrow{G} = (W, F)$.

These edges correspond to the transitions between abstract positions that we can make by playing moves in the game. A move either changes the vertex in the spine-so-far position or it doesn't. If it does not, then it represents another pendant leaf, decreasing the number of unrepresented pendant leaves.

We claim that $\overrightarrow{G}$ is a directed acyclic graph. Assume there is a cycle. Let $\{v_1, i\}$ be a position in the cycle such that v_1 is closest to u and, if there is more than one position in the cycle with the vertex at the same minimum distance to u, that i is largest of those. Then let $\{v_2, j\}$ be the position in the cycle such that there is an edge from $\{v_2, j\}$ to $\{v_1, i\}$. If $v_1 = v_2$, then by the construction of F, $i = j - 1$, a contradiction to the maximality of i. If $v_1 \neq v_2$, then v_1 is adjacent to v_2 but is not on the path from u to v_2, a contradiction to v_1 being closest to u. Therefore there is no such cycle.

We can, using this graph as a game tree (note that game trees are generally directed acyclic graphs, and not actually trees), calculate nimbers for the abstract game positions in a backtrack order - the reverse of a breadth-first search order started at $\{u, |N(u)|\}$. We start by assigning a nimber of 0 to every sink position $\{v, 0\}$, and then proceed to backtrack from there. We then calculate the nimber for each position by taking the minimum excluded value of the nimbers of its children. This will give us a nimber for the initial position of just the mandatory head being played, telling us the winner of the game, assuming no foldunders. We can use this nimber to determine the winner of unions of these games if we explicitly forbid foldunders.

5.4.3 Connected permutation representation game using a tree

Consider two infinite parallel lines running up and down. Let L be the set of all line segments between these two lines.

Consider a connected permutation game on graph $G = (V, E)$, with set V' containing the vertices already represented. As we will always play this game using the set relationship *intersection*, we omit that from the game position, so a position of this game played using tree $T = (V, E)$ will be expressed as: $(\mathcal{L}, V', f)$ where:

- $\mathcal{L}$ is a subfamily of L
- $V' \subset V$
- and f is a function from V' to $\mathcal{L}$ such that $(L, \mathcal{L}, f, \cap)$ is an intersection representation of $T[V']$.

Definition 9. *Let $(\mathcal{L}, V', f)$ be a position of the permutation game using tree $T = (V, E)$. The* lowest *vertices in V' are the vertices corresponding to lines that at some point have no other line below them in the representation.*

Definition 10. *Let $(\mathcal{L}, V', f)$ be a position of the permutation game using tree $T = (V, E)$. The* far-lowest *vertex in V' is the vertex assigned the line such that the endpoints of that line are geometrically below the endpoints of all lines that do not intersect that line, and only one neighbour of that line is represented. The* far-highest *vertex in V' is the vertex assigned the line such that the endpoints of that line are geometrically above the endpoints of all lines that do not intersect that line, and only one neighbour of that line is represented.*

These are unique in connected representations of trees with more than two vertices represented: assume that there are two far-lowest vertices in a representation. They must be adjacent, as their lines must intersect. At least one of them must have another neighbour, as the representation is connected. Then that one with another neighbour is not far-lowest. A similar argument shows that the far-highest vertex in a connected representation with more than two vertices represented is unique.

Say we are going to play the connected permutation representation game using a tree with the restriction that no line will have both endpoints above the endpoints of the first line placed, or, symmetrically, no line will have both endpoints below the endpoints of the first line placed. We then call that first vertex the *mandatory head*, and will also refer to its assigned set as the *mandatory head*.

We say that the vertex corresponding to the line that is far-lowest (if the mandatory head is far-highest) or far-highest (if the mandatory head is far-lowest) is in the *spine-so-far* position.

Lemma 43. *At every position of every connected permutation representation game using a tree, the represented vertices induce a caterpillar.*

Proof. This follows from the fact that all trees that are permutation graphs are caterpillars. □

Let $(\mathcal{L}, V', f)$ be a position of the connected permutation game using tree $T = (V, E)$. Let u be the far-highest vertex and w the far-lowest. We claim that the path from u to w, inclusive, is a spine of $T[V']$. Because u is the far-highest and w is the far-lowest vertex, we know that no line has both endpoints above $f(u)$ or below $f(w)$. Then every other line in $\mathcal{L}$ intersects the line of at least one of the vertices on the path between u and w. Then, because T is a tree, each vertex not on the path between u and w is adjacent to exactly one vertex on that path, and not to any other vertices in $T[V']$. Therefore that path is a spine of $T[V']$.

Definition 11. *The* placed-spine *of G is the path between the far-highest and far-lowest vertices in V', inclusive.*

Lemma 44. *Let $V' \subset V$ be a represented subset in the connected permutation game that induces a caterpillar. Then leaves can be added to anywhere on the placed spine in subsequent game positions.*

Proof. Let v_i be a vertex on the placed-spine of a caterpillar, corresponding to line l_i. Assume that we cannot place a line l_j that intersects only l_i. If there is already a represented leaf adjacent to v_i, then that line intersects only v_i, and we can produce another such line by joining left and right endpoints an arbitrarily small distance below the endpoints of that line.

Then assume that there are no such represented leaves. If v_i has two neighbours on the spine, then the only neighbours of v_i that are represented are its two neighbours on the spine, v_{i-1}, v_{i+1}. Because these two vertices are not adjacent, we know that their lines, l_{i-1} and l_{i+1}, do not intersect. Without loss of generality, assume that l_{i-1} is above l_{i+1}, and that the left endpoints of l_{i-1}, l_{i+1}. are above the left endpoint of l_i and the right endpoints of l_{i-1}, l_{i+1} below the right endpoint of l_i.

We claim that a line between endpoints just above the left endpoint of l_{i+1} and just below the right endpoint of l_{i-1} will intersect only l_i. Assume it does not. Then there is a line l_x with an endpoint above the right endpoint of l_{l-1} and below the left endpoint of l_{i+1}. Such a line must intersect both l_{i-1} and l_{i+1}, a contradiction.

If v_i has only on neighbour on the placed spine, then a line with endpoints just below or above endpoints of the line of that neighbour will intersect only l_i.

□

Lemma 45. *A specification of the placed-spine of the caterpillar represented at the end of a connected permutation game is sufficient to allow us to calculate the winner of the game.*

Proof. By Lemma 44, at the end of the game all leaves attached to the specified spine will be represented. No other vertex can be added, provided the spine is maximal (that is, its endpoints are leaves). Then the winner of the game is determined by the parity of the number of vertices in the spine plus the leaves. □

We will need an observation and a lemma about a player's ability to force certain vertices to be on the spine of the caterpillar:

Observation 9. *Let $T = (V, E)$ be a tree. Let S be a position of a connected permutation game using T with V' the represented vertices and $\mathcal{P}$ their assigned lines on the permutation diagram, with mandatory head u. Let v be a vertex in the spine-so-far position in S such that all neighbours of its neighbour in $T[V']$ are already represented in S, and are therefore all in V'. Then in every position S' reachable from S, vertex v is on the placed-spine.*

We will now diverge somewhat from our previous notation. We will be solving positions of the connected permutation game with respect to the first or second player, and not the next or previous player. This is a technical necessity: the next lemma cannot be directly reformed to use next and previous players. It can, however, be later adapted to give an answer in terms of the next or previous player given the parity of the number of previously represented vertices.

Lemma 46. *Let $T = (V, E)$ be a tree. Let S be a position of a connected permutation game using T with V' the represented vertices and $\mathcal{P}$ their assigned lines on the permutation diagram, with mandatory head u.*

Let vertex v be in the spine-so-far position, and the path between u and v be of length at least two. Let N_p be the neighbours of v that are not already represented.

