On Algorithms for (P_5,Gem)-Free Graphs

Hans L. Bodlaender [1]

Institute for Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. Email: `hansb@cs.uu.nl`

Andreas Brandstädt [2]

Fachbereich Informatik, Universität Rostock, A.-Einstein-Str. 21, 18051 Rostock, Germany. Email: `ab@informatik.uni-rostock.de`

Dieter Kratsch   Michaël Rao

Université de Metz, Laboratoire d'Informatique Théorique et Appliquée, 57045 Metz Cedex 01, France. Email: `(kratsch|rao)@sciences.univ-metz.fr`

Jeremy Spinrad [3]

Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville TN 37235, U.S.A. Email: `spin@vuse.vanderbilt.edu`

Abstract

A graph is (P_5,gem)-free, when it does not contain P_5 (an induced path with five vertices) or a gem (a graph formed by making an universal vertex adjacent to each of the four vertices of the induced path P_4) as an induced subgraph.

We present $O(n^2)$ time recognition algorithms for chordal gem-free graphs and for (P_5,gem)-free graphs. Using a characterization of (P_5,gem)-free graphs by their prime graphs with respect to modular decomposition and their modular decomposition trees [6], we give linear time algorithms for the following NP-complete problems on (P_5,gem)-free graphs: Minimum Coloring, Maximum Weight Stable Set, Maximum Weight Clique, and Minimum Clique Cover.

Key words: (P_5,gem)-free graphs; recognition algorithms; modular decomposition; independent set; clique; coloring; clique cover.

[1] The research of Hans Bodlaender was partly supported by EC contract IST-1999-14186: Project ALCOM-FT (Algorithms and Complexity - Future Technologies).
[2] The research of Andreas Brandstädt was supported by LORIA, Nancy.
[3] The research of Jeremy Spinrad was supported by the University of Metz.

1 Introduction

Graph decompositions play an important role in graph theory. The central role of decompositions in the recent proof of one of the major open conjectures in Graph Theory, the so-called Strong Perfect Graph Conjecture of C. Berge, is an exciting example [9]. Furthermore various decompositions of graphs such as decomposition by clique cutsets, tree-decomposition and clique-width are often used to design efficient graph algorithms. There are even beautiful general results stating that a variety of NP-complete graph problems can be solved in linear time for graphs of bounded treewidth and bounded clique-width, respectively [1,13].

Despite the fact that modular decomposition is a well-known decomposition in graph theory having algorithmic uses that seem to be simple and obvious, there is relatively little research concerning non-trivial uses of modular decomposition such as designing polynomial time algorithms for NP-complete problems on special graph classes.

An important exception are the many linear and polynomial time algorithms for cographs [10,11]. Cographs, or equivalently, P_4-free graphs are known to have a cotree representation. This representation allows for various problems that are NP-complete for arbitrary graphs a linear time algorithm when restricted to cographs. Among these, there are the problems Maximum (Weight) Stable Set, Maximum (Weight) Clique, Minimum Coloring and Minimum Clique Cover [10,11].

The original motivation to study the structure of (P_5,gem)-free graphs in [6] had been to construct a faster, possibly linear time algorithm for the Maximum Stable Set problem on (P_5,gem)-free graphs. In [6] the authors established a characterization of the (P_5,gem)-free graphs by their prime induced subgraphs called the Structure Theorem for (P_5,gem)-free graphs. We show in this paper that the Structure Theorem is a powerful tool to design efficient algorithms for NP-complete problems on (P_5,gem)-free graphs. All our algorithms use the modular decomposition tree of the input graph and the structure of the prime (P_5,gem)-free graphs. We are convinced that efficient algorithms for other NP-complete graph problems (e.g. domination problems) on (P_5,gem)-free graphs can also be obtained by this approach.

It is remarkable that there are only few papers establishing efficient algorithms for NP-complete graph problems using modular decomposition and that most of them consider a single problem, namely Maximum (Weight) Stable Set. For work dealing with other problems we refer to [4,5,19]. Concerning the limits of modular decomposition it is known, for example, that Achromatic Number, List Coloring, and $\lambda_{2,1}$-Coloring with pre-assigned colors remain NP-complete on cographs [2,3,22]. This implies that these three problems are NP-complete on (P_5,gem)-free graphs. [4]

There is also a strong relation between modular decomposition and the clique-width of graphs. For example, if all prime graphs of a graph class have bounded size then this class has bounded clique-width. Problems definable in a certain logic, so-called LinEMSOL($\tau_{1,L}$)-definable problems, such as Maximum (Weight) Stable Set, Maximum (Weight) Clique and Minimum (Weight) Dominating Set, can be solved in linear time on any graph class of bounded clique-width, assuming a k-expression describing the graph is part of the input [13]. Many other NP-complete problems which are not LinEMSOL($\tau_{1,L}$)-definable can be solved in polynomial time on graph classes of bounded clique-width [16,23]; see also [32].

Brandstädt et al. have shown that the clique-width of (P_5,gem)-free graphs is at most five [7]. However their approach does not provide a linear time algorithm to compute a suitable k-expression, thus it does not imply linear time algorithms for LinEMSOL($\tau_{1,L}$)-definable problems on (P_5,gem)-free graphs.

Finding a linear time algorithm that computes a 5-expression for a given (P_5,gem)-free graph remains an interesting open problem.

Our paper is organized as follows: Section 2 gives several preliminaries: basic notions (Section 2.1), modular decomposition (Section 2.2), cographs (Section 2.3), and a review of the general approach for obtaining efficient algorithms for NP-complete graph problems using modular decomposition (Section 2.4). Section 3 describes the structure of (P_5,gem)-free graphs and provides the Structure theorem. Section 4 presents an $O(n^2)$ algorithm to recognize chordal cogem-free graphs and an $O(n^2)$ algorithm to recognize (P_5,gem)-free graphs. In Section 5 we present a linear time algorithm to compute a maximum weight stable set in (P_5,gem)-free graphs. In Section 6 a linear time algorithm to compute a maximum weight clique on (P_5,gem)-free graphs is given. In Section 7 we present a linear time algorithm to compute a minimum coloring of (P_5,gem)-free graphs. In Section 8 we present a linear time algorithm to compute a minimum clique cover of (P_5,gem)-free graphs. Summarizing, we establish efficient and practical algorithms to solve four basic NP-complete graph problems on (P_5,gem)-free graphs.

[4] A proof, similarly to the one in [3] shows that $\lambda_{2,1}$-Coloring is NP-complete for graphs with at most one prime induced subgraph, the P_4, and hence for (P_5,gem)-free graphs.

2 Preliminaries

2.1 Basic notions

Let $G = (V, E)$ be a finite undirected graph, and let $|V| = n$ and $|E| = m$. Let $N(v) := \{u : u \in V, u \neq v, \{u, v\} \in E\}$ denote the *open neighborhood* of v and $N[v] := N(v) \cup \{v\}$ the *closed neighborhood* of v. The *complement graph* $\overline{G} = (V, \overline{E})$ of G is defined by $\overline{E} = \{\{u, v\} : u, v \in V, u \neq v \text{ and } \{u, v\} \notin E\}$. For $U \subseteq V$, let $G[U]$ denote the subgraph of G induced by U. A graph G is *connected* if for each pair u, v of vertices of G there is a path joining u and v. A graph is *co-connected* if its complement $\overline{G}$ is connected. A *(connected) component* of a graph G is a maximal connected subgraph of G. If for $U \subset V$, a vertex not in U is adjacent to exactly k vertices in U then it is called *k-vertex* for U.

For $k \geq 1$, let P_k denote a path with k vertices and $k-1$ edges, and for $k \geq 3$, let C_k denote a cycle with k vertices and k edges. A *hole* is a C_k for $k \geq 5$. Let $\mathcal{F}$ denote a set of graphs. A graph G is *$\mathcal{F}$-free* if none of its induced subgraphs is in $\mathcal{F}$.

A vertex set $U \subseteq V$ is a *clique* in G if the vertices in U are pairwise adjacent. A vertex set $U \subseteq V$ is a *stable* (*independent*) set in G if U is a clique in $\overline{G}$. $\omega(G)$ denotes the maximum cardinality of a clique in G and $\alpha(G) := \omega(\overline{G})$ denotes the maximum cardinality of a stable set in G. For a graph G with vertex weight function $w : V \rightarrow \mathbb{N}$, the weight of a vertex set $U \subseteq V$ is defined to be $w(U) := \sum_{u \in U} w(u)$. $\alpha_w(G)$ denotes the maximum weight of a stable set of G and $\omega_w(G)$ denotes the maximum weight of a clique of G. We assume that arithmetic operations on vertex weights can be performed in constant time. A stable set and a clique, respectively, of maximum cardinality (weight) is said to be a *maximum (weight) stable set* and a *maximum (weight) clique*, respectively.

A function $f : V \rightarrow \mathbb{N}$ is a *(proper) coloring* of the graph $G = (V, E)$, if $\{u, v\} \in E$ implies $f(u) \neq f(v)$. The *chromatic number* of G, denoted $\chi(G)$, is the smallest k such that the graph G has a k-coloring $f : V \rightarrow \{1, 2, \ldots, k\}$. A *clique cover* of a graph $G = (V, E)$ is a partition of its vertex set V into cliques $K_1, K_2, \ldots, K_t$. $\kappa(G)$ denotes the minimum number of cliques in any clique cover of G. It is well-known that $\chi(G) = \kappa(\overline{G})$ for all graphs.

Finally we shall consider the following weighted versions of coloring and clique cover. Let $G = (V, E)$ be a graph with vertex weight function $w : V \rightarrow \mathbb{N}$. A *weighted k-coloring* of (G, w) assigns to each vertex v of G $w(v)$ different colors, i.e., integers of $\{1, 2, \dots, k\}$, such that $\{x, y\} \in E$ implies that no color assigned to x is equal to a color assigned to y. $\chi_w(G)$ denotes the smallest k such that the graph G with weight function w has a weighted k-coloring. Note that each weighted k-coloring of (G, w) corresponds to a multiset $S_1, S_2, \dots, S_k$ of stable sets of G where S_i, $i \in \{1, 2, \dots, k\}$, is the set of all vertices of G to which color i is assigned.

A *weighted clique cover* of (G, w) is a multiset of cliques $K_1, K_2, \dots, K_t$ of G such that each vertex v of G is contained in at least $w(v)$ of these cliques. The minimum number of cliques in a weighted clique cover of (G, w) is denoted by $\kappa_w(G)$. Note that $\chi_w(G) = \kappa_w(\overline{G})$.

2.2 Modular decomposition

Modular decomposition is a fundamental decomposition technique that can be applied to graphs, partially ordered sets, hypergraphs and other structures. It has been described and used under different names and it has been rediscovered various times. Gallai introduced and studied modular decomposition in his seminal 1967 paper [18] where it is used to decompose comparability graphs. A translation into English is now available: [25].

A vertex set $M \subseteq V$ is a *module* in G if for all vertices $x \in V \setminus M$, x is either adjacent to all vertices in M, or non-adjacent to all vertices in M. The *trivial modules* of G are $\emptyset, V$ and the singletons. A *homogeneous set* in G is a nontrivial module in G. A graph containing no homogeneous set is called *prime*. Note that the smallest nontrivial prime graph is the P_4. A homogeneous set M is *maximal* if no other homogeneous set properly contains M.

