RAIRO. RECHERCHE OPÉRATIONNELLE

KLAUS JANSEN
PETRA SCHEFFLER
GERHARD WOEGINGER

The disjoint cliques problem

RAIRO. Recherche opérationnelle, tome 31, nº 1 (1997), p. 45-66
<http://www.numdam.org/item?id=RO_1997__31_1_45_0>

Recherche opérationnelle/Operations Research
(vol. 31, n° 1, 1997, pp. 45-66)

THE DISJOINT CLIQUES PROBLEM (*)

by Klaus JANSEN (1), Petra SCHEFFLER (2) and Gerhard WOEGINGER (3)

Communicated by Philippe CHRÉTIENNE

Abstract. – *Given a graph $G = (V, E)$, we consider the problem of finding a set of D pairwise disjoint cliques in the graph with maximum overall number of vertices. We determine the computational complexity of this problem restricted to a variety of different graph classes. We give polynomial time algorithms for the problem restricted to interval graphs, cographs, directed path graphs and partial k-trees. In contrast, we show the NP-completeness of this problem for undirected path graphs.*

Moreover, we investigate a closely related scheduling problem. Given D times units, we look for a sequence of workers $w_1, \cdots, w_k$ and a partition $J_1, \cdots, J_k$ of the job set such that J_i can be executed by w_i within D time units. The goal is to find a sequence with minimum total wage of the workers.

Keywords: Disjoint cliques, interval graph, cograph, directed path graph, partial k-trees, computational complexity, scheduling problem.

Résumé. – *Étant donné un graphe $G = (V, E)$, nous considérons le problème consistant à trouver dans ce graphe un ensemble de D cliques deux à deux disjointes ayant un nombre total maximum de sommets. Nous déterminons la complexité de ce problème lorsqu'il est restreint à diverses classes de graphes. Nous donnons des algorithmes en temps polynomial pour le problème restreint aux graphes d'intervalle, aux cographes, aux graphes de chemins orientés, et aux k-arbre partiels. Par contre, nous montrons que ce problème, appliqué aux graphes non-orientés, est NP-complet.*

En outre, nous examinons un problème d'ordonnancement fortement associé. Étant données D unités de temps, nous cherchons une suite $w_1, \cdots, w_k$ d'ouvriers et une partition $J_1, \cdots, J_k$ de l'ensemble des travaux tels que J_i puisse être exécuté par w_i en D unités de temps au plus. L'objectif est de trouver une suite minimisant le salaire total des ouvriers.

Mots clés : Cliques disjointes, graphe d'intervalles, cographe, graphe à chemin orienté, k-arbre partiel, complexité, problème d'ordonnancement.

1. INTRODUCTION

Let J be a set of unit-time jobs and let G be a compatibility graph on J. Two adjacent jobs in G are compatible and may be performed at the same

---

(*) Received November 1993.

(1) Institut für Informatik, Technische Universität München, 80 290 München, Germany – on leave from Universität Trier, Germany.

(2) FB Mathematik, MA 6-1, TU Berlin, Strasse des 17. Juni 136, 10623 Berlin, Germany.

(3) Institut für Informationsverarbeitung, TU Graz, Klosterwiesgasse 32, 8010 Graz, Austria.

time. Given a time period of D time units, we look for a feasible sequence of workers $w_1, \cdots, w_k$ together with a partition of the job set J into k sets $J_1, \cdots, J_k$ such that each job set J_i can be executed by a worker w_i within D time units. The cost of a sequence is the overall wage of all workers. The goal is to find a feasible sequence with minimum cost.

In Section 2, we propose an approximation algorithm for this scheduling problem with worst case ratio $O(\log |J|)$. The algorithm uses the computation of a maximum set of jobs executable by one worker within D time steps. This problem can be described graph theoretical as follows:

INSTANCE: A finite undirected graph $G = (V, E)$ and two positive integers $D, B \leq |V|$.

QUESTION: Do there exit D pairwise disjoint cliques $C_1, \cdots, D_D$ in G such that $\sum_{i=1}^{D} |C_i| \geq B$ holds?

A set $C \subseteq V$ is a clique in a graph $G = (V, E)$ if each pair $v, v' \in C$ of vertices with $v \neq v'$ is connected by an edge $\{v, v'\} \in E$. We call the problem DISJOINT UNION OF CLIQUES (DUC for short). In this paper, we analyse the computational complexity of DUC for several graph classes. We obtain a polynomial time approximation algorithm for the scheduling problem restricted to graph classes on which DUC is polynomial time solvable.

For arbitrary undirected graphs, DUC is easily seen to be the NP-complete: For $D = 1$, it turns into the well-known CLIQUE problem, and for $B = |V|$, it becomes the PARTITION INTO CLIQUES problem (*cf.* Garey and Johnson [GJ] for more information on these two problems). Both problems (CLIQUE and PARTITION INTO CLIQUES) are polynomially solvable for chordal graphs. This yields to the natural question whether DUC is also polynomially solvable when restricted to chordal graphs.

The NP-completeness of DUC restricted to split graphs, a subclass of the chordal graphs has been proved by Yannakakis and Gavril [YG]. But for interval graphs (Section 3) and for directed path graphs (Section 4) we give algorithms with time complexity $O(D \cdot |V|^2)$ and $O(D^2 \cdot |V|^2)$, respectively. Furthermore, we show the NP-completeness for undirected path graphs (Section 5)–another subclass of chordal graphs. Moreover, we study some other important graph classes for that we found polynomial time algorithms. These are the cographs (Section 6) and partial k-tress (Section 7) with $O(|V|^2)$ and $O(D^2 \cdot |V|)$ as time complexity, respectively.