Then we claim that:

- *We can partition N_p into N_p^f and N_p^s such that all games reachable from S that have a vertex from N_p^f in the spine are first-player wins, and all games reachable from S that have a vertex from N_p^s in the spine are second-player wins. We call N_p^f the win-set for the first player, and N_p^s the win-set for the second player.*
- *If $|N_p^f| > |N_p^s|$ there is a strategy for the first player to win from S,*
- *if $|N_p^f| < |N_p^s|$ there is a strategy for the second player to win from S, and*
- *if $|N_p^f| = |N_p^s|$ there is a strategy for the first player to win from S if and only if the parity of the number of vertices of the spine represented thus far and all leaves attached to that spine (represented and unrepresented) is odd.*

Proof. Root T at u. We will call a vertex not yet represented *unrepresented*. Let d be the distance to the furthest vertex from u in T. We proceed by induction on the distance from u. Consider nodes at distance d from u. They must be leaves. We can determine the winner of a game with u and a given leaf on the spine by calculating the parity of the maximal caterpillar with that spine. As these nodes have no children, we can vacuously partition all their children into N_p^s and N_p^f. Since $|N_p^s| = |N_p^f| = 0$, the lemma is proven in this case.

Assume that for every node w at distance k from u, we can partition its children that are not on the path between w and u as above, and the given rules hold for determining whether there is a winning strategy for the first or second player if w is on the spine.

Now consider a node v at distance $k-1$ from u. All children of v are at distance k from u, and therefore the rules given will determine whether there is a winning strategy for the first or second player if each of them is on the spine. So then if N_p are the neighbours of v that are not on the path from v to u, we can partition N_p into N_p^f and N_p^s such that all games reachable from S that have a vertex from N_p^f in the spine are first-player wins, and all games reachable from S that have a vertex from N_p^s in the spine are second-player wins.

We show how the player with the larger win-set can produce a game position in which a vertex in their win-set is in the spine-so-far position when all neighbours of v are represented, and therefore arrive at a game position that is a win for that player.

We argue the case in which the first player has the larger win-set, that is $|N_p^f| > |N_p^s|$. The opposite case requires a symmetric argument.

On the first player's next turn after v is placed in the spine-so-far position, he chooses any one of the vertices in N_p^f and places it in the spine-so-far position. On every subsequent turn, first player takes one of two actions until either all vertices in N_p are represented, or all vertices in N_p^s are represented and a vertex from N_p^f is in the spine-so-far position:

- If there is a member of N_p^s in the spine-so-far position (the second player must have put it there on his turn immediately previous), then the first player places a member of N_p^f in the spine-so-far position.
- Otherwise, the first player places a member of N_p^s anywhere except the spine-so-far position.

Inductively, and because $|N_p^f| > |N_p^s|$, we know that once all vertices of N_p are represented, one of the vertices in N_p^f will be in the spine-so-far position, and will by Observation 9 be in the spine in every game reachable from that position.

The case in which $|N_p^f| = |N_p^s|$ is only marginally trickier than the case in which N_p^f and N_p^s are different sizes. Here we provide a winning strategy for the player who places the second member of N_p. We call this player Bob, the other player Jim, and their win sets N_p^{Bob} and N_p^{Jim}.

Once Jim has placed a member of N_p, then for every subsequent turn until either all vertices in N_p are represented, or all vertices in N_p^{Jim} are represented and a vertex from N_p^{Bob} is in the spine-so-far position, Bob will play a member of N_p as follows:

- If there is a member of N_p^{Jim} in the spine-so-far position (Jim must have put it there on his turn immediately previous), then Bob places a member of N_p^{Bob} in the spine-so-far position.
- Otherwise, Bob places a member of N_p^{Jim} anywhere except the spine-so-far position.

If Bob uses this strategy, then when all of N_p are played, a member of N_p^{Bob} will be in the spine-so-far position, and will by Observation 9 be in the spine in every game reachable from that position.

Because placing the first member of N_p is a losing move if $|N_p^s| = |N_p^f|$, neither player will make that move unless there is no other choice.

Therefore, the position described with only one neighbour of v represented is a first player win if and only if the parity of the number of vertices of the spine represented thus far and all leaves attached to that spine (represented and unrepresented) is odd. □

Using the results from Lemma 46, we define an algorithm to determine whether the first or second player will win a connected permutation representation game using a tree given a tree and a mandatory head move.

Our algorithm determines, given a mandatory head move u for each other vertex v in the tree whether a game in which both u and v are in the spine of the caterpillar at the end of the game is a win for the first player, or for the second player.

We give an informal description of our algorithm: first, observe that for every leaf l_i that is not u, we can determine the winner of a game that includes both l_i and u in the spine of the represented caterpillar just by calculating the parity of the number of vertices on the path between u and l_i and all vertices adjacent to that path (leaves in the represented caterpillar). Labeling the leaves of the tree in this way as first or second player wins will be the first step of our algorithm.

Next, consider a vertex v such that all of its children (if the tree we are playing on were rooted at u) are correctly labeled as first or second player wins if they were on the spine. Then, we can determine the first or second player win value of that vertex by Lemma 46. Our algorithm iterates through the vertices of the tree rooted at the mandatory head in postorder, calculating this value at each vertex.

5.5 Conclusion

We have described a new class of games on graphs based on set representations. In general, solving positions of these games is PSPACE-hard, and in the case of the permutation and interval games is PSPACE-complete. However, we have given algorithms to solve, in polynomial-time, positions of the connected permutation and interval games when playing using a tree with a specified first-move restriction.

Bibliography

[1] Michael H Albert, Richard J Nowakowski, and David Wolfe. *Lessons in play: An introduction to the combinatorial theory of games*. A K Peters, Ltd, 2007.

[2] Hans L. Bodlaender. Kayles on special classes of graphs - an application of sprague-grundy theory. In *Proceedings of the 18th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '92, pages 90–102, London, UK, 1993. Springer-Verlag.

[3] Hans L. Bodlaender and Dieter Kratsch. Kayles and nimbers. *J. Algorithms*, 43(1):106–119, 2002.

[4] C.L Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3:35–39, 1901.

[5] B. Dushnik and E.W. Miller. Partially ordered sets. *American Journal of Mathematics*, 63:600–610, 1941.

[6] Rudolf Fleischer and Gerhard Trippen. Kayles on the way to the stars. In H. Jaap van den Herik, Yngvi Björnsson, and Nathan S. Netanyahu, editors, *Computers and Games*, volume 3846 of *Lecture Notes in Computer Science*, pages 232–245. Springer, 2004.

[7] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory (B)*, 16:47–56, 1974.

[8] Fanica Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Inf. Process. Lett.*, 73(5-6):181–188, 2000.

[9] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*. North-Holland, 2004.

[10] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *J. Comput. Syst. Sci.*, 16(2):185–225, 1978.

[11] Jeremy Spinrad. On comparability and permutation graphs. *SIAM Journal of Computing*, 14:658–670, 1985.

5.6 Further work on set representation games

Here we include further work on games that was not included in the previous paper. We describe several new types of games using graphs, and show that some set representation games can be considered as these new types of games. We give algorithms for resolving states of some connected set representation games using long stars.

5.6.1 Coupled Positions

We describe a characteristic of two games that we use later to show relationships between games.