Modular decomposition of graphs is based on the following decomposition theorem.

Theorem 1 ([18]). *Let $G = (V, E)$ be a graph with at least two vertices. Then exactly one of the following conditions holds:*

(i) *G is not connected, and it can be decomposed into its connected components;*
(ii) *$\overline{G}$ is not connected, and G can be decomposed into the connected components of $\overline{G}$;*
(iii) *G is connected and co-connected. There is some $U \subseteq V$ and a unique partition P of V such that*
  - ***(a)*** *$|U| > 3$,*
  - ***(b)*** *$G[U]$ is a maximal prime induced subgraph of G, and*
  - ***(c)*** *for every class S of the partition P, S is a module of G and $|S \cap U| = 1$.*

There does not seem to be a convention how to describe modular decomposition. We take the freedom to describe it in the way we find most convenient for presenting our algorithmic results in Sections 5 to 8.

Following Theorem 1 there are three decomposition operations.

0-Operation: If G is disconnected then decompose it into its connected components $G_1, G_2, \ldots, G_r$.
1-Operation: If $\overline{G}$ is disconnected then decompose G into $G_1, G_2, \ldots G_s$, where $\overline{G}_1, \overline{G}_2, \ldots \overline{G}_s$ are the connected components of $\overline{G}$.
2-Operation: If $G = (V, E)$ is connected and co-connected then its maximal homogeneous sets are pairwise disjoint and they form the partition P of V. The graph $G^* = G[U]$ is called the *characteristic graph* of G and it is obtained from G by contracting every maximal homogeneous set of G to a single vertex.

Note that the characteristic graph of a connected and co-connected graph G is prime.

The decomposition theorem and the above mentioned operations lead to the uniquely determined *modular decomposition tree* T of G. The leaves of the modular decomposition tree are the vertices of G. The interior nodes of T are labeled 0, 1 or 2 according to the operation corresponding to the node. Thus we call them 0-node (parallel node), 1-node (series node) and 2-node (prime node). Any interior node x of T corresponds to the subgraph of G induced by the set of all leaves in the subtree of T rooted at x, denoted by $G(x)$.

0-node. The children of a 0-node x correspond to the components obtained by a 0-operation applied to the disconnected graph $G(x)$.
1-node. The children of a 1-node x correspond to the components obtained by a 1-operation applied to the non co-connected graph $G(x)$.
2-node. The children of a 2-node x correspond to the subgraphs induced by the maximal homogeneous sets of the connected and co-connected graph $G(x)$ and to single vertices which are not contained in any homogeneous set. Additionally, the characteristic graph of $G(x)$ is assigned to the 2-node x.

The modular decomposition tree is of basic importance for many algorithmic applications, and in [26,14,15], linear time algorithms are given for determining the modular decomposition tree of an input graph. See Figure 1 for an example of a modular decomposition tree. [5]

[5] In case of 2-nodes, we use letters to indicate the one-to-one correspondence between the vertices of the characteric graph of $G(x)$ and the children of the 2-node x which represent either maximal homogeneous sets or single vertices of $G(x)$.

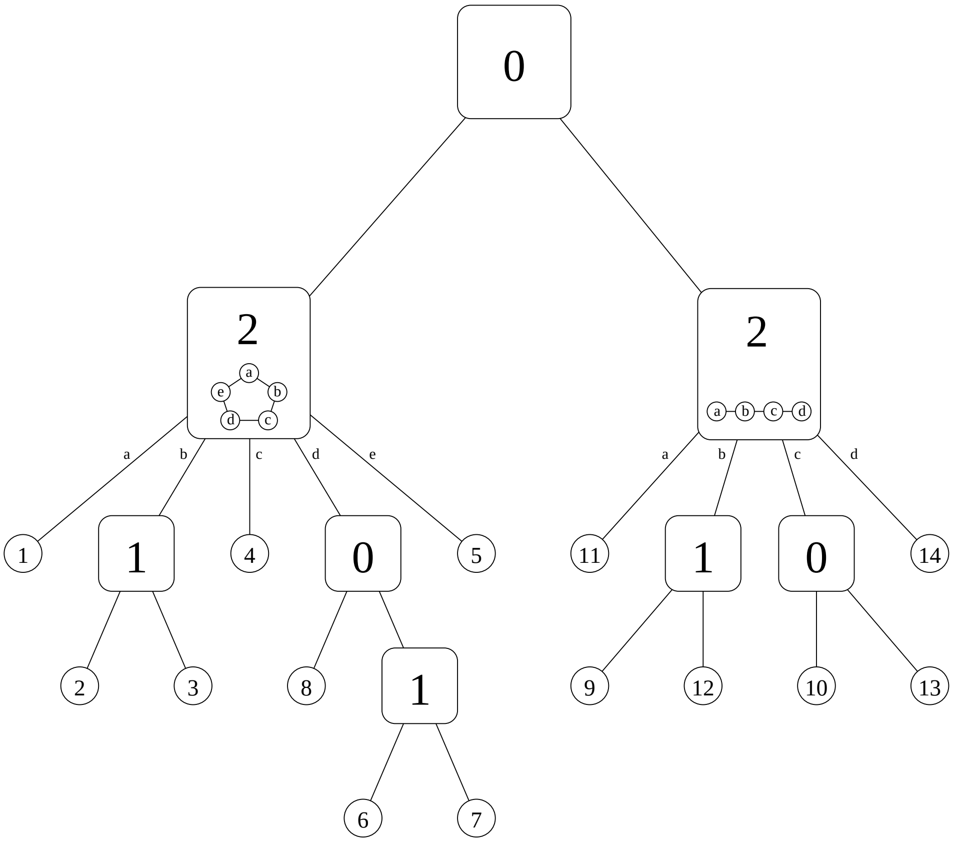

Fig. 1. A modular decomposition tree.

We recall the generic approach to solve an NP-complete graph problem using modular decomposition below in Section 2.4.

2.3 Cographs

A graph is a *cograph* if in its modular decomposition tree, all internal nodes are 0-nodes or 1-nodes. Cographs are exactly the P_4-free graphs, and their modular decomposition trees are called *cotrees*. The cotree representation allows to solve various NP-hard problems in linear time when restricted to cographs, among them the problems Maximum (Weight) Stable Set, Maximum (Weight) Clique, Minimum Coloring and Minimum Clique Cover [10,11]. In [12], a linear time recognition algorithm for cographs is presented. This algorithm outputs a cotree if the input graph is indeed a cograph, otherwise it outputs a P_4. This recognition algorithm is also part of the linear time modular decomposition algorithms in [26,14].

See [10–12] for more information on cographs and [8] for a survey on this and many other graph classes.

Often, algorithms exploiting the modular decomposition have the following structure. Let Π be an NP-complete graph problem to be solved on some graph class $\mathcal{G}$, such as Maximum Stable Set on (P_5,gem)-free graphs.

First the algorithm computes the modular decomposition tree T of the input graph G using one of the linear time algorithms. Then in a bottom up fashion the algorithm computes for each node x of T the optimal value(s) for the subgraph $G(x)$ of G induced by the set of all leaves of the subtree of T rooted at x. Thus the computation starts assigning the optimal value to the leaves (note that these represent induced subgraphs with one vertex). Then the algorithm computes the optimal value of an interior node x by using the optimal values of all children of x depending on the type of the node. Finally the optimal value of the root is the optimal value of Π for the input graph G. Note that various more complicated variants of this scenario can be useful.

Thus to specify such a modular decomposition based algorithm we only have to describe how to obtain the value for the leaves, and which formula to evaluate or which subproblem to solve on 0-nodes, 1-nodes and 2-nodes, using the values of all children as input. It is well-known how to do this for 0-nodes and 1-nodes for the NP-complete graph problems Maximum Weight Stable Set, Maximum Weight Clique, Minimum Coloring and Minimum Clique Cover from the corresponding cograph algorithm [10,11]. On the other hand to find out and solve the algorithmic problem for the 2-nodes, called the 2-node subproblem, for solving problem Π using modular decomposition can be quite challenging.

In this paper we shall study the problems Maximum (Weight) Stable Set, Maximum (Weight) Clique, Minimum Coloring and Minimum Clique Cover that have been studied in many other papers (see e.g. [19,21]). The most frequently studied NP-complete graph problem using modular decomposition is the Maximum (Weight) Stable Set problem. The use of modular decomposition has also been studied for the graph problems Maxcut [4], Treewidth and Min Fill-in [5]. More information on 2-node subproblems can be found in [28]. We study the four basic NP-complete graph problems on (P_5,gem)-free graphs in Sections 5 to 8.

As said, the problem Π is NP-complete; thus to guarantee polynomial running time of a modular decomposition based algorithm we have to restrict the possible input graphs to some suitable graph class $\mathcal{G}$; in particular, this should pose some restriction on the possible characteristic graphs assigned to the 2-nodes. If $\mathcal{G}$ is hereditary then the possible labels of 2-nodes of a modular decomposition tree of a graph $G \in \mathcal{G}$ are exactly the prime graphs in $\mathcal{G}$.

Consequently if we have a structural description of all the prime graphs of $\mathcal{G}$ then what remains is to show that the 2-node subproblem of Π can be solved sufficiently fast on all prime graphs in $\mathcal{G}$.

Polynomial time algorithms obtained by using modular decomposition are often practical in the following sense. They have neither huge hidden constants nor large exponents as is quite common for algorithms on graphs of bounded treewidth and graphs of bounded clique-width, when the treewidth or clique-width is not sufficiently small.

3 The Structure Theorem for (P_5,gem)-Free Graphs

To state the Structure Theorem of (P_5,gem)-free graphs we need to define three classes of (P_5,gem)-free graphs which contain all prime (P_5,gem)-free graphs.

Definition 2. *A graph $G = (V, E)$ is called* matched cobipartite *if its vertex set V is partitionable into two cliques K_1, K_2 with $|K_1| = |K_2|$ or $|K_1| = |K_2| - 1$ such that the edges between K_1 and K_2 form a matching and at most one vertex in K_1 and at most one vertex in K_2 are not covered by the matching. We denote the class of all matched cobipartite graphs by* class 1.

Definition 3. *A graph G is called* specific *if it is the complement of a prime induced subgraph of one of the three graphs in Figure 2.*
We denote the class of all specific graphs by class 2.

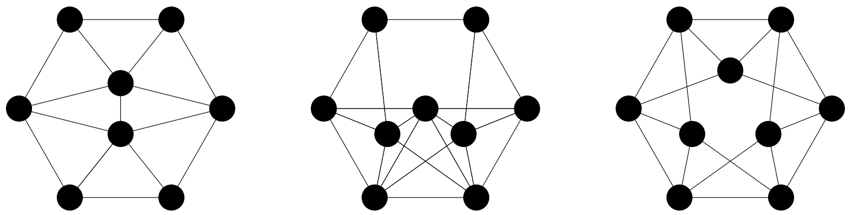

Fig. 2.