Definition 12. *Let firstgame and secondgame be two games. We say that firstgame and secondgame have the* coupled positions *property if there exist polynomial-time transformations transFrom1To2 and transFrom2To1 such that for every position S of firstgame, for every child position S_2 of S, there is guaranteed to be a child position of transFrom1To2(S) that has the same outcome class as transFrom1To2(S_2), and for every position S_B of secondgame, for every child position S_{B2} of S_B, there is guaranteed to be a child position of transFrom2To1(S_B) that has the same outcome class as transFrom2To1(S_{B2}). Note that transFrom1To2(S_2) may not be a child position of transFrom1To2(S), and transFrom2To1(S_{B2}) may not be a child position of transFrom2To1(S_B).*

Observation 10. *Let firstgame, secondgame be two games with the coupled positions property with transformations transFrom1To2 and transFrom2To1. Then a position in firstgame has the same outcome class as its transformation into secondgame, and a position in secondgame has the same outcome class as its transformation into firstgame.*

Lemma 47. *Let firstgame and secondgame be two games with the coupled positions property. Then there is a polynomial-time algorithm for solving positions of firstgame if and only if there is a polynomial-time algorithm for solving positions of secondgame .*

Proof. Follows from Observation 10 □

5.7 Subgraph game

Let $G = (V, E)$ be a graph and $\mathcal{H}$ a graph class. We define a game using these: on a player's turn, he selects a vertex of G such that all selected vertices induce a graph in $\mathcal{H}$. A player cannot select a previously selected vertex. The last player to make a selection wins. We call this the maximal $\mathcal{H}$ subgraph game.

More precisely, a game position of the maximal $\mathcal{H}$ subgraph game using graph $G = (V, E)$ is a pair $(V', \mathcal{H})$ where V' is a subset of V such that $G[V'] \in \mathcal{H}$. A move in this game is a change from a game position $S_1 = (V_1', \mathcal{H})$ to a position $S_2 = (V_2', \mathcal{H})$ such that $V_1' \subset V_2'$, $V_2' \backslash V_1'$ contains a single vertex v, and $G[V_2'] \in \mathcal{H}$.

Theorem 14. *Computing the winner of an instance of the maximal $\mathcal{H}$ subgraph game is PSPACE-complete even when $\mathcal{H}$ is the class of cliques or of independent sets.*

Proof. First, because a recursive minimax search can solve an instance of the maximal connected $\mathcal{H}$ subgraph game in exponential time and polynomial space, the problem of solving an instance of the maximal connected $\mathcal{H}$ subgraph game is in PSPACE.

Let $\mathcal{H}$ be the class of cliques. Then resolving the winner of the maximal connected $\mathcal{H}$ subgraph game on a given graph is equivalent to resolving the winner of a Kayles game on the complement of that graph.

Let $\mathcal{H}$ be the class of independent sets. Then resolving the winner of the maximal $\mathcal{H}$ subgraph game on a given graph is equivalent to resolving the winner of a Kayles game on that graph. □

5.8 Growing Game

Let $G = (V, E)$ be a graph, and $\mathcal{H}$ a graph class. In the connected $\mathcal{H}$ proper representation game using G, which we will call the growing game, two players alternate turns.

A game position of the growing game using graph $G = (V, E)$ is a tuple
$(V' \subset V, J, \mathcal{J}, f, \cap, \mathcal{H})$ where $(J, \mathcal{J}, f, \cap)$ is a graph representation of $G[V']$ and $J \in \mathcal{H}$.

To define a legal move, we must describe two operations on a graph representation $(J, \mathcal{J}, f, \cap)$.

- Host subdivision
- Host vertex addition

Let $e = (u, w)$ be an edge in J. A *host subdivision* of e in the representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$, produces a new representation $(J', \mathcal{J}', f', \cap)$ such that :

- $J' = (V_J \cup \{v\}, (E_J \cup \{(u, v), (v, w)\}) \backslash \{e\})$ where v is a new vertex not in V_J
- For every subgraph $s_i \in \mathcal{J}$
  - If e is not in s_i then $s'_i = s_i$
  - If e is in $s_i = (V_{s_i}, E_{s_i})$ then $s'_i = (V_{s_i} \cup \{v_e\}, (E_{s_i} \cup \{(u, v), (v, w)\}) \backslash \{e\})$
- Then $\mathcal{J}' \leftarrow$ the set of each s'_i produced from $s_i \in \mathcal{J}$ as described above.
- For each vertex $v \in V$, the bijection $f'(v)$ yields s'_i if and only if $f(v) = s_i$

A *host vertex addition* with new neighbourhood $V_B \subset V$ in the representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$, produces a new representation $(J', \mathcal{J}', f', \cap)$ such that :

- $J' = (V_J \cup \{v\}, E_J \cup \{$ edges between v and each vertex in V_B $\})$ where v is a new vertex not in V_J
- $\mathcal{J}' \leftarrow \mathcal{J}$

- $f' \leftarrow f$

Definition 13. *Let $G' = (V', E')$ be the graph produced by adding vertex v with neighbourhood V_B to graph G. A V_B-*extension *of graph representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ of $G = (V, E)$ is a representation $(J', \mathcal{J}', f', \cap)$ of graph G' that can be produced by a series of host subdivisions and host vertex additions, and then the addition of a subgraph s of J' to $\mathcal{J}'$ such that s contains all vertices in J' that are not in J (it may contain others as well), and f' maps v to s but is otherwise the same as f.*

Then a move in the growing game from position $(G = (V, E), V', J = (V_J, E_J), \mathcal{J}, f, \cap, \mathcal{H})$ is:

- a selection of a vertex $v \in V$ that is not yet represented and therefore not yet in V'
- set $V'' \leftarrow V' \cup \{v\}$
- the production of a $(N_v \cap V')$-extension $(J', \mathcal{J}', f', \cap)$ of $(J = (V_J, E_J), \mathcal{J}, f, \cap)$, $(J', \mathcal{J}', f', \cap)$ such that $(J', \mathcal{J}', f', \cap)$ is a representation of $G[V'']$ and J' is in $\mathcal{H}$.

After this move, the position of the game is $(V'', J', \mathcal{J}', f', \cap, \mathcal{H})$.

Theorem 15. *Resolving the outcome class of a position of the growing game is PSPACE-hard.*

Proof. We have already seen that resolving the outcome class of a position of the proper interval game is PSPACE-complete (Theorem 13). We will give a reduction from a position of that game to a position of the growing game.

In fact, we will show that the proper interval game and the growing game played using paths have the coupled states property.

Let $S = (\mathcal{Q}, V', f)$ be a position of the proper interval game using graph $G = (V, E)$. We give a transformation into a state of the growing game played using paths. Let $J = (V_J, E_J)$ be a path on $2|V'|$ vertices with vertices numbered $v_1 ... v_{2|V'|}$. Let $\mathcal{H}$ be the class of all paths.

Let P be a list of the endpoints of intervals in $\mathcal{Q}$ sorted by position on the real line. There will be two endpoints for each interval. We will denote the left endpoint of interval z as z_l and the right endpoint as z_r. Let $P(z_l)$ be the position of z_l in P.

Then for each vertex $w_i \in V'$ we create a subpath p_i of J that contains exactly the vertices on the path between $v_{P(f(w_i)_l)}$ and $v_{P(f(w_i)_r)}$. We place all of these subpaths in $\mathcal{J}$. We add to a bijection g the value $[v_i, p_i]$. Then the endpoints of the subpaths on J corresponding to vertices in V' are in the same order as the endpoints of the intervals on the real line corresponding to those same vertices. Therefore two paths $g(v_i), g(v_j)$ intersect if and only if the intervals $f(v_i), f(v_j)$ intersect, so we have that $(J, \mathcal{J}, g, \cap)$ is a graph representation of G. Since J is a path, we also have that $(J, \mathcal{J}, g, \cap, \mathcal{H})$ is a position of the growing game using paths.

Let $(J, \mathcal{J}, g, \cap, \mathcal{H})$ be a position of the growing game using paths, where $\mathcal{H}$ is the class of paths. Number the vertices of J with $v_1 ... v_k$. For vertex $u_i \in V'$ let the subpath $g(u_i)$ span vertices $v_h ... v_j$ of J. Then for each vertex $u_i \in V'$, let interval b_i be the interval on the real line spanning h, k. Let $f(u_i) = b_i$, and let $\mathcal{L}$ contain all such b_i. Then $S = (\mathcal{L}, V', f)$ is a position of the proper interval game.

We claim that with these transformations the proper interval game and the growing game using paths have the coupled states property.