To establish the definition of class 3 we do need some more notions. A graph is *chordal* if it contains no induced cycles C_k, $k \geq 4$. See e.g. [21,8] for properties of chordal graphs. A graph is *cochordal* if its complement graph is chordal. A vertex v is *simplicial* in G if its neighborhood $N(v)$ in G is a clique. A vertex v is *cosimplicial* in G if it is simplicial in $\overline{G}$, i.e., its *non-neighbourhood* $\overline{N}(v) = V \setminus N[v]$ is a stable set in G. It is well-known that every chordal graph has a simplicial vertex and that such a vertex can be found in linear time.

We also need the following notion of substituting a C_5 into a vertex. For a graph G and a vertex v in G, let the result of the extension operation $ext(G, v)$ denote the graph G' resulting from G by replacing v with a C_5 $(v_1, v_2, v_3, v_4, v_5)$

of new vertices such that v_2, v_4 and v_5 have the same neighborhood in G as v and v_1, v_3 have only their C_5 neighbors, i.e., have degree 2 in G'. For a vertex set $U \subseteq V$ of G, let $ext(G, U)$ denote the result of applying repeatedly the extension operation to all vertices of U. Note that the resulting graph does not depend on the order of replacing U vertices.

Based on the operation $ext(G, v)$, we define the graph classes $\mathcal{C}_k$, $k \geq 0$, k being the number of C_5's contained in a graph $G \in \mathcal{C}_k$.

Definition 4. *For $k \geq 0$, let $\mathcal{C}_k$ be the class of prime graphs $G' = ext(G, Q)$ obtained by taking any (not necessarily prime) cochordal gem-free graph G and by then extending any clique Q of exactly k cosimplicial vertices of G. Thus, $\mathcal{C}_0$ is the class of prime cochordal gem-free graphs. We denote $\bigcup_{k=0}^{\infty} \mathcal{C}_k$ by* class 3*.*

It is shown in [6] that each graph in class 3 has neither $\overline{C_4} = 2K_2$ nor $\overline{C_6}$ as an induced subgraph. All graphs in class 3 are either cochordal or they contain a C_5. Class 3 is the crucial class for the problems to be considered in this paper. Therefore we list some useful properties of the graphs in this class (see [6]). We shall often rely on the partition property of the following lemma.

Lemma 5. *Let $G = (V, E)$ be a graph of* class 3*. Then G is a cochordal gem-free graph, or for every C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ of G, the vertex set V has a partition into $\{v_1, v_2, v_3, v_4, v_5\}$, the stable set A of 0-vertices for C and the set B of 3-vertices for C such that all vertices of B have the same neighbors in C, say v_2, v_4, v_5, and $G[B]$ is a cograph.*

Lemma 6. *Let $G = (V, E)$ be a connected graph of class 3 such that $G \neq C_5$ and G is not cochordal. Then it has a C_5, and for each C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ of G there are two nonadjacent vertices, say v_1 and v_3, of degree two and all other vertices of G have degree larger than two with the possible exception of 0-vertices for C. Furthermore the C_5's of G are pairwise vertex-disjoint.*

Now we are ready to state the structure theorem for (P_5,gem)-free graphs.

Theorem 7 ([6]). *A connected and co-connected graph G is (P_5,gem)-free if and only if*

(1) *all homogeneous sets of G are P_4-free, and*
(2) *for the characteristic graph G^* of G, one of the following conditions holds:*
  (2.1) *G^* is a matched cobipartite graph;*
  (2.2) *G^* is a specific graph;*
  (2.3) *there is a $k \geq 0$ such that G^* is in $\mathcal{C}_k$.*

The theorem does not provide a list of all prime (P_5,gem)-free graphs. Nevertheless it provides a characterization of all prime graphs that are (P_5,gem)-free that will turn out to be very useful for the design of algorithms using modular decomposition.

Remark 1. *If a graph is disconnected then it is (P_5,gem)-free if and only if each of its components is (P_5,gem)-free. If $\overline{G}$ is disconnected then G is (P_5,gem)-free if and only if G is a cograph.*

4 Recognition

A natural recognition algorithm for connected and co-connected (P_5,gem)-free graphs runs a linear time modular decomposition algorithm on the input graph and then verifies the conditions of the structure theorem (Theorem 7).

Let us discuss some easy cases first. If the input graph is disconnected then we run the algorithm on each component. If the input graph has a disconnected complementary graph then we accept if it is a cograph and otherwise we reject.

Now assume that the input graph is connected and co-connected. If its modular decomposition tree T has either more than one 2-node or if the only 2-node is not the root of the tree then we reject the input graph. If the connected and co-connected input graph G passed all subsequent tests then its modular decomposition tree has the right structure. All what remains to check is that the characteristic graph G^*, which is the graph assigned to the root of T, belongs to one of the classes 1 to 3.

In the following subsections we consider how to check whether a characteristic graph belongs to one of these three classes.

4.1 Matched cobipartite and specific graphs

It is not hard to see that the classes 1 and 2 can be recognized in linear time.

(1) *Is G matched cobipartite?*

(1.1) If the graph G has at most 3 vertices or less than $n^2/4$ edges then reject G.
(1.2) If not, partition the vertex set into two cliques K_1 and K_2 by applying the linear time recognition algorithm for bipartite graphs to $\overline{G}$ [21].
(1.3) Verify that $|K_1| = |K_2|$ or $|K_1| = |K_2| - 1$, and that the edges between K_1 and K_2 form a matching and at most one vertex in K_1 and at most one vertex in K_2 are not covered by the matching.

Clearly this algorithm can be implemented to run in linear time since any graph G passing the test (1.1) has $\Omega(n^2)$ edges. The correctness is based on the uniqueness of the partition in (1.2) (which is due to the co-connectedness of G).

(2) *Is G a specific graph?*

This can be checked in time $O(1)$ since all specific graphs have at most nine vertices.

4.2 Chordal cogem-free graphs

The interesting class concerning recognition is the class of cochordal gem-free graphs. We shall present an $O(n^2)$ time recognition algorithm for chordal cogem-free graphs, and thus obtain an $O(n^2)$ time recognition algorithm for cochordal gem-free graphs.

First, our recognition algorithm uses a linear time recognition algorithm for chordal graphs (see e.g. [21]) and rejects the input if it is not chordal.

In the remainder, we assume that the input graph $G = (V, E)$ is chordal. The algorithm proceeds by computing the following partition of the vertex set of G to be used to classify possible cogems and to decide whether G has a cogem.

Take some simplicial vertex v of the chordal graph G, i.e., $N[v]$ is a clique. Note that every chordal graph has a simplicial vertex and that a simplicial vertex in a chordal graph can be found in linear time [30]. Partition the vertex set V into $X = N[v]$ and $Y = V - N[v]$. Clearly X is a clique. If $G[Y]$ is not a cograph which can be checked in linear time then reject G since it has a cogem (induced by the vertex v and a P_4 of $G[Y]$). From now on we assume that $G[Y]$ is a cograph and that T is the cotree of $G[Y]$ obtained by the cograph recognition algorithm of [12].

We divide possible cogems of G into four types where $x, x_i \in X$ and $y, y_j \in Y$.

type 1: one vertex of X is an endvertex of the P_4: $x - y_1 - y_2 - y_3$
and all four vertices of the P_4 are nonadjacent to y_4

type 2: one vertex of X is a midvertex of the P_4: $y_1 - x - y_2 - y_3$
and all four vertices of the P_4 are nonadjacent to y_4

type 3: one edge of $G[X]$ is a wing of the P_4: $x_1 - x_2 - y_1 - y_2$
and all four vertices of the P_4 are nonadjacent to y_3

type 4: one edge of $G[X]$ is the center of the P_4: $y_1 - x_1 - x_2 - y_2$
and all four vertices of the P_4 are nonadjacent to y_3

We call a cogem a *standard cogem* with respect to $(N[v], Y)$ if its P_4 is $v - x - y_1 - y_2$ and all these four vertices are nonadjacent to y_3. Clearly this is a particular type 3 cogem.

Lemma 8. *G has a type 1, type 2 or type 3 cogem with respect to $(N[v], Y)$ if and only if it has a standard cogem with respect to $(N[v], Y)$.*

Proof. With the above notation we get:

If $x - y_1 - y_2 - y_3| \, y_4$ is a type 1 cogem then $v - x - y_1 - y_2| \, y_4$ is a standard cogem.

If $y_1 - x - y_2 - y_3| \, y_4$ is a type 2 cogem then $v - x - y_2 - y_3| \, y_4$ is a standard cogem.

If $x_1 - x_2 - y_1 - y_2| \, y_3$ is a type 3 cogem but not a standard cogem, then $v - x_2 - y_1 - y_2| \, y_3$ is a standard cogem. $\square$

Thus our algorithm may restrict its search to standard cogems and type 4 cogems in G. It will also rely on the fact that $G[Y]$ is a chordal cograph, and thus P_4- and C_4-free. Hence $G[Y]$ is a trivially perfect graph [20]; each of its connected induced subgraphs has a dominating vertex, i.e., a vertex adjacent to all other vertices of this connected induced subgraph [21,33,34].

Now we design an algorithm to find a cogem in a chordal graph. In stage one the algorithm recursively searches for a standard cogem and removes some vertices of Y that cannot be in any cogem until the remaining graph cannot contain a standard cogem. Then in a second stage the algorithm searches for a type 4 cogem. Consequently the current graph during the execution of the first stage of the algorithm is always an induced subgraph of G with vertex set $N[v] \cup Y'$ and $Y' \subseteq Y$.

Remark 2. *The lazy strategy postponing the search for a type 4 cogem until it has to be done only once (in stage two) allows us to obtain running time $O(n^2)$; a direct approach leads to an $O(nm)$ time algorithm.*

Definition 9. *Let $(N[v], Y')$ be a partition of the vertex set of $G[N[v] \cup Y']$, where $Y' \subseteq Y$ and $N[v]$ is a clique. Let $Y_1, Y_2, \ldots, Y_t$ be the components of the trivially perfect graph $G[Y']$ and let $X' = N(v)$. We call a vertex $x \in X'$* fully adjacent *to Y_i if $Y_i \subseteq N(x)$. We call a vertex $x \in X'$* non-adjacent *to Y_i if $Y_i \cap N(x) = \emptyset$. We call vertex $x \in X$* semi-adjacent *to Y_i if $\emptyset \neq Y_i \cap N(x) \neq Y_i$.*

One of the key principles of our algorithm is that during stage one it searches only for easy-to-find standard cogems and nothing else.

Easy-to-find cogem:
Suppose there is a vertex $x \in X'$ such that x is semi-adjacent to a component Y_i and not fully adjacent to some other component Y_j, $i \neq j$. Then G has a standard cogem $v - x - y_1 - y_2|\ y$ where $y \in Y_j$, $y_1, y_2 \in Y_i$.
(Note that there is an edge between $N(x) \cap Y_i$ and $Y_i \setminus N(x)$ since $G[Y_i]$ is connected. Each of these edges can be taken as the edge $\{y_1, y_2\}$ of the above mentioned standard cogem.)