Let $S_{GROW} = (J, \mathcal{J}, g, \cap, \mathcal{H})$ be a position of the growing game where $\mathcal{H}$ is the class of paths, and $S_{INT} = (\mathcal{Q}, V', f)$ be its transformed version in the interval representation game. Consider a child position of S_{GROW} and its transformation S^2_{INT} into the interval representation game. We can produce a child position of S_{INT} that is endpoint-equivalent to S^2_{INT}, and therefore has the same outcome class: the endpoints of the intervals assigned to vertices in V' in the representation in S^2_{INT} are in the same order as they are in the representation in S_{INT}, by the transformation. Let v be the vertex represented in S^2_{INT} but not in S_{INT}, and s its assigned interval. Where are the endpoints of s in the representation in S^2_{INT}? Let $Left_{left}$ be the set of endpoints in S^2_{INT} to the left of the left endpoint of s, $Left_{right}$ the set of endpoints in S^2_{INT} to the right of the left endpoint of s, $Right_{left}$ be the set of endpoints in S^2_{INT} to the left of the right endpoint of s, $Right_{right}$ the set of endpoints in S^2_{INT} to the right of the right endpoint of s. Because the part of the representation of S^2_{INT} that represents vertices in V' is endpoint equivalent to the representation in S_{INT}, there is a point x in the representation of S_{INT} that is to the right of exactly $Left_{right}$ and the left of $Left_{right}$, and another point y that is to the right of exactly $Right_{right}$ and the left of $Right_{right}$. The child state of S_{INT} that consists of a representation that is the representation in S_{INT} with an added interval from x to y assigned to v is endpoint equivalent to S^2_{INT}.

Let $S_{INT} = (\mathcal{Q}, V', f)$ be a position of the interval representation game, and $S_{GROW} = (J, \mathcal{J}, g, \cap, \mathcal{H})$ its transformed equivalent in the growing game. Let S^2_{INT} be a child position of S_{INT} and S^2_{GROW} the transformation of S^2_{INT} into the growing game. By the transformations, S^2_{GROW} can be reached in a move from S_{GROW} by two host subdivisions and a path assignment.

Because the interval representation game and the growing game with paths have the coupled states property, a position of one has the same outcome class as its transformation in the other game, and resolving the outcome of a growing game position is as hard as resolving the outcome of an interval representation game state.

□

5.9 Dueling Bureaucrat Games

We describe a game with an appealing story that we came across while working on set representation games. We will use this game later in a result on solving a set representation game on long stars. This game differs from the other games we discuss - it is a *partizan* game, that is, it is not impartial.

The moves available to the two players are not identical, and their goals are not the same.

5.9.1 An Informal Description

There are two bureaucrats, Bob and Jim, in charge of appointing two cochairs of a committee. Bob is very lazy and wants to do as little work as possible. Jim is naive and wants to get as much done as possible. There are two pools of candidates: Democrats and Republicans. Bob and Jim both know that if one of the chairs is a Republican and the other is a Democrat, the committee will do nothing, as the chairs will just fight all the time. Therefore, Bob wants one chair from each party. Jim doesn't care which party the chairs belong to, so long as they both belong to the same one.

In the usual way of bureaucrats, Jim and Bob have come up with a bizarre way of appointing chairs. They will take turns. On a player's turn, he may either assign a candidate to one of the chairships (replacing and discarding the person previously assigned that chairship), or he may discard a candidate from either pool. He may only discard a candidate if both chairships currently have someone assigned to them, and he may not discard a candidate currently assigned a chairship without replacing him with someone else from the pool. Once a candidate is discarded, he is completely out of the running.

Observation 11. *One candidate is discarded every turn.*

So who will get his way? Is it better to choose first, or second?

We consider only the case in which the initial Republican and Democrat pools are of equal size. Here is a strategy that Bob can use to win if Jim has the first move:

- If Jim has just assigned a chairship to a candidate, then Bob should assign a candidate of the other party to the other chairship.
- If Jim has just discarded a candidate of one party, then Bob should discard a candidate of the other party.

How do we know that Bob will win with this strategy? If Jim has the first move, then Bob must have the last move. It's therefore sufficient to show that after every move of Bob's the two chairs are of different parties.

On Jim's first move he must place a candidate in a chairship. On Bob's first move, he'll place a candidate in the other chairship so that the two chairs are of different parties, so we know that after Bob's first move the two chairs are of different parties.

Certainly Bob is never going to assign a candidate to a chairship such that both candidates are from the same party. So if the two chairships are ever assigned candidates from the same party, Jim must have done it. As soon as Jim does this, Bob's strategy dictates that he'll immediately assign a candidate of the other party to a chairship.

How do we know a candidate from that other party is available for Bob to choose? Notice that whichever party Jim picks a candidate from, Bob always then picks one from the other. This

means that after every turn Bob takes the number of candidates from each party will be equal. This guarantees that after every choice Jim makes, there is always a candidate from the other party for Bob to choose.

Then we can conclude that after every move of Bob's the two chairs are of different parties. Since Bob has the last move, this means he will have his way in the end.

Here's a strategy that Jim can use. If the pools are of equal size and Bob has the first move, then Jim has the last move. If Jim is presented with two candidates of the same party currently in the chairship slots, then he need only discard the last candidate from the pool to win. If Jim is presented with two candidates of different parties currently in the chairship slots, then, no matter which party the only remaining candidate in the pool is a member of, Jim can assign him to one of the chairships such that both chairs are members of the same party.

Therefore if Bob has the first move, then Jim will have his way in the end.

Overall, we can conclude that:

Statement 5. *Regardless of who has the first turn, the second player will win if the pools are of equal size.*

5.9.2 Dueling Bureaucrats - Formal Version

Let A and B be disjoint sets. We describe a game using A and B. In this game there are two players: the same-player and the diff-player. On a player's turn he may either discard a member of A or B, or he may choose a member of A or B and place it in a third set C, removing it from A or B. The set C may never have more than two elements: if a player adds an element to C that would increase the number of elements in C to three, he must discard a member of C. At the beginning of the game C is empty. If C has fewer than two elements, then the player may not discard an element of A or B but must instead place an element of A or B in C.

The same-player wins the game if at the end of the game both elements of C come from the same one of A or B. The diff-player wins the game if at the end of the game one element of C is from A, and one is from B.

We consider the case in which $|A| = |B|$.

Same-player is the first player

We describe a diff-player winning strategy in this case.

- If same-player has just assigned a member of A to C, then diff-player should replace the other member of C with a member of B.
- If same-player has just assigned a member of B to C, then diff-player should replace the other member of C with a member of A.
- If same-player has just discarded a member of A, diff-player should discard a member of B.

- If same-player has just discarded a member of B, diff-player should discard a member of A.

Notice that every time same-player removes a member of A diff-player removes a member of B and every time same-player removes a member of B diff-player removes a member of A. Then, by the equal initial sizes of A and B, after every turn of diff-player $|A| = |B|$. Therefore on every diff-player turn, there is a member in the one of A or B that same-player did not remove an item from.

Notice also that after every diff-player turn, the two members of C will not be from the same one of A or B. Then, inductively, after diff-player's last turn the two members of C will not be from the same one of A or B. Because the total size of $A \cup B$ was even at the start of the game, diff-player will have the last move. Therefore diff-player wins.

Diff-player is the first player

Because $|A| = |B|$ same-player will have the last turn. Consider the contents of C before same-player's last turn. If both members of C are from the same one of A or B, then same-player wins by discarding the last remaining element of whichever one of A or B is not empty. If one member of C is from A and the other from B, then if A has one remaining element $a \in A$, same-player wins by replacing the member of C originally from B with a. If B has one remaining element $b \in B$, same-player wins by replacing the member of C originally from A with b.