If the algorithm finds an easy-to-find cogem in the current graph $G[N[v] \cup Y']$ then it rejects the input graph G. If there is no easy-to-find cogem in $G[N[v] \cup Y']$ then every vertex $x \in X'$ being semi-adjacent to some component of $G[Y']$ is fully adjacent to all other components. In addition there might be vertices $x \in X'$ such that for every component Y_i of $G[Y']$, x is either fully adjacent or non-adjacent to Y_i.

This motivates the following preprocessing step of our algorithm. In stage zero compute for every vertex $x \in X'$ and every 1-node t of the cotree T of $G[Y]$ whether x is fully adjacent, semi-adjacent or not adjacent to the connected subgraph of $G[Y]$ induced by the set of vertices assigned to the leaves of the subtree of T rooted at t. This is all information we need in stage one, since any component Y_i ever treated in stage one, with the exception of singletons, is obtained by the recursive removal of all dominating vertices from some larger component, which corresponds to taking a 1-node and obtaining a grandchild 1-node by the removal of all dominating vertices. The preprocessing can be done in time $O(n)$ per vertex $x \in X'$ where the computation is done in a bottom up fashion on the cotree T. Thus the overall running time of the preprocessing is $O(n^2)$.

During stage one the algorithm will compute and maintain for every vertex $x \in X'$ the number of components of $G[Y']$ to which x is fully adjacent, semi-adjacent and not adjacent, denoted by $\text{full}(x)$, $\text{semi}(x)$, and $\text{not}(x)$. Then the current graph $G[N[x] \cup Y']$ has an easy-to-find cogem if and only if there is a vertex $x \in X'$ such that $\text{semi}(x) \geq 2$, or $\text{semi}(x) = 1$ and $\text{not}(x) > 0$.

Now stage one of the algorithm starts with the chordal graph G and the partition $(N[v], Y)$ for some simplicial vertex v of G. During each step of stage one the algorithm either finds a cogem of G, and thus rejects the input, or it removes some vertices of Y' from the current graph.

Suppose the algorithm does not find an easy-to-find cogem while treating the graph $G[N[v] \cup Y']$ in stage one. Then every vertex $x \in X'$ fulfills $\text{semi}(x) = 0$, or $\text{semi}(x) = 1$ and $\text{not}(x) = 0$. Consequently, if $v - x - y_1 - y_2|\ y_3$ is a (not easy-to-find) standard cogem of $G[N[v] \cup Y']$ then the vertices y_1, y_2, y_3 belong to the same component of $G[Y']$. Additionally, if $y_1 - x_1 - x_2 - y_2|\ y_3$ is a type 4 cogem of $G[N[v] \cup Y']$ then either the vertices y_1, y_2, y_3 also belong to the same component of $G[Y']$, or to each component of $G[Y']$ both vertices x_1 and x_2 are either not adjacent or fully adjacent and y_1, y_2, y_3 belong to three different components of $G[Y']$. In the latter case the algorithm will not destroy the type 4 cogem of $G[N[v] \cup Y']$ when deleting vertices of Y' as long as it never removes all vertices of a component of $G[Y']$ (and thus our algorithm never removes the last vertex of a component).

Finally we have to describe how to choose the Y'-vertices to be removed at the end of a step in stage one. $G[Y]$ and thus each $G[Y']$ is a trivially perfect graph. Therefore each component of $G[Y']$ contains a dominating vertex. Clearly such a vertex cannot be in a standard cogem or in a type 4 cogem with y_1, y_2, y_3 in one component. Consequently, the algorithm removes all dominating vertices of each $G[Y']$ component, but it will never remove the last vertex of a component.

Analysing the running time of stage one we obtain: the algorithm removes a vertex in each round, thus there are at most n rounds in stage one. In each round, for each vertex $x \in X'$ there is direct access to the values of the variables full(x), semi(x) and not(x) corresponding to $(N[v], Y')$. Using this it can be decided in time $O(1)$ whether there is an easy-to-find cogem containing x. Using the cotree of a trivially perfect graph it is easy to find the dominating vertices of each component, to remove them and to update $full(x)$, $semi(x)$ and $not(x)$ for all $x \in X'$ in overall time $O(n)$ per round. Therefore the running time of stage one is $O(n^2)$.

Suppose the algorithm does not find an easy-to-find cogem in stage one. Then it terminates stage one when all components of $G[Y']$ consist of a single vertex. Thus stage two only has to verify whether the graph $G_s := G[N(v) \cup Y']$ obtained at the end of stage one has a type 4 cogem. Note that G_s is a split graph since $N(v)$ is a clique and Y' is an independent set.

The chordal distance-hereditary graphs, called ptolemaic graphs, are precisely the chordal gem-free graphs (see [8]). Consequently, if the complement of G_s is distance-hereditary then there is no (type 4) cogem in G_s, and thus G is a chordal cogem-free graph. Otherwise if the complement of G_s is not distance-hereditary, then since the complement of G_s is chordal, it contains a gem. Thus G contains a cogem and the algorithm rejects it. Finally, stage two can be implemented to run in time $O(n^2)$ by using a linear time recognition algorithm for distance-hereditary graphs.

Theorem 10. *There is an $O(n^2)$ algorithm to recognize chordal cogem-free graphs.*

Corollary 11. *There is an $O(n^2)$ algorithm to recognize cochordal gem-free graphs.*

4.3 Class 3

First the algorithm checks whether the input graph G is prime using a linear time modular decomposition algorithm.

The main part of the following algorithm destroys all C_5's of the graph by deleting one of the degree two vertices of each C_5, and thus the remaining graph should be C_5-free. Then the algorithm verifies whether the remaining graph is (P_5,gem)-free, i.e., cochordal and gem-free.

(0) If G is cochordal then accept if it is gem-free and reject if it contains a gem. If $G = C_5$ then accept. Continue if G is neither cochordal nor the C_5.
(1) Determine the set of all vertices of degree 2.
(2) Try to extend each pair of non adjacent degree 2 vertices with a common neighbour to a C_5 (by checking whether their non common neighbours are adjacent).
(3) For each C_5 $C = (x_1, x_2, x_3, x_4, x_5)$ determined in (2)
(3.1) determine the set A of all 0-vertices of C, and
(3.2) determine $B = V \setminus (A \cup C)$ and verify that $N(x_i) = B$ for all vertices x_i of degree larger than two in C.
(4) Reject the graph G if the condition of (3.2) is not satisfied for some of the C_5's determined in (2). Otherwise construct the graph G' by removing precisely one of the two degree two vertices of each C_5 determined in (2) from the graph G.
(5) Check whether the remaining graph G' is cochordal, and if so whether it is also gem-free. If yes then accept G. Otherwise reject G.

Theorem 12. *There is an $O(n^2)$ algorithm to recognize class 3 graphs.*

Proof. First let us show that the algorithm is correct. Clearly the input graph is in class 3 if it is accepted in (0). If the graph G is in class 3, but it has not been accepted in (0), then it will be accepted in (5) since the graph G' obtained from G by destroying all C_5's via the removal of one vertex must be cochordal gem-free. (Note that the only complement of a chordless cycle that a graph of class 3 may contain as induced subgraph is a C_5.)

Combining some results of [6] one can easily obtain that, if a graph is prime $(2K_2, \overline{C_6}$,gem)-free then it is in class 3 or it is a specific graph. Studying in more detail the corresponding proof in [6], one can obtain that there is precisely one prime $(2K_2, \overline{C_6}$,gem)-free graph not belonging to class 3, and that our algorithm will reject this specific graph.

Suppose our algorithm fails on the input graph G. Thus the prime input graph G does not belong to class 3, and thus it has a $2K_2$, $\overline{C_6}$ or a gem as induced subgraph. Let C_5 $C = (x_1, x_2, x_3, x_4, x_5)$ be a C_5 of G determined in (2). Let x_1 and x_3 be its vertices of degree two and let C satisfy the condition in (3.2). Then x_1 and x_3 cannot be vertices of a $\overline{C_6}$ since their degree is two. Furthermore none of them is a vertex of a gem in G, since their neighbours are non adjacent. However they may be vertices of a $2K_2$. If x_1 is in a $2K_2$ then there is a $2K_2$ in G induced by $\{x_1, x_5, a_1, a_2\}$ with $a_1, a_2 \in A$, and thus $\{x_3, x_4, a_1, a_2\}$ also induces a $2K_2$ in G. Hence G has a $2K_2$ containing x_1 if and only if G has a $2K_2$ containing x_3. Furthermore if such a $2K_2$ exists then A is not a stable set. Finally notice that if $G[B]$ is not a cograph then G has a gem consisting of any vertex of $\{v_2, v_4, v_5\}$ and a P_4 in $G[B]$.

Therefore the graph G' constructed in (4) has no $2K_2$, $\overline{C_6}$ and gem as induced subgraph if and only if the input graph G has no $2K_2$, $\overline{C_6}$ and gem as induced subgraph. Thus since we consider an input G not belonging to class 3 the graph G' has a $2K_2$, $\overline{C_6}$ or a gem as induced subgraph. Since G is accepted it does not fail the test in (5). Hence G' is cochordal gem-free, and thus it has neither $2K_2$, nor $\overline{C_6}$ nor a gem as induced subgraph. Thus G is not prime $(2K_2, \overline{C_6}$,gem)-free, contradicting our choice of G.

By Corollary 11, (0) and (5) can be implemented to run in time $O(n^2)$. There are $O(n)$ pairs of vertices of degree two with common neighbour and they can be extended to a C_5, if possible, in time $O(1)$. (3.1) and (3.2) can be executed in overall time $O(n+m)$ for all C_5's determined in (2) by passing through the adjacency lists of all vertices occurring in a C_5 that has to be checked. Thus the overall running time is $O(n^2)$. □

4.4 (P_5,gem)-free graphs

The recognition algorithm for (P_5,gem)-free graphs starts with a linear time algorithm to compute the modular decomposition tree of the input graph. If the tree contains no 2-node, and thus the graph is a cograph, then the algorithm accepts the input. Otherwise the algorithm has to check that each connected component of the input graph is either a cograph or has a modular decomposition tree where only the root is a 2-node. All this can be done in linear time by inspection of the modular decomposition tree of the input graph. Finally the algorithm checks whether the characteristic graph of each connected component belongs to one of the three classes of the structure theorem using the algorithms described above.

Theorem 13. *There is a recognition algorithm for (P_5,gem)-free graphs with running time $O(n^2)$.*

It is an interesting open question whether there is a linear time algorithm to recognize (P_5,gem)-free graphs.

5 Maximum Weight Stable Set

The Maximum Weight Stable Set problem is the following: Given a graph $G = (V, E)$ and a vertex weight function $w : V \to \mathbb{N}$, find the maximum weight of a stable set in G denoted by $\alpha_w(G)$. (Note that nonnegative weights are naturally coming up in the 2-node subproblem of the Maximum Stable Set problem.)

As discussed in the previous section, all we have to specify are the (initial) values of the leaves of the modular decomposition tree T of the input graph G and the subproblems to be solved for the interior nodes of T.