We can conclude that:

Theorem 16. *In a dueling bureaucrats game for which the two sets A, B are of the same initial size the second player will win regardless of whether he is the diff-player or the same-player.*

5.9.3 Confirming Dueling Bureaucrats

Jim and Bob are cranky, and both think that there's not enough paperwork involved in this chair selection process. They've decided to add another possible move action. In addition to appointing and discarding candidates, they now add in a confirm option. If a candidate is currently assigned a chairship, Jim or Bob also now have the option of *confirming* that chair. They take a few minutes, fill out some forms in triplicate, and send them off. Once these forms are sent they cannot be gotten back, so once a chair is confirmed, he or she cannot be removed from that chairship. If there are no unassigned candidates left, the candidates assigned chairships are automatically confirmed. The game ends when there are two confirmed chairs.

What does this new rule add to the chairship game? Who will win? Once again, we'll only look at the case in which the number of Republican candidates is the same as the number of Democrat candidates.

If Jim plays second in the confirming dueling bureaucrats game and Bob never confirms a chairship, Jim has a winning strategy as in the game without confirming.

Jim can prevent Bob from ever confirming using the following strategy: at the end of Jim's turn, there must always be two candidates of the same party in the chairship positions. If Bob ever confirms a chair, then Jim can simply confirm the other and win. Therefore, Bob will never confirm a chair if the two chairships are assigned to two candidates of the same party.

By preventing Bob from confirming in this way, Jim can use his winning strategy on his last turn from the version of the game without confirming, and therefore Jim has a winning strategy as the second player in confirming dueling bureaucrats

The outcome if Bob plays second remains open.

5.10 Representation games on long stars

We have presented solutions for some connected set representation games with mandatory head for trees in general. Here, we present additional solutions on the long stars.

5.10.1 Connected proper interval representation game using long stars

First, we will consider playing the connected proper interval representation game using long stars.

A *long star* is a tree with only one vertex of degree greater than two (the *central vertex*), such that the central vertex has no leaf neighbours. The *rays* of a long star are the maximal paths that exclude the central vertex.

In the notation of the set representation game, the universe of sets we are choosing from is the family of all proper intervals on the real line, and the set relationship is intersection.

Recall that: If $S = (\mathcal{L}, V', f)$ is an end position of a connected proper interval representation game using a tree T, with V' the represented vertices, then $G[V']$ is a path.

The represented vertices at the end of every possible connected proper interval representation game using a tree T will induce a path, and will include two leaves of T. Then the represented vertices at the end of a game must consist of the central vertex, and all of the vertices on exactly two rays. We characterise which long stars are first player win, and which are second player win.

The *parity* of a ray is the parity of the number of vertices on that ray.

Statement 6. *If every ray of a long star is of even parity, or every ray is of odd parity, then the connected game using that long star is first player win.*

Proof. The parity of the number of vertices represented will be odd in every game end state. Therefore the first player will have won. □

Statement 7. *If there are at least two odd-parity rays on a long star, then the connected game using that long star is first player win.*

Proof. We describe a first player strategy in this case. On the first turn, first player can choose the leaf of one of the odd rays. The players will alternate vertices down the length of that odd ray.

Because of the odd parity of the ray, the second player will be forced to select the central vertex. Then the first player can choose the neighbour of the central vertex that is on another odd ray. Because only vertices from two rays can be represented, this guarantees that the game will end with exactly the two odd rays and the central vertex represented. Since that is an odd number of vertices, this is a first player win. □

Statement 8. *If there is exactly one odd parity ray on a long star, then the game on that long star is a second player win.*

Proof. Observe that by the parity of the number of vertices, any game that includes vertices from the single odd ray will be a second player win.

If the first player plays on the odd parity ray, then we have a second player win. If the first player plays on an even ray, the first player will be forced to play the central vertex, at which point the second player can select a vertex on the odd ray. If the first player plays the central vertex as his first move, the second player can immediately select a vertex on the odd ray.

□

Then we have overall that:

Lemma 48. *In the connected proper interval on a line representation game using a long star, the game is a second player win if and only if there is exactly one odd parity ray in the long star.*

5.10.2 Connected permutation game using long stars

We now consider winning strategies for the connected permutation game using long stars based on the parity of the rays of that long star. At the end of every game, the represented graph will consist of two rays, the central vertex, and all neighbours of the central vertex, as this is a maximal caterpillar. If all rays are of the same parity, then the winner is determined entirely by the parity of the number of rays: if there are an even number of rays then any game on that long star will be a first player win, if odd a second player win.

We then need consider the case in which there is at least one ray of each parity.

Consider the case with an odd number of rays. In this case, the first player wins if the two rays that are represented are of different parity. Then the first player has a winning strategy as follows: the first player chooses for his first move a leaf of a ray that is in the minority parity. Then, when the central vertex is played, there will be more unrepresented neighbours of the central vertex on rays of the original majority parity than of the original minority parity. The first player will win if one of the neighbours on a majority parity ray is on the spine at the end of the game, and the second player will win if one of the neighbours on a minority parity ray is on the spine at the end of the game. To use the notation of Lemma 46, the neighbours on majority parity rays are in N_p^f and the neighbours on minorty parity rays are in N_p^s. Because $|N_p^f| > |N_p^s|$, the first player will be able to force the choice of a ray of the majority parity, by Lemma 46.

Consider the case with an even number of rays. In this case, the first player wins if the two rays that are represented are of the same parity. If the number of rays with odd parity and the number of rays with even parity differ by at least one, then the first player has a winning strategy as follows: the first player chooses for his first move a leaf of a ray that is in the majority parity. Then, when the central vertex is played, the first player will be able to force the choice of a ray of that same parity, by Lemma 46.

We are left with the case in which there are an even number of rays, and the number of rays of even parity is the same as the number of rays of odd parity. Recall that the first player will win if the rays represented are of the same parity. We describe a winning strategy for the first player: the first player chooses the central vertex as his first move.

Notice that the first player wins if two rays of the same parity are on the spine, and the second player wins otherwise.

We will cast this situation as a game of the Confirming Dueling Bureaucrats game: consider the chairships as the spine-so-far positions, the rays of even parity as Republicans, the rays of odd parity as Democrats. The first player in the permutation game is Jim, and the second is Bob. When a player wants to assign a candidate as a chair, he places the first vertex on the corresponding ray in one of the two spine-so-far positions. When he wants to confirm a chair, he represents the second vertex along that ray. When he wants to discard a candidate, he represents the first vertex on that corresponding ray in any position except one of the spine-so-far position.

If the first player selects the central vertex as his first move, then the second player (Bob in the bureaucrat game) will have the first move in this corresponding Confirming Dueling Bureaucrats game. As we have shown, Jim then has a winning strategy. Then Jim will be able to force two candidates of the same party to be in the two chairships, and the first player of the permutation game will be able to force two rays of the same parity to be on the spine in the permutation game.

We conclude that:

Theorem 17. *In every connected permutation representation game using a long star*

- *if all rays are of the same parity and there are an even number of rays then there is a first player winning strategy.*
- *if all rays are of the same parity and there are an odd number of rays then there is a second player winning strategy.*
- *if not all rays are of the same parity then there is a first player winning strategy.*

5.11 Games with more than one definition

We have described several types of games: connected representation games, representation growing games, and subgraph games. Can some of our games be described in more than one of these

frameworks?

5.11.1 Connected proper interval representation game using a tree as a subgraph game

We claim that we can recast the connected proper interval representation game using a tree game as a subgraph game as well as a growing game. Consider the subgraph game played with graph class $\mathcal{H}$ being the class of paths using trees. We claim that such a game has the coupled positions property with the connected proper interval representation game using a tree.

We describe the required transformations. Let $S = (\mathcal{Q}, V', f)$ be a position of the connected proper interval representation game using a tree $T = (V, E)$. We give a transformation into a position of the subgraph game on paths using T: the tuple $(V', paths)$ is a valid position of the subgraph game, as $G[V']$ must be a path as it is a tree that is a proper interval graph.

We now give a transformation from a position of the subgraph game $(V', paths)$ using T to a legal position of the connected proper interval representation game using a tree T. Let the vertices in V' be ordered as $v_1 ... v_k$ along the path that they induce in T. Then for vertex v_i let interval l_i be the interval on the real line spanning $i - 1/2$ to $i + 1/2$. Let $\mathcal{Q}$ be composed of these intervals for every vertex in V'. Let bijection f contain $[v_i, l_i]$ for each such vertex and interval. Notice that $l_i \cap l_j \neq$ if and only if either $i = j + 1$ or $i = j - 1$. Then $S = (\mathcal{Q}, V', f)$ is a position of the connected proper interval representation game using a tree $T = (V, E)$.

Let $S^1_{INT} = (\mathcal{Q}^1, V'^1, f^1)$ be a position of the connected proper interval representation game using a tree $T = (V, E)$, $S^2_{INT} = (\mathcal{Q}^2, V'^2, f^2)$ a child position of S^1_{INT}, and $S^1_{SG} = (V'^1, paths)$ the transformation of S^1_{INT} into the subgraph game on paths. Let $S^2_{SG} = (V'^2, paths)$ be the transformation of S^2_{INT} into the subgraph game on paths. S^2_{SG} is reachable in a single move from S^1_{SG}. Therefore there is a child of S^1_{SG} that has the same outcome class as S^2_{SG}.

Let $S^1_{SG} = (V'^1, paths)$ be a position of the subgraph game on paths, $S^2_{SG} = (V'^2, paths)$ a child position of S^1_{SG}, $S^1_{INT} = (\mathcal{Q}^1, V'^1, f^1)$ the transformation of S^1_{SG} into the connected proper interval representation game using a tree T, and $S^2_{INT} = (\mathcal{Q}^2, V'^2, f^2)$ the transformation of S^2_{SG} into the connected proper interval representation game using a tree T. There is a child of S^1_{INT} that is endpoint equivalent to S^2_{INT} and therefore has the same outcome class.

Then the connected proper interval representation game using a tree T has the coupled positions property with the subgraph game on paths using T, and so by Lemma 47 a polynomial-time algorithm to solve one of these games also provides a polynomial-time algorithm for solving the other.

5.11.2 Connected permutation representation game using a tree as a subgraph game

If we are allowed to add a few extra rules about selecting new vertices, we can recast the connected permutation representation game using a tree with mandatory head as a subgraph game.

Let $T = (V, E)$ be a tree with mandatory head $u \in V$. We define the subgraph permutation analog game (SPAG) as follows: A game position consists of a subset of vertices $V' \subset V$ that induces a caterpillar with u at one end of the spine, and two vertices labeled $a, b \in V'$ such that b is a leaf adjacent only to a, and a is the end of the spine farthest from u. If $|V|' = 1$ then there is no b. We express a game position S as a tuple $S = (u, a, b, V')$.

A move consists of going from one position $S_1 = (u, a_1, b_1, V_1')$ to $S_2 = (u, a_2, b_2, V_2')$ where $V_1' \subset V_2'$, v is the only vertex in V_2' but not V_1' and either:

- $a_1 = a_2, b_1 = b_2$ and v is a neighbour of a vertex on the path between b_1 and u or
- $a_2 = b_1$ and $b_2 = v$ and v is a neighbour of b_1 or
- $a_2 = a_1$ and $b_2 = v$ and v is a neighbour of a_1

We say that two positions of the connected permutation representation game using a tree with mandatory head are *endpoint equivalent* if the ordering of the endpoints of the lines segments corresponding to vertices down the sides of the two infinite lines in the representation are the same. Observe that two positions that are endpoint equivalent and have the same mandatory head and are being played using the same tree have the same outcome class.

We now define transformations for changing a position of the SPAG using a tree into a position of the connected permutation representation game using a tree , and *vice versa*.

Let $S_{SPAG} = (u, a, b, V')$ be a position of the SPAG using a tree. We produce a position of the connected permutation representation game using a tree with mandatory head u. Let $P = [v_0 = u...v_k = b]$ be the path between u and b in V'. For $v_i \in P$, if i is even, we assign v_i a line connecting $i + 1$ on the left parallel line to $i - 1$ on the right. If i is odd, we assign v_i a line connecting $i - 1$ on the left parallel line to $i + 1$ on the right. Observe that the line for v_i so far intersects only the lines for v_{i-1} and v_{i+1}, if they exist.

The vertices in $V' \backslash P$ induce an independent set, and each is adjacent to exactly one vertex on P. We now add in the lines for these vertices. For a vertex w that is adjacent to $v_i \in P$ we assign it a line that connects $i + 1 - \epsilon$ on the left and $i - 1 + \epsilon$ on the right if i is odd, and a line that connects $i - \epsilon$ on the left and $i + \epsilon$ on the right if i is even. If another member of $V' \backslash P$ has already been assigned that line, we assign w another line just slightly below that one. Note that a line for a vertex v in $V' \backslash P$ intersects exactly the single line for the member of P that v is adjacent to in T.

Because two lines intersect if and only if their corresponding vertices are adjacent in $T[V']$, these lines constitute a representation of $T[V']$, and we have a position of the connected permutation representation game using a tree with mandatory head u. For an example of the placement of a line at position i in P and some of the lines around it, see Figure 5.5.

Let $S_{PERM} = (u, V', \mathcal{L}, f)$ be a position of the connected permutation representation game using a tree with mandatory head u. We produce a position of the SPAG using T with mandatory

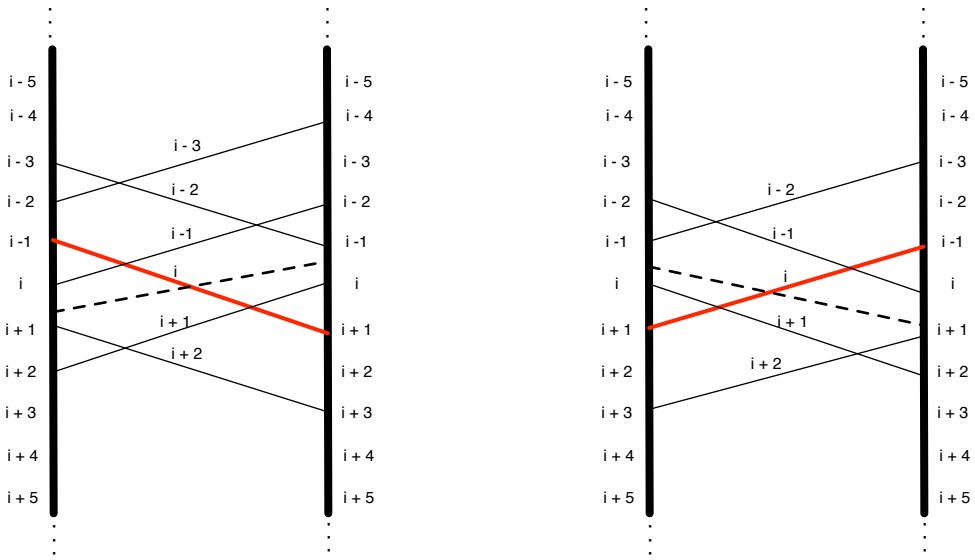

Figure 5.5: An illustration of the transformation from a position of the SPAG to the connected permutation representation game using a tree . If v_i is the ith vertex along the spine of the caterpillar in the SPAG instance, we show the placement of line i to correspond with vertex v_i, as well as lines for vertices v_{i-2} through v_{i+2}. We also show in dotted line the placement of a line to intersect only the line for v_i to indicate where a line for a leaf pendant from the spine adjacent only to vertex v_i could be placed. On the left is the line placement if i is odd, on the right if i is even.

head u. Let a and b be the two vertices with the two lowest lines in $\mathcal{L}$, with a being the one closer to u, and therefore on the path from b to u.