Subproblems
For each node x of the modular decomposition tree T of G (in the above described bottom up fashion) compute $\alpha_w(G(x))$ as follows:
If x is a **leaf** of T and v the vertex assigned to x then $\alpha_w(G(x)) := w(v)$;
If x is a **0-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\alpha_w(G(x)) := \sum_{i=1}^{r} \alpha_w(G(x_i))$;
If x is a **1-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\alpha_w(G(x)) := \max\limits_{i=1,\ldots,r} \alpha_w(G(x_i))$;
If x is a **2-node** of T, G^* the characteristic graph assigned to x and $x_1, \ldots x_r$ the children of x then assign to the vertex set V^* of G^* the weight function $w^* : V^* \to \mathbb{N}$ such that $w^*(v_i) := \alpha_w(G(x_i))$ where the child x_i of x is assigned to the vertex v_i of G^*. Finally compute $\alpha_w(G(x))$ as the maximum weight of a stable set in the prime graph G^* with vertex weight function w^*.

Therefore the subproblem to be solved at a 2-node, called the 2-node subproblem, is the Maximum Weight Stable Set problem where the input is a prime graph G^* with vertex weight function w^*.

Let us make a short detour to discuss the relation between the original problem and the 2-node subproblem. For the Maximum Weight Stable Set problem the subproblem to be solved at a 2-node is also the Maximum Weight Stable Set problem. Thus the original problem and the 2-node subproblem are identical. This might actually be the reason that many papers on the use of modular decomposition for NP-complete problems study the Maximum (Weight) Stable Set problem.

Nevertheless there are only few of the well-known NP-complete graph problems for which the 2-node subproblem is identical to the original problem, and there are problems for which the 2-node subproblem seems to be much more complicated than the original one, such as the problems Hamiltonian Circuit and Bandwidth. It is likely that evaluating the difference between original and 2-node subproblem is important for knowing whether modular decomposition works well with a certain problem Π or not; see [27] for an early paper on this subject.

Finally we have to show how the algorithm computes the maximum weight of a stable set in a prime (P_5,gem)-free graph. Based on Theorem 7 there are three classes of prime graphs. Given a characteristic graph G^*, the algorithm recognizes in linear time the class to which G^* belongs using the linear time recognition algorithms for the classes 1 and 2 of Subsection 4.1. Now let us consider the Maximum Weight Stable Set problem for the three classes of (P_5,gem)-free prime graphs.

Class 1: Matched cobipartite graphs

The graph $G^* = G[V^*]$ is cobipartite and prime. Thus $|V^*| \geq 4$, G^* is not complete and each maximal stable set of G^* has cardinality two.

Let K_1, K_2 be a partition of the vertex set V^* of G^* into two cliques computed by the linear time recognition algorithm for class 1. Let a_1, a_2 be two vertices of highest weight in K_1 and let b_1, b_2 be two vertices of highest weight in K_2. Since each vertex has at most one neighbour in the other clique we conclude that

$$\alpha_{w^*}(G^*) = \max\{w^*(a_i) + w^*(b_j) : \ i, j \in \{1, 2\}, \{a_i, b_j\} \notin E\}.$$

Thus $\alpha_{w^*}(G^*)$ can be computed in linear time.

Class 2: Specific graphs

Since specific graphs have at most 9 vertices the maximum weight of a stable set in a specific graph can be computed in $O(1)$ time.

Class 3

Finally we study the Maximum Weight Stable Set problem on class 3.

First we reconsider the recognition of class 3 graphs. Given a class 3 graph G, our algorithms will need all C_5's, and thus they also know the smallest k such that $G \in \mathcal{C}_k$.

Theorem 14. *There is a linear time algorithm to compute all (pairwise vertex-disjoint) C_5's for graphs of class 3.*

Proof. First, compute the set of all degree two vertices of G. Then for each pair of vertices u and v of degree two with a common neighbour, check whether $N[u] \cup N[v]$ induces a C_5 of G.

All C_5's of G can be found in this way. Since any vertex of degree two has a common neighbour with at most one other vertex of degree two by the definition of class 3, all C_5's are pairwise vertex-disjoint.

This also implies that the running time of the algorithm is linear. $\square$

Theorem 15. *The Maximum Weight Stable Set problem can be solved in linear time for all graphs of class 3.*

Proof. First compute in linear time all vertices of degree two and all C_5's of G^* using Theorem 14. If $G^* = (V^*, E^*)$ is cochordal then S is a maximum weight stable set of G^* if and only if S is a maximum weight clique of the chordal graph $\overline{G^*}$ with weight function w^*. A perfect elimination ordering $v_1, v_2, \dots, v_n$ of $\overline{G^*}$ can be computed in time $O(|V^*| + |E^*|)$ when the graph $G^* = (V^*, E^*)$ is given [26]. Now each maximal stable set S of G^* is a maximal clique of the chordal graph $\overline{G^*}$ and thus $S = N^i{}_{\overline{G^*}}[v_i] = N_{\overline{G^*}}[v_i] \cap \{v_i, v_{i+1}, \dots, v_n\}$ for some $i \in \{1, 2, \dots, n\}$. Thus $w^*(S) = W^i - \sum_{u \in N^i{}_{G^*}(v_i)} w^*(u)$, where $W^i = \sum_{j=i}^{n} w^*(v_j)$. Thus $\omega_{w^*}(\overline{G^*}) = \alpha_{w^*}(G^*)$ can be computed in linear time.

If G^* is a C_5 then the maximum weight of a stable set can be computed in time $O(1)$. Otherwise, for every C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ there are two nonadjacent vertices v_1 and v_3 of degree two and three vertices v_2, v_4, v_5 of degree at least three. Let S be a stable set of G^*. If S contains a vertex of degree greater than two of a C_5 then there is only one C_5, say C, of G^* containing all these degree greater than two vertices of S. Hence any maximal stable set of this type contains the stable set A of all 0-vertices for C and two nonadjacent vertices of C. There are $O(n)$ maximal stable sets of this type and all their weights can be computed in overall time $O(n)$ since $w^*(S) = w^*(A) + w^*(x) + w^*(y)$ where x and y are two nonadjacent vertices of a C_5.

If $X \subseteq V^*$ contains no vertex of degree greater than two of a C_5 in G^* then X is a maximal stable set of the graph G' obtained from G^* by removing all vertices of degree greater than two within a C_5 of G^*. Clearly G' is cochordal and thus $\alpha_{w^*}(G')$ can be computed in linear time as shown above. $\square$

Corollary 16. *There is a linear time algorithm solving the maximum weight stable set problem on cochordal graphs.*

We note that the modular decomposition tree of a connected (P_5,gem)-graph contains at most one 2-node. Summarizing we obtain the following main result of this section.

Theorem 17. *There is a linear time algorithm to compute the maximum weight of a stable set of a (P_5,gem)-free graph G with vertex weight function w.*

It is not difficult to modify the algorithm such that it computes a maximum weight stable set of (P_5,gem)-free graphs within the same time bound. First the algorithms for class 1, 2 and 3 can be easily modified such that they compute a maximum weight stable set. Then the operations used for the computation in the modular decomposition tree can be modified such that for every node x instead of $\alpha_w(G(x))$ a maximum weight stable set of $G(x)$ is computed.

6 Maximum Weight Clique

The Maximum Weight Clique problem is the following: Given a graph $G = (V, E)$ and a vertex weight function $w : V \to \mathbb{N}$, find the maximum weight of a clique in G, denoted $\omega_w(G)$.

The following simple $O(nm)$ time algorithm computes a maximum weight clique for gem-free graphs: For every vertex v, determine a maximum weight clique of the cograph induced by $N_G[v]$, and then maximize over all these values.

We present a linear time algorithm to solve the Maximum Weight Clique problem on (P_5,gem)-free graphs. The approach is similar to the one in the previous section.

Subproblems
For each node x of the modular decomposition tree T of G compute $\omega_w(G(x))$ as follows:
If x is a **leaf** of T and v the vertex assigned to x then $\omega_w(G(x)) := w(v)$;
If x is a **0-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\omega_w(G(x)) := \max\limits_{i=1,\ldots,r} \omega_w(G(x_i))$;
If x is a **1-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\omega_w(G(x)) := \sum_{i=1}^{r} \omega_w(G(x_i))$;
If x is a **2-node** of T, G^* the characteristic graph assigned to x and $x_1, \ldots x_r$

the children of x
then assign to the vertex set V^* of G^* the weight function $w^*: V^* \to \mathbb{N}$ such that $w^*(v_i) := \omega_w(G(x_i))$ where the child x_i of x is assigned to the vertex v_i of G^*. Finally compute $\omega_w(G(x))$ as the maximum weight of a clique in the prime graph G^* with vertex weight function w^*.

Class 1: Matched cobipartite graphs

The graph $G^* = G[V^*]$ is cobipartite and prime. Thus $|V^*| \geq 4$, G^* is not complete and each maximal clique of G^* has cardinality at least two.

Let K_1, K_2 be a partition of the vertex set V^* of G^* into two cliques. Then each maximal clique of G^* is either K_1, K_2 or an edge of G^*.

Hence $\omega_{w^*}(G^*)$ can be computed in linear time.

Class 2: Specific graphs

Since specific graphs have at most 9 vertices the maximum weight of a clique in a specific graph can be computed by an $O(1)$ time algorithm.

Class 3

First we consider the Maximum Weight Clique problem on cochordal graphs.

Lemma 18. *The Maximum Weight Clique problem can be solved by a linear time algorithm for cochordal graphs.*

Proof. Using the linear time algorithm of Frank [17] to compute the maximum weight of a stable set of a chordal graph G, we obtain an $O(n^2)$ algorithm to compute the maximum weight of a clique in a cochordal graph $\overline{G}$ since $\omega_w(\overline{G}) = \alpha_w(G)$. We shall modify Frank's algorithm such that it can be used to compute $\omega_w(G)$ of a cochordal graph in linear time.

This is Frank's algorithm: First it computes a perfect elimination ordering $v_1, v_2, \ldots, v_n$ of the chordal input graph $G = (V, E)$ where $w(v_i)$ is the non-negative integer weight of v_i $(1 \leq i \leq n)$.

Then a maximum weight clique of $\overline{G}$ is constructed as follows. Initially, let $c\_w(v_i) = w(v_i)$, for all $1 \leq i \leq n$. For each i from 1 to n, if $c\_w(v_i) > 0$ then colour v_i red, and subtract $c\_w(v_i)$ from $c\_w(v_j)$ for all $v_j \in \{v_i\} \cup (N(v_i) \cap \{v_{i+1}, \ldots, v_n\})$. When all vertices have been processed, set $I = \emptyset$ and, for each i from n down to 1, if v_i is red and not adjacent to any vertex of I then $I = I \cup \{v_i\}$. When all vertices have been processed, the algorithm terminates and outputs the maximum weight stable set I of (G, w).

Our goal is to simulate Frank's algorithm applied to $\overline{G}$ for a cochordal input graph $G = (V, E)$ such that the running time is linear in the size of the cochordal graph G. First our algorithm computes a perfect elimination ordering $v_1, v_2, \ldots, v_n$ of $\overline{G}$ in linear time (see [26]). Let $w(v_i)$ be the nonnegative integer weights of v_i $(1 \leq i \leq n)$.

The maximum weight of a clique of G is constructed as follows. Initially, let $W' = 0$ and $s(v_i) = 0$ for all i $(1 \leq i \leq n)$. For each i from 1 to n, if $w(v_i) - W' + s(v_i) > 0$ then colour v_i red, set $W' = w(v_i) + s(v_i)$ and add $w(v_i) - W' + s(v_i)$ to $s(v_j)$ for all $v_j \in (N(v_i) \cap \{v_{i+1}, \ldots, v_n\})$.

When all vertices have been processed, set $K = \emptyset$ and, for each i from n down to 1, if v_i is red and adjacent to all vertices of K then $K = K \cup \{v_i\}$. Finally the algorithm outputs the maximum weight clique K of (G, w).

Clearly our algorithm runs in linear time. Its correctness follows from the fact that when treating the vertex v_i, the difference $W' - s(v_i)$ is precisely the value the original Frank algorithm applied to the complement of G would have subtracted from $c\_w(v_i)$ up to the point when it treats v_i. Thus our algorithm simulates Frank's algorithm on $\overline{G}$, and thus it is correct. □

Theorem 19. *The Maximum Weight Clique can be solved in linear time for all graphs of class 3.*

Proof. First compute in linear time all vertices of degree two and all C_5's of G^* using the algorithm of Theorem 14.

If G^* is cochordal then by Lemma 18 there is a linear time algorithm to compute the maximum weight of a clique of G^*. If G^* is a C_5 then the maximum weight of a clique can be computed in time $O(1)$.

Otherwise, let u be a vertex of degree two of a C_5. Then there are two maximal cliques of G containing u, and each of them corresponds to an edge incident to u. The maximum weight of all these cliques can be computed in time $O(m)$. All other maximal cliques of G^* are maximal cliques of the cochordal graph obtained by removing all degree two vertices of C_5's from G^*. We have already shown that their maximum weight can be found in linear time. □

Summarizing we obtain the following main result of this section.

Theorem 20. *There is a linear time algorithm to compute the maximum weight of a clique of a (P_5,gem)-free graph G with vertex weight function w.*

7 Minimum Coloring

The Minimum Coloring problem is the following: Given a graph $G = (V, E)$ determine the smallest number of colors in a coloring of G, denoted by $\chi(G)$. A weighted version arises as 2-node subproblem: Given a graph G^* and a vertex weight function $w^* : V \to \mathbb{N}$, find the smallest k for which (G^*, w^*) has a weighted k-coloring, i.e., compute $\chi_{w^*}(G^*)$.

Minimum Coloring is not LinEMSOL($\tau_{1,L}$) definable. Nevertheless there is a polynomial time algorithm for graphs of bounded clique-width [23]. However this algorithm is only of theoretical interest. For graphs of clique-width at most five (and currently five is the best upper bound known for the maximum clique-width of (P_5,gem)-free graphs [7]), the exponent r of the running time $O(n^r)$ of this algorithm is larger than 2000.

We present a linear time algorithm to solve the Minimum Coloring problem on (P_5,gem)-free graphs.

Subproblems
For each node x of the modular decomposition tree T of G compute $\chi(G(x))$ as follows:

If x is a **leaf** of T then $\chi(G(x)) := 1$;
If x is a **0-node** of T and $x_1, x_2, \dots x_r$ its children
then $\chi(G(x)) := \max_{i=1,\dots,r} \chi(G(x_i))$;
If x is a **1-node** of T and $x_1, x_2, \dots x_r$ its children
then $\chi(G(x)) := \sum_{i=1}^{r} \chi(G(x_i))$;
If x is a **2-node** of T, G^* the characteristic graph assigned to x and $x_1, \dots x_r$ the children of x
then assign to the vertex set V^* of G^* the weight function $w^* : V^* \to \mathbb{N}$ such that $w^*(v_i) := \chi(G(x_i))$. Finally compute $\chi(G(x)) := \chi_{w^*}(G^*)$.

As already mentioned, all our problems are linear time solvable for cographs, and thus we only have to study the complexity of the computation on all 2-nodes (and there is precisely one of them). The 2-node problem is a weighted coloring problem where the sum of the vertex weights of the graph G^* is always at most the number of vertices of the given (P_5,gem)-free graph G.

We shall rely on the following lemma.

Lemma 21. *Let G be a perfect graph and w be a vertex weight function of G. Then $\chi_w(G) = \omega_w(G)$ and $\kappa_w(G) = \alpha_w(G)$.*

Proof. Let G' be the graph obtained from G by substituting each vertex v of G by a clique of cardinality $w(v)$. It is not hard to see that any weighted coloring of (G, w) corresponds to a coloring of G' and vice versa. Thus $\chi_w(G) = \chi(G')$. Furthermore it is not hard to show that $\omega_w(G) = \omega(G')$.

Let G be perfect. Then its complement is perfect by Lovász's Perfect Graph Theorem [24]. $\overline{G'}$ is obtained from the perfect graph $\overline{G}$ by vertex multiplication, and thus it is perfect [24]. Finally G' as the complement of the perfect graph $\overline{G'}$ is perfect by the Perfect Graph Theorem. Since G' is perfect we have $\chi(G') = \omega(G')$ and thus $\chi_w(G) = \omega_w(G)$.

Similarly, since $\overline{G'}$ is perfect we obtain $\chi_w(\overline{G}) = \omega_w(\overline{G})$. Hence $\kappa_w(G) = \alpha_w(G)$. □

Now let us assume that N is the sum of the weights of the given prime graph G^*; as we already mentioned, $N \leq |V(G)|$.

Class 1: Matched cobipartite graphs

The graph G^* is cobipartite and thus perfect. By Lemma 21, $\chi_{w^*}(G^*) = \omega_{w^*}(G^*)$. The Maximum Weight Clique problem on matched cobipartite graphs has been considered in the previous section. We established a linear time algorithm to compute $\omega_{w^*}(G^*) = \chi_{w^*}(G^*)$.

Class 2: Specific graphs

We will show below that we can solve the weighted coloring problem in $O(1)$ time for a graph of size $O(1)$; this assumes that we can do computations with numbers expressing weights in $O(1)$ time per operation. Clearly, this implies that $\chi_{w^*}(G^*)$ can be computed in $O(1)$ time for each specific graph G^*.

Consider the graph G^* of size $O(1)$, with weights w^*. We formulate the problem to compute $\chi_{w^*}(G^*)$ as an integer linear programming problem, and then argue that this ILP can be solved in constant time.

Let $\mathcal{I}$ be the collection of all maximal independent sets of G^*. We build an integer linear programming with for each $I \in \mathcal{I}$ a variable x_I, as follows.

$$\text{minimize} \sum_{I\in\mathcal{I}} x_I \tag{1}$$

$$\text{such that}$$

$$\sum_{I\in\mathcal{I}:v\in I} x_I \geq w(v) \quad \text{for all } v \in V \tag{2}$$

$$x_I \in \{0,1,2,\ldots\} \quad \text{for all } I \in \mathcal{I} \tag{3}$$

With $\vec{x}$ we denote a vector containing for each $I \in \mathcal{I}$ a value x_I.

Let z be the optimal value of this ILP. z equals the minimum number of colors needed for (G^*, w^*). If we have a coloring of (G^*, w^*) with a minimum number of colors, then assign to each color one maximal independent set $I \in \mathcal{I}$, such that all vertices that received this color belong to I. (For each color, the vertices that have received that color form an independent set I'; we assign a maximal independent that contains I' as a subset to the color.) Let x_I be the number of colors assigned to I. Clearly, x_I is a non-negative integer. For each $v \in V$, as v has $w(v)$ colors, we have $\sum_{I\in\mathcal{I}:v\in I} x_I \geq w(v)$. $\sum_{I\in\mathcal{I}} x_I$ equals the total number of colors. Conversely, suppose we have an optimal solution x_I of the ILP. For each $I \in \mathcal{I}$, we can take a set of x_I unique colors, and use these colors to color the vertices in x_I. As I is independent, this gives a proper coloring, and as $\sum_{I\in\mathcal{I}:v\in I} x_I \geq w(v)$, each vertex has sufficiently many colors available. So, this gives a coloring of (G^*, w^*) with z colors.

The relaxation of the ILP is the linear program, obtained by dropping the integer condition (3):

$$\text{minimize} \sum_{I\in\mathcal{I}} x_I \tag{4}$$

$$\text{such that}$$

$$\sum_{I\in\mathcal{I}:v\in I} x_I \geq w(v) \quad \text{for all } v \in V \tag{5}$$

Let $\vec{x}'$ be an optimal solution of this relaxation, with value $z' = \sum_{I\in\mathcal{I}} x'_I$.

Note that the linear program has a constant number of variables (namely, the number of maximal independent sets of G^*) and a constant number of constraints (at most one per vertex of G^*), and hence can be solved in constant time. (E.g., one can just enumerate all corners of the polyhedron spanned by program, and take the optimal one.) Note that we can write the linear program in the form $\max\{cx \mid Ax \leq b\}$, such that each element of A is either 0 or 1. Let Δ be the maximum value of a subdeterminant of this matrix A. It follows that Δ is bounded by a constant. Write $n = |\mathcal{I}|$. We remark that a computer experiment that we have carried out revealed that $\Delta \leq 3$ when G^* is a specific graph.

We now can use a result of Cook, Gerards, Schrijver, and Tardos, see Theorem 17.2 from [31]. This theorem tells us that the ILP has an optimal solution $\vec{x}''$, such that for each $I \in \mathcal{I}$, $|x'_I - x''_I| \leq n\Delta$.

Thus, the following is an algorithm that finds the optimal solution to the ILP (and hence the number of colors needed for (G^*, w^*)) in constant time. First, find an optimal solution $\vec{x}'$ of the relaxation. Then, enumerate all integer vectors $\vec{x}''$ with for all $I \in \mathcal{I}$, $|x'_I - x''_I| \leq n\Delta$. For each such $\vec{x}''$, check if it fulfils conditions (2), and select the solution vector that fulfils the conditions with the minimum value. By Theorem 17.2 from [31], this is an optimal solution of the ILP. This method takes constant time, as n and Δ are bounded by constants, and thus 'only' a constant number of vectors have to be checked, and each is of constant size.

A straightforward implementation of this procedure would not be practical, as more than $(n\Delta)^n$ vectors are checked, with n the number of maximal independent sets in one of the specific graphs. In a practical setting, one could first solve the linear program, and use that value as starting point in a branch and bound procedure.

As we have an $O(1)$ algorithm for weighted coloring on graphs of size bounded by a constant, it follows that Minimum Coloring can be solved in linear time for graphs whose modular decomposition has a constant upper bound on the size of the characteristic graphs. This improves upon a remark by McDiarmid and Reed [29] who noticed that weighted coloring can be solved in polynomial time on constant sized graphs.

Class 3

Let G^* be a graph of class 3 with weight function w^*. First compute in linear time all vertices of degree two and all C_5's of G^* using the algorithm of Theorem 14.

If G^* is C_5-free, and thus $G^* \in \mathcal{C}_0$ and G^* cochordal, $\chi_{w^*}(G^*) = \omega_{w^*}(G^*)$ can be computed by the linear time algorithm for the maximum weight clique problem presented in the previous section.