Then $S_{SPAG} = (u, a, b, V')$ is a position of the SPAG using T with mandatory head u.

Let $S^2_{SPAG} = (u, a_2, b_2, V'_2)$, $S^1_{SPAG} = (u, a_1, b_1, V'_1)$ be legal positions of the SPAG on tree $T = (V, E)$, and $S^2_{PERM} = (u, \mathcal{L}_2, V'_2, f_2)$, $S^1_{PERM} = (u, \mathcal{L}_1, V'_1, f_1)$ their transformed versions in the connected permutation representation game using a tree .

We claim that if position $S^2_{SPAG} = (u, a_2, b_2, V'_2)$ is reachable by a legal move from position $S^1_{SPAG} = (u, a_1, b_1, V'_1)$ in the SPAG, then a game position endpoint equivalent to S^2_{PERM} is reachable by a legal move from position S^1_{PERM} in the connected permutation representation game using a tree T.

Assume that $S^2_{SPAG} = (u, a_2, b_2, V'_2)$ is reachable by a legal move from position $S^1_{SPAG} = (u, a_1, b_1, V'_1)$ in the SPAG.

We consider three cases. Either:

1. $a = a_1, b = b_1$ and v is a neighbour of a vertex on the path between b and u or
2. $a_1 = b$ and $b_1 = v$ and v is a neighbour of b or
3. $a_1 = a$ and $b_1 = v$ and v is a neighbour of a

Observe that in any case the part of the representation in S^2_{PERM} that represents vertices that are

not v is endpoint equivalent to the representation in S^1_{PERM}, by the nature of the transformation.

In the first case, $a = a_1, b = b_1$ and v is a neighbour of a vertex on the path between b and u. Then the placement of the line for v can be achieved in a single turn.

In the second case $a_1 = b$ and $b_1 = v$ and v is a neighbour of b. Then we can reach a position endpoint equivalent to S^2_{PERM} from S^1_{PERM} by making $\mathcal{L}_2$ be $\mathcal{L}_1$ plus a line that is below all lines it does not intersect and that intersects only $f(b)$, making f' be f plus an assignment of v to the new line, and making V'_2 be V' plus v.

In the third case $a_1 = a$ and $b_1 = v$ and v is a neighbour of a. Then we can reach a position endpoint equivalent to S^2_{PERM} from S^1_{PERM} by making $\mathcal{L}_2$ be $\mathcal{L}_1$ plus a line that is below all lines it does not intersect and that intersects only $f(a)$, making f' be f plus an assignment of v to the new line, and making V'_2 be V' plus v.

Now let $S^1_{PERM} = (u, \mathcal{L}_1, V'_1, f_1)$. $S^2_{PERM} = (u, \mathcal{L}_2, V'_2, f_2)$ be legal positions in the connected permutation representation game using a tree T, and let $S^2_{SPAG} = (u, a_2, b_2, V'_2)$, $S^1_{SPAG} = (u, a_1, b_1, V'_1)$ be their transformed versions in the SPAG.

We claim that if position S^2_{PERM} is reachable by a legal move from S^1_{PERM} in the connected permutation representation game using a tree, then S^2_{SPAG} is reachable by a legal move from S^1_{SPAG} in the SPAG.

Let v be the vertex in $V'_2 \backslash V'_1$. There are two possibilities for $f_2(v)$. It is either a lowest line in the representation, or it is not. If it is not, then the two bottom lines in $\mathcal{L}_1$ and $\mathcal{L}_2$ correspond to the same two vertices, and so $a_1 = a_2$ and $b_1 = b_2$, and $S^2_{SPAG} = (u, a_1, b_1, V'_1 \cup \{v\})$, a position reachable in a legal move from S^1_{SPAG}.

If $f_2(v)$ is a lowest line in the representation, then the other lowest line is either the line corresponding to a_1 or b_1. If the other lowest line corresponds to a_1, then $a_1 = a_2$ and $S^2_{SPAG} = (u, a_1, b_2, V'_1 \cup \{v\})$ where b_2 is a neighbour of a_1, a position reachable in a legal move from S^1_{SPAG}. If the other lowest line corresponds to b_1, then $a_2 = b_1$ and $S^2_{SPAG} = (u, b_1, b_2, V'_1 \cup \{v\})$ where b_2 is a neighbour of b_1, a position reachable in a legal move from S^1_{SPAG}.

Then overall we can say that:

Lemma 49.

- *Let $S^2_{PERM} = (u, \mathcal{L}_2, V'_2, f_2)$ be a position in the connected permutation representation game using a tree reachable by a legal move from position $S^1_{PERM} = (u, \mathcal{L}_1, V'_1, f_1)$. Then if $S^2_{SPAG} = (u, a_2, b_2, V'_2)$, $S^1_{SPAG} = (u, a_1, b_1, V'_1)$ are the transformed versions of those positions in the SPAG, S^2_{SPAG} is reachable by a legal move from S^1_{SPAG} in the SPAG.*

- *Let $S^2_{SPAG} = (u, a_2, b_2, V'_2)$ be a position in the SPAG reachable by a legal move from position $S^1_{SPAG} = (u, a_1, b_1, V'_1)$. Then if $S^2_{PERM} = (u, \mathcal{L}_2, V'_2, f_2)$, $S^1_{PERM} = (u, \mathcal{L}_1, V'_1, f_1)$ are the transformed versions of those positions in the connected permutation representation game using a tree , then a position endpoint equivalent to S^2_{PERM} is reachable by a legal move from S^1_{PERM} in the connected permutation representation game using a tree .*

It follows that the SPAG and the connected permutation representation game using a tree with the same mandatory heads played using the same tree have the coupled positions property, and therefore a polynomial time algorithm to solve one would suffice, with the transformations, to solve the other.

5.12 Single Extension

Sometimes, the growing game ends with all vertices represented, regardless of play. We describe a property of the graph provided and the type of representation that tells us exactly when this will happen.

Recall that if $G' = (V', E')$ is the graph produced by adding vertex v with neighbourhood V_B to graph G. A *V_B-extension* of graph representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ of $G = (V, E)$ is a representation $(J', \mathcal{J}', f', \cap)$ of graph G' that can be produced by a series of host subdivisions and host vertex additions, and then the addition of a subgraph s of J' to $\mathcal{J}'$ such that s contains all vertices in J' but not in J, and f' maps v to s but is otherwise the same as f.

Then we define the single extension property as:

Definition 14. *Let $\mathcal{H}$ and $\mathcal{K}$ be graph classes. We say that the ordered pair $(\mathcal{H}, \mathcal{K})$ has the* single extension *property if for every graph $G = (V, E) \in \mathcal{H}$ with proper subgraph intersection representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ such that J is in $\mathcal{K}$ for every graph $G' \in \mathcal{H}$ that is G plus one extra vertex v with arbitrary neighbourhood V_n, there is a V_n-extension of $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ such that the host graph of the V_n-extension is in $\mathcal{K}$.*

Lemma 50. *The ordered pair $(trees, trees)$ has the single extension property.*

Proof. Let $T = (V_T, E_T)$ be a tree with proper subgraph intersection representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ such that J is a tree. Then because there are no cliques of size three in T, and because the representation is proper, in each subgraph in $\mathcal{J}$ there is at least an edge e that is only in that subgraph and not in any other.

Let T' be a tree that can be created by adding a single vertex, v', to T. Let u be the neighbour of v' in T'.

Let $(J' = (V'_J, E'_J), \mathcal{J}', f', \cap)$ be a representation created by first performing a host subdivision on edge e, and then a host vertex addition of a vertex adjacent to only the subdividing vertex, and then adding to $\mathcal{J}$ a subtree consisting of only those two new vertices with f' mapping that new subtree to v.