Otherwise $G^* \in \mathcal{C}_k$ and $k \geq 1$. If $G^* = C_5$ then with the technique applied to specific graphs $\chi_{w^*}(G^*)$ can be computed by an $O(1)$ algorithm.

Finally let $C = (v_1, v_2, v_3, v_4, v_5)$ be a C_5 of G^* and let v_1 and v_3 be the vertices of degree two of C. By Lemma 5, the set A of 0-vertices for C is a stable set, $B = V^* \setminus (C \cup A) = N(v_2) \setminus C = N(v_3) \setminus C = N(v_5) \setminus C$, and

$G^*[B]$ is a cograph. Therefore there are precisely four maximal stable sets of G^* containing at least one of the three vertices v_2, v_4, v_5, namely $\{v_1, v_4\} \cup A$, $\{v_2, v_4\} \cup A$, $\{v_2, v_5\} \cup A$ and $\{v_3, v_5\} \cup A$. All other maximal stable sets of G^* are supersets of the maximal stable set $\{v_1, v_3\}$ of C. More precisely, $\{v_1, v_3\} \cup A'$ is a maximal stable set of G^* if and only if A' is a maximal stable set of $G^* - C$. (Note that $\{v_1, v_3\} \cup A$ is a stable set of G^*.)

Lemma 22. *Let $k \geq 1$, $G^* \in \mathcal{C}_k$ (possibly $G^* = C_5$). Let $C = (v_1, v_2, v_3, v_4, v_5)$ be a C_5 in G^* such that vertices v_1 and v_3 have degree two. Let w^* be the vertex weight function of G^*. Then there is a minimum weight coloring $\mathcal{S}^*$ of (G^*, w^*) with precisely* $\max(w^*(v_2), w^*(v_4) + w^*(v_5))$ *stable sets containing at least one of the vertices of $\{v_2, v_4, v_5\}$.*

Proof. Let $\mathcal{S}$ be any minimum weight coloring of (G^*, w^*). Let $C = (v_1, v_2, v_3, v_4, v_5)$ be a C_5 in G^* such that, if $G^* \neq C_5$, v_1 and v_3 are its vertices of degree two. Since $N(v_1) \setminus C = N(v_3) \setminus C = \emptyset$ and $N(v_2) \setminus C = N(v_3) \setminus C = N(v_5) \setminus C = B$ we may assume that every stable set of $\mathcal{S}$ contains either none or two vertices of C.

Therefore we study weighted colorings of a C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ of G^* with vertex weights w^*, where all stable sets are non-edges of C and call them partial weight colorings of C. Note that any weighted coloring of $C = (v_1, v_2, v_3, v_4, v_5)$ must contain at least $w^*(v_2)$ stable sets containing v_2, and it must contain $w^*(v_4) + w^*(v_5)$ stable sets containing v_4 or v_5.

Let $\mathcal{S}'$ be a weighted coloring of G^* containing the smallest possible number of stable sets S with $S \cap \{v_2, v_4, v_5\} \neq \emptyset$. Let q be the number of stable sets S of $\mathcal{S}'$ satisfying $S \cap \{v_2, v_4, v_5\} \neq \emptyset$ and suppose that, contrary to the statement of the lemma, $q > \max(w^*(v_2), w^*(v_4) + w^*(v_5))$. Let $c(v)$ be the number of stable sets of $\mathcal{S}'$ containing the vertex v. Then $q > w^*(v_4) + w^*(v_5)$ implies $c(v_4) > w^*(v_4)$ or $c(v_5) > w^*(v_5)$. Without loss of generality we may assume $c(v_4) > w^*(v_4)$. Hence there is a stable set $S' \in \mathcal{S}'$ containing v_4. Consequently either $S' \subseteq \{v_2, v_4\} \cup A$ or $S' \subseteq \{v_1, v_4\} \cup A$. In both cases we replace the stable set S' of the weighted coloring $\mathcal{S}'$ of G^* by $\{v_1, v_3\} \cup A$. By the replacement we decrement by one the number of stable sets containing v_4 and possibly the number of stable sets containing v_2. Thus we obtain a new weighted coloring $\mathcal{S}''$ of G^* with $q - 1$ stable sets S with $S \cap \{v_2, v_4, v_5\} \neq \emptyset$. This contradicts the choice of q.

Consequently $q = \max(w^*(v_2), w^*(v_4) + w^*(v_5))$. ☐

Corollary 23. *Let G be a C_5 with vertices $(v_1, v_2, v_3, v_4, v_5)$, and w be the vertex weight function of G. Then there is a minimum weighted coloring of G with precisely* $\max(w(v_2), w(v_4) + w(v_5))$ *stable sets containing at least one of the vertices of $\{v_2, v_4, v_5\}$.*

To extend any partial weight coloring of a C_5 to G^* only two parameters are important:

- the number of copies of $\{v_1, v_3\}$ of the partial weight coloring, denoted by s, and
- the number of non-edges of the C_5 different from $\{v_1, v_3\}$ of the partial weight coloring, denoted by t.

For each of the s stable sets $\{v_1, v_3\}$ of the C_5, each $\{v_1, v_3\} \cup A'$ where A' is some maximal stable set of $G^* - C$ is a maximal stable set of G^*. For each of the t non-edges S of the C_5 being different from $\{v_1, v_3\}$, $S \cup A$ is the unique extension to a maximal stable set of G^*.

By Lemma 22, for each C_5 of G^* there is a minimum weight coloring of G^* with $t = \max(w^*(v_2), w^*(v_4) + w^*(v_5))$ stable sets containing at least one vertex of $\{v_2, v_4, v_5\}$. Taking such a minimum weight coloring we can clearly remove vertices v_1 and v_3 from stable sets containing both until we obtain the smallest possible value of s in a partial weight coloring of C with $t = \max(w^*(v_2), w^*(v_4) + w^*(v_5))$. By Corollary 23, there is a partial weight coloring with $\chi_{w^*}(G[C])$ stable sets, such that $\max(w^*(v_2), w^*(v_4) + w^*(v_5))$ stable sets contain at least one of the vertices of $\{v_2, v_4, v_5\}$, and thus $s = \chi_{w^*}(G[C]) - t$ can be computed in constant time.

Now we are ready to present our coloring algorithm that computes a minimum weight coloring of (G^*, w^*) for a graph G^* of $\mathcal{C}_k$, $k \geq 1$. It removes at most k times the precomputed C_5 from the current graph until the remaining graph has no C_5 and is therefore a cochordal graph. Then an optimal weight coloring for the resulting cochordal graph can be computed by the linear time algorithm for the maximum weight clique problem presented in Section 6.

In each round, i.e., when removing a C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ from the current graph G' with current weight function w', the algorithm proceeds as follows: It computes in constant time an optimal partial weight coloring of C such that $t = \max(w'(v_2), w'(v_4) + w'(v_5))$ and s as small as possible. Then the algorithm removes all vertices of the C_5 and obtains the graph $G'' = G' - C$. Then it removes all vertices of A with weight at most t and decrements the weight of all other vertices in A by t, where A is the set of 0-vertices for C in G'. Recursively the algorithm solves the minimum weight coloring problem on the obtained graph G'' with weight function w''. Finally the minimum number of stable sets in a weighted coloring of (G', w') is obtained using the formula

$$\chi_{w'}(G') = t + \max(s, \chi_{w''}(G'')).$$

Thus the algorithm removes at most $k \leq n$ times a C_5. Each minimum partial weight coloring of the C_5 can be computed in constant time. For the final cochordal graph the minimum weight coloring can be solved in linear time. Thus the overall running time of the algorithm to compute a minimum weight coloring of a graph of class 3 is linear.

Summarizing we obtain

Theorem 24. *There is a linear time algorithm to solve the minimum coloring problem on (P_5,gem)-free graphs.*

8 Minimum Clique Cover

The Minimum Clique Cover problem is the following: Given a graph $G = (V, E)$ determine the smallest number of cliques of G in a collection of cliques of G covering its vertex set V. This number is denoted by $\kappa(G)$. Our algorithm to solve the minimum clique cover problem on (P_5,gem)-free graphs is similar to the algorithm of the previous section.

The 2-node subproblem is a weighted version of Minimum Clique Cover: Given a graph G^* and a vertex weight function $w^* : V \to \mathbb{N}$, find the smallest number of cliques in a weighted clique cover of (G^*, w^*), i.e., compute $\kappa_{w^*}(G^*)$.

Minimum Clique Cover is not LinEMSOL($\tau_{1,L}$) definable. We present a linear time algorithm to solve the Minimum Clique Cover problem on (P_5,gem)-free graphs.

Subproblems

For each node x of the modular decomposition tree T of G compute $\kappa(G(x))$ as follows:

If x is a **leaf** of T then $\kappa(G(x)) := 1$;
If x is a **0-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\kappa(G(x)) := \sum_{i=1}^{r} \kappa(G(x_i))$;
If x is a **1-node** of T and $x_1, x_2, \ldots x_r$ its children
then $\kappa(G(x)) := \max\limits_{i=1,2,\ldots,r} \kappa(G(x_i))$;
If x is a **2-node** of T, G^* the characteristic graph assigned to x and $x_1, \ldots x_r$ the children of x
then assign to the vertex set V^* of G^* the weight function $w^* : V^* \to \mathbb{N}$ such that $w^*(v_i) := \kappa(G(x_i))$. Then $\kappa(G(x)) := \kappa_{w^*}(G^*)$.

It is not hard to see that the overall computation on leaves, 0-nodes and 1-nodes can be done in linear time. The 2-node problem is a weighted clique cover problem for which the sum of the vertex weights of the graph G^* is at most the number of vertices of the (corresponding component of the) given (P_5,gem)-free graph G. Let N denote the sum of the weights of the prime graph G^* assigned to a 2-node.

Class 1: Matched cobipartite graphs

The graph G^* is cobipartite and thus perfect. By Lemma 21, $\kappa_{w^*}(G^*) = \alpha_{w^*}(G^*)$. A linear time algorithm to compute the maximum weight of a stable set in a matched cobipartite graph has been presented in Section 5. Using it we obtain a linear time algorithm to compute $\kappa_{w^*}(G^*) = \alpha_{w^*}(G^*)$.

Class 2: Specific graphs

In Section 7, we have seen that we can compute $\chi_w(G^*)$ in $O(1)$ time for graphs G^* of size bounded by a constant. As $\kappa_w(G^*) = \chi_w(\overline{G^*})$, we also can compute $\kappa_w(G^*)$ is $O(1)$ time for each graph whose size is bounded by a constant, so in particular also for specific graphs.

Class 3

Let G^* be a graph of class 3 with weight function w^*. First compute in linear time all vertices of degree two and all C_5's of G^* using the algorithm of Theorem 14.

If G^* is C_5-free, and thus $G \in \mathcal{C}_0$, then G^* is cochordal and $\kappa_{w^*}(G^*) = \alpha_{w^*}(G^*)$ can be computed using the linear time algorithm for the maximum weight stable set problem on cochordal graphs (see Section 5).