The representation $(J' = (V'_J, E'_J), \mathcal{J}', f', \cap)$ is a $\{u\}$-extension of $(J = (V_J, E_J), \mathcal{J}, f, \cap)$, and is a representation of T'. □

Lemma 51. *The ordered pair $(paths, paths)$ has the single extension property.*

Proof. Let $P = (V_P, E_P)$ be a path with proper subgraph intersection representation $(J = (V_J, E_J), \mathcal{J}, f, \cap)$ such that J is a path. Because the intersection representation is proper, the two vertices on the ends of P must be assigned subgraphs that each contain a vertex contained only in that subgraph and not in any other member of $\mathcal{J}$.

Let P' be a path that can be created by adding a single vertex, v, to P. Vertex v' must be adjacent to one of the end vertices in P. We call this vertex u.

Let $(J' = (V'_J, E'_J), \mathcal{J}', f', \cap)$ be a representation created by performing a host vertex addition of a vertex adjacent to only the end vertex of J contained only in $f(u)$, and then adding to $\mathcal{J}$ a subpath consisting of only the new vertex and its neighbour with f' mapping that new subpath to v.

The representation $(J' = (V'_J, E'_J), \mathcal{J}', f', \cap)$ is a $\{u\}$-extension of $(J = (V_J, E_J), \mathcal{J}, f, \cap)$, and is a representation of P'. □

Lemma 52. *Let $G = (V, E)$ be a graph, $\mathcal{H}$ a graph class, and $\mathcal{A}$ a graph class consisting of exactly G and its induced subgraphs. The ordered pair $(\mathcal{A}, \mathcal{H})$ has the single extension property if and only if every connected $\mathcal{H}$ representation game using G ends with all vertices of G represented, regardless of play.*

Proof. Assume that $(\mathcal{A}, \mathcal{H})$ has the single extension property. Now assume that there is an end state of the game in which not all vertices are represented. However, by the fact that $(\mathcal{A}, \mathcal{H})$ has the single extension property, that end state cannot be an end state, a contradiction.

Assume that every connected $\mathcal{H}$ representation game using G ends with all vertices of G represented, regardless of play. Then the class of graphs composed of exactly G and all of its induced subgraphs must have the single extension property with H. □

Chapter 6

Conclusion

6.1 Future work

The work presented in this thesis suggests many areas for future research. While we show the hardness of recognising several subclasses of subtree overlap graphs, parameterised by leafage of the underlying tree and maximum degree in the underlying tree, the complexity of recognising subtree overlap graphs in general is open.

Our work on relating S-covered subtree overlap graphs to the S-filament graphs showed that overlap graphs of caterpillars are exactly interval filament graphs. It was previously known to be NP-complete to recognise interval filament graphs, and so now we know that it is NP-complete to recognise caterpillar overlap graphs. Might this filament approach to overlap graphs give insight into some other subclasses of subtree overlap graphs? What other covering subtrees are interesting? We suggest perhaps the caterpillar-covered subtree overlap graphs for future study, as they are a relatively simple class of trees that contain paths.

Our result holds only for filaments on subtrees of a tree. Gavril defines many other types of filaments. Is there an analagous covering theorem for other filaments, for example the circular arc filaments, that would characterise these classes in terms of overlap representations? Gavril has already shown that they have $\mathcal{G}$-mixed partitions, an important part of our work.

The multi-chain ordering we used to list colour permutation and interval graphs also suggests future research. Our algorithm will work for any graph for which every induced subgraph has a multi-chain ordering. We currently don't know much about these graphs - do they have some other characterisation? Can they be recognised efficiently, and can multi-chain orderings be generated in linear time? At the moment the only algorithm for generating multi-chain orderings in general graphs requires constructing a breadth-first search tree from each vertex.

We also think that the multi-chain ordering may be useful in devising algorithms for other problems. A similar ordering was used by Heggernes et al. [1] to compute bandwidth on bipartite permutation graphs. Can the multi-chain ordering be used to compute bandwidth on a larger class of graphs? As outlined by Heggernes et al. [1], bandwidth is a notoriously difficult problem. Very

few graph classes are known to have polynomial time algorithms for bandwidth. Heggernes et al. [1] identify determining the complexity of bandwidth on permutation graphs as an important open question.

We've only just begun to describe our newly defined set representation, subgraph, and growing games. We solved some cases of some games with mandatory heads, and we provided solutions when the games were played using trees. The complexity of solving these games without a mandatory head, using graphs other than trees, and using other types of set representations is open.

6.2 Conclusion

We have explored research areas related to set representations of graphs. In particular, much of our work has dealt with graph classes with set representation characterisations. As outlined in the introduction, this has been a popular area of research, with many classes defined and algorithms described.

First, we focused on recognising and characterising graph classes with set representation characterisations. We considered various subclasses of subtree overlap graphs. We showed that recognising the overlap graphs of subtrees in a tree with a fixed number of leaves in the underlying host tree is NP-complete. We showed that recognising the overlap graphs of subtrees of subdivisions of a particular tree is NP-complete, provided that specified tree is not a path. We showed that recognising the overlap and intersection graphs of paths in a tree with fixed maximum degree is NP-complete. We found all of these results somewhat surprising, but particularly the hardness result on the intersection class. There exist polynomial time algorithms for recognising the intersection graphs of paths in trees - in this instance adding the mixed maximum degree constraint made the recognition problem harder rather than easier. The complexity of recognising subtree overlap graphs remains open. As with the intersection graphs of paths in a tree with fixed maximum degree, fixing parameters of the representation may have made our subtree overlap recognition problems harder - perhaps subtree overlap graphs can be recognised in polynomial time.

We then moved on to characterisations of subclasses of subtree overlap graphs in terms of their filament representations. Showing that graphs that have one type of representation are exactly graphs that have some other type of representation gives us more ways of looking at these classes of graphs. In particular, this work showed equivalence between some intersection and overlap classes, including the interval filaments and caterpillar overlap graphs. Previously, the complexity of recognising caterpillar overlap graphs was open, but because recognition is hard for interval filament graphs, we now know it to be hard.

Next we described a polynomial-time algorithm that uses set representation characteristics for an otherwise-hard problem. We gave a polynomial-time algorithm for list colouring graphs with a multi-chain vertex ordering if all of their induced subgraphs have multi-chain orderings and the total number of colours is fixed. We showed that two classes defined by their set representations,

permutation and interval graphs, have these properties. This vertex ordering may apply to larger classes of graphs, and may lead to more polynomial-time algorithms for hard problems.

Finally, we defined a family of games using set representations of graphs. In the set representation game, players select sets from a given pool to build a set representation of a graph. The last player to make a legal move wins. We show this family of games in general, and some subclasses of this game in particular, are generalisations of Kayles, and therefore it is NP-hard to determine, with perfect play, which player will win a position.

These games are impartial, so we can use the well-developed Sprague-Grundy theory to analyse them and their sums. We showed that our set representation games are equivalent to another type of game we define: the subgraph games. In these games, players must choose vertices from a graph such that the subgraph induced by the chosen vertices is always in some specified graph class. The subgraph games are a generalisation of Kayles. Kayles is a subgraph game played with independent sets as the specified type of graphs that the chosen vertices must induce.

We've done work on three areas at the core of research on graph classes defined by their set representations: recognition complexity, characterisations, and polynomial-time algorithms using the set representation characterisation of the class. We also moved ideas on set representations of graphs into combinatorial game theory, defining a number of games, and giving complexity results and polynomial-time algorithms to solve some games. Set representations of graphs in these, and other, areas will continue to be active areas of research.

Bibliography

[1] Pinar Heggernes, Dieter Kratsch, and Daniel Meister. Bandwidth of bipartite permutation graphs in polynomial time. In *Proceedings of the 8th Latin American conference on Theoretical informatics*, LATIN'08, pages 216–227, Berlin, Heidelberg, 2008. Springer-Verlag.