Otherwise $G^* \in \mathcal{C}_k$ and $k \geq 1$. If G^* is a C_5 then with the technique applied to specific graphs $\kappa_{w^*}(G^*)$ can be computed by an $O(1)$ algorithm.

Finally let $C = (v_1, v_2, v_3, v_4, v_5)$ be a C_5 of G^* and let v_1 and v_3 be its vertices of degree two. The set A of 0-vertices for C is a stable set, $B = V^* \setminus (C \cup A) = N(v_2) \setminus C = N(v_3) \setminus C = N(v_5) \setminus C$, and $G^*[B]$ is a cograph. Therefore all but one of the five maximal cliques of C, i.e., the five edges of C, are maximal cliques of G^*. The other maximal cliques of G^* containing vertices of C are all of the type $\{v_4, v_5\} \cup B'$ and $\{v_2\} \cup B'$ where B' is a maximal clique of the cograph $G^*[B]$. Finally there are maximal cliques $\{a\} \cup A'$ of G^*, where A' is a maximal clique of $G^*[N(a)]$ and $a \in A$.

Lemma 25. *Let $k \geq 1$, $G^* \in \mathcal{C}_k$. Let $C = (v_1, v_2, v_3, v_4, v_5)$ be a C_5 of G^* such that vertices v_1 and v_3 have degree two. Let w^* be the vertex weight function of G^*. Then there is a minimum weight clique cover of G^* with precisely $w^*(v_1) + w^*(v_3)$ cliques containing either v_1 or v_3.*

Proof. Consider a minimum weight clique cover $\mathcal{K}$ of G^*. Clearly any clique containing v_1 or v_3 is a subset of C.

Suppose $\mathcal{K}$ contains more than $w^*(v_1)+w^*(v_3)$ cliques containing v_1 or v_3. Then we obtain a minimum weight clique cover $\mathcal{K}'$ with precisely $w^*(v_1) + w^*(v_3)$ cliques containing v_1 and v_3 by applying the following replacement operations. Let $c(v)$ be the number of cliques of $\mathcal{K}$ containing v. Then replace $c(v_1)-w^*(v_1)$ of those cliques K by $K \setminus \{v_1\}$. Similarly replace $c(v_3) - w^*(v_3)$ cliques of $\mathcal{K}$ containing v_3. □

Corollary 26. *Let G be a C_5 with vertices $(v_1, v_2, v_3, v_4, v_5)$, and w be the vertex weight function of G. Then there is a minimum weight clique cover of G with precisely $w(v_1) + w(v_3)$ cliques containing either v_1 or v_3.*

Thus we study the weighted clique covers of a C_5 C with vertex weight function w^*. We call any such weighted clique cover of the C_5 a partial weight clique cover of C. To extend any partial weight clique cover of C to G^* only two parameters are important:

- the number of cliques of the partial weight clique cover of C containing either v_1 or v_3, denoted by s, and
- the number of cliques $\{v_4, v_5\}$, $\{v_4\}$, $\{v_5\}$ and $\{v_2\}$ of the partial weight clique cover of C, denoted by t.

To extend a partial weight clique cover of the C_5 $C = (v_1, v_2, v_3, v_4, v_5)$ to a minimum weight clique cover of G^* we have to extend the t cliques $\{v_4, v_5\}$, $\{v_4\}$, $\{v_5\}$ or $\{v_2\}$ of the C_5 by adding to each of these t cliques a clique B' of $G^*[B]$. Furthermore we may assume that each vertex $a \in A$ appears in $w^*(a)$ cliques of $\mathcal{K}$. (Note that none of the latter cliques contains a vertex of C.)

The algorithm to compute a minimum weight clique cover of (G^*, w^*) for a graph G^* of $\mathcal{C}_k$, $k \geq 1$, removes k times the precomputed C_5 from the current graph until the remaining graph has no C_5 and is therefore a cochordal graph. Given a C_5 C to be removed an optimal partial weight clique cover of C with $s = w^*(v_1) + w^*(v_3)$ has as few as possible cliques $\{v_4, v_5\}$, $\{v_4\}$, $\{v_5\}$ and $\{v_2\}$. By Corollary 26, $t = \kappa_{w^*}(G[C]) - s$, can be computed in constant time.

The algorithm solves the minimum weight clique cover problem recursively. Let G' be the current graph with weight function w'. Then the algorithm computes the smallest possible value of t in a partial weight cover of C with weight function w' and $s = w'(v_1)+w'(v_3)$. Recursively the algorithm computes $\kappa_{w''}(G'-C)$, where $w''(v) = w'(v)$ for all vertices of $G'-C$. Then the minimum number of cliques in a clique cover of (G', w') is obtained by the formula

$$\kappa_{w'}(G') = s + \sum_{a \in A} w'(a) + \max(t, \kappa_{w''}(G' - C) - \sum_{a \in A} w'(a)),$$

where A is the set of 0-vertices for C in G'. Note that $\sum_{a \in A} w'(a)$ cliques of the minimum weight clique cover of $G' - C$ contain a vertex of A and cannot be extended by adding a vertex of C. Thus only $\kappa_{w''}(G' - C) - \sum_{a \in A} w'(a)$ cliques can be extended to cliques of G' by adding $\{v_4, v_5\}$, $\{v_4\}$, $\{v_5\}$ or $\{v_2\}$.

Thus our recursive algorithm removes $k \leq n$ times a C_5 and computes a minimum weight clique cover for a cochordal graph. The minimum weight of a clique cover of the remaining cochordal graph can be computed in linear time. Consequently the overall running time for the minimum weight clique cover problem on graphs of class 3 is linear.

Summarizing we obtain

Theorem 27. *There is a linear time algorithm to solve the minimum clique cover problem on (P_5,gem)-free graphs.*

Acknowledgement

Part of the research was done while A. Brandstädt and J. Spinrad were visiting the University of Metz. Thanks are due to Alexander Schrijver for pointing towards Theorem 17.2 from his book [31].

References

[1] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms* 12 (1991), 308-340

[2] H. Bodlaender, Achromatic number is NP-complete for cographs and interval graphs, *Inform. Process. Lett.* 31 (1989) 135-138

[3] H.L. Bodlaender, H.J. Broersma, F.V. Fomin, A.V. Pyatkin, G.J. Woeginger, Radio labeling with pre-assigned frequencies, *Proceedings of the 10th European Symposium on Algorithms (ESA'2002)*, LNCS 2461 (2002) 211-222

[4] H.L. Bodlaender, K. Jansen, On the complexity of the maximum cut problem, *Nord. J. Comput.* 7 (2000) 14-31

[5] H.L. Bodlaender, U. Rotics, Computing the treewidth and the minimum fill-in with the modular decomposition, *Algorithmica* 36 (2003) 375-408

[6] A. Brandstädt, D. Kratsch, On the structure of (P_5,gem)-free graphs, *Manuscript* 2002; to appear in *Discrete Applied Math.*

[7] A. Brandstädt, H.-O. Le, R. Mosca, Chordal co-gem-free graphs have bounded clique width, *Manuscript* 2002; to appear in *Discrete Applied Math.*

[8] A. Brandstädt, V.B. Le, J. Spinrad, Graph Classes: A Survey, *SIAM Monographs on Discrete Math. Appl., Vol.* 3, SIAM, Philadelphia (1999)

[9] M. Chudnovsky, N. Robertson, P.D.Seymour, R.Thomas, The Strong Perfect Graph Theorem, *Manuscript* 2002

[10] D.G. Corneil, H. Lerchs, L. Stewart-Burlingham, Complement reducible graphs, *Discrete Applied Math.* 3 (1981) 163-174

[11] D.G. Corneil, Y. Perl, L.K. Stewart, Cographs: recognition, applications, and algorithms, *Congressus Numer.* 43 (1984) 249-258

[12] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Computing* 14 (1985) 926-934

[13] B. Courcelle, J.A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, *Theory of Computing Systems* 33 (2000) 125-150

[14] A. Cournier, M. Habib, A new linear algorithm for modular decomposition, *Trees in Algebra and Programming - CAAP '94*, LNCS 787 (1994) 68-84

[15] E. Dahlhaus, J. Gustedt, R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* 41 (2001) 360-387

[16] W. Espelage, F. Gurski, E. Wanke, How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time, *Proceedings of the 27th Workshop on Graph-Theoretic Concepts in Computer Science (WG 2001)*, LNCS 2204 (2001) 117-128

[17] A. Frank, Some polynomial algorithms for certain graphs and hypergraphs, *Proceedings of the Fifth British Combinatorial Conference (Univ. Aberdeen, Aberdeen, 1975)* 211-226, *Congressus Numerantium* No. XV, Utilitas Math., Winnipeg, Man. (1976)

[18] T. Gallai, Transitiv orientierbare Graphen, *Acta Mathematica Academiae Scientiarum Hungaricae* 18 (1967) 25-66

[19] V. Giakoumakis, I. Rusu, Weighted parameters in ($P5$, $\overline{P5}$)-free graphs, *Discrete Appl. Math.* 80 (1997) 255-261

[20] M.C. Golumbic, Trivially perfect graphs, *Discrete Math.* 24 (1978) 105-107

[21] M.C. GOLUMBIC, Algorithmic Graph Theory and Perfect Graphs, *Academic Press, New York* (1980)

[22] K. JANSEN, P. SCHEFFLER, Generalized coloring for tree-like graphs, *Discrete Appl. Math.* 75 (1997) 135-155

[23] D. KOBLER, U. ROTICS, Edge dominating set and colorings on graphs with fixed clique-width, *Discrete Appl. Math.* 126 (2003) 197-221

[24] L. LOVÁSZ, Normal hypergraphs and the perfect graph conjecture, *Discrete Math.* 2 (1972) 253-267

[25] F. MAFFRAY, M. PREISSMANN, A translation of Gallai's paper: 'Transiv orientierbare Graphen', in *Perfect graphs*, J.L. Ramirez Alfonsin, B.A. Reed, (eds.), Wiley (2001) 25-66

[26] R.M. MCCONNELL, J. SPINRAD, Modular decomposition and transitive orientation, *Discrete Math.* 201 (1999) 189-241

[27] R.H MÖHRING, F.J. RADERMACHER, Substitution Decomposition for Discrete Structures and Connections with Combinatorial Optimization, Algebraic and Combinatorial Methods in Operations Research, 257-355, 1984

[28] M. RAO, Décomposition modulaire de graphes et problèmes NP-complets, Rapport de DEA, Université de Metz (2002)

[29] C. MCDIARMID, B. A. REED, Channel assignment and weighted coloring, *Networks* 36 (2000) 114-117

[30] D. ROSE, R. TARJAN, G. LUEKER, Algorithmic aspects of vertex elimination on graphs, *SIAM J. Computing* 5 (1976) 266-283

[31] A. SCHRIJVER, *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester (1986)

[32] E. WANKE, k-NLC graphs and polynomial algorithms, *Discrete Appl. Math.* 54 (1994) 251-266

[33] E.S. WOLK, The comparability graph of a tree, *Proc. Amer. Math. Soc.* 13 (1962) 789-795

[34] E.S. WOLK, A note on "The comparability graph of a tree", *Proc. Amer. Math. Soc.* 16 (1965) 17-20