The problem Duc was analysed first by Frank [Fr]. He considered comparability graphs and their complement graphs (co-comparability graphs) and gave an algorithm for both graph classes with time complexity $O(a \cdot b \cdot |V|^2)$ where a is the cardinality of a maximum clique and where b is the cardinality of a maximum independent set. Gavril [Ga] proposed a slightly better algorithm which needs $O(D \cdot |V|^2)$ time steps for comparability graphs and $O(|V|^3 + b|V|^2 \log(|V|))$ for co-comparability graphs. For subclasses like the interval graphs (Section 3) and cographs (Section 6) we found algorithms with a better time complexity. We notice that Duc can be solved in $O(\sqrt{|V|}|E|)$ time for bipartite graphs using a matching algorithm of Micali and Vazirani [MV]. The exact definitions of the graph classes are given in the corresponding sections.

Notation: We consider the optimization version of Duc in all those parts of our paper where we give algorithms. That means, we describe how to find the maximum number $\omega_D(D)$ of vertices in D disjoint cliques given a graph G and a positive integer D. Given a graph G and a set $H \subset V$ we denote by $G|_H$ the subgraph of G induced by H.

2. APPLICATION TO A SCHEDULING PROBLEM

The problem Duc is closely related to a scheduling problem defined as follows. Let J be a set of unit-time jobs and let $G = (J, E)$ be a compatibility graph on J. If two jobs are adjacent in G, they are *compatible* with each other and may be performed at the same time. The kind of jobs we consider are simple supervision and control jobs as supervising the operation of machines. Thus, these jobs are highly parallelizable and one person may perform two or more of these jobs at the same time. If a worker performs several jobs at the same time, he needs only one unit time to complete them all.

A worker w is described by his wage $c(w) \in \mathbb{N}$ and by a set of jobs $J(w) \subseteq J$ he is able to fulfill. We assume that there are several *types* of workers collected in a set W. Workers of the same type have similar education, similar knowledge and similar abilities, and hence they are able to perform the same jobs and they earn the same wages. Moreover, we assume that for each type there is an arbitrarily large number of workers available. A worker w can execute a job set J' within D unit time intervals if $J' \subseteq J(w)$ and if J' can be partitioned into D subsets of pairwise compatible jobs. (The intuition is that during each time interval, the worker has to supervise several compatible jobs *in parallel*).

We are looking for a schedule of the jobs to an appropriate subset of the workers. The main goal is to keep the overall money paid to the workers as small as possible while completing all jobs. Given a time period D, a *feasible schedule* S with respect to D consists of a sequence of workers $w_1, \cdots, w_k$ together with a partition of the jobs in J into k sets $J_1, \cdots, J_k$ such that job set J_i can be executed by worker w_i within time D. The *cost* of a feasible schedule is defined to be $\sum_{i=1}^{k} c(w_i)$, the overall wage of all employed workers. An *optimum* schedule is a feasible schedule with minimum cost.

It is straightforward to see that a set $J' \subset J(w)$ of jobs can be executed by a worker w within D time intervals if and only if the subgraph of G induced by J' is the union of a most D cliques. Thus, the problem DUC can be considered being a special case of our scheduling problem (to find the maximum number of jobs for one worker). On the other hand, we propose the following approximation algorithm for the scheduling problem.

1. Compute for each type of worker $w \in W$ the graph G_w which is the vertex-induced subgraph of G induced by the set $J(w)$.

2. Compute for each graph G_w the size $\omega_D(G_w)$.

3. Choose a type of worker w' that maximizes the quotient $\omega_D(G_w)/c(w)$.

4. Compute the corresponding set $H(w')$ of jobs in $G_{w'}$ which generates $\omega_D(G_{w'})$ and remove these jobs from each of the graphs G_w. Iterate the algorithm until all jobs are covered.

The main step in the algorithm is Step 2, computing for the induced compatibility graph G_w the maximum size of D cliques. Using the same proof technique as in [Ja], we get the following theorem.

THEOREM 2.1: *Let $\mathcal{G}$ be a graph class that is closed under the induced subgraph operation and on that* DUC *is solvable in polynomial time. Consider a scheduling problem with compatibility graph $G \in \mathcal{G}$. Then, the above approximation algorithm constructs a schedule whose cost is at most a factor $O(\log |J|)$ away from the optimum cost.*

3. INTERVAL GRAPHS

In this section we give an algorithm for DUC restricted to interval graphs with time complexity $O(D \cdot |V|^2$. Since each interval graph is also a co-comparability graph, we improve the algorithm of Gavril with time complexity $O(|V|^3 + b|V|^2 \log(|V|))$. A graph $G = (V, E)$ is an *interval graph* if one can associate with each vertex $v \in V$ a closed interval I_v on the real line such that two vertices $u, v \in V$ are adjacent in G if and only if $I_u \cap I_v \neq \emptyset$. More precisely, an interval graph can be described as a sequence of all its maximal cliques $A_1, \cdots, A_n$ such that each vertex v only occurs in consecutive cliques. An example of an interval graph is given in Figure 1. We note that the number n is bounded by $|V|$. A consecutive arrangement of the maximal cliques can be obtained in $O(|V| + |E|)$ time [BL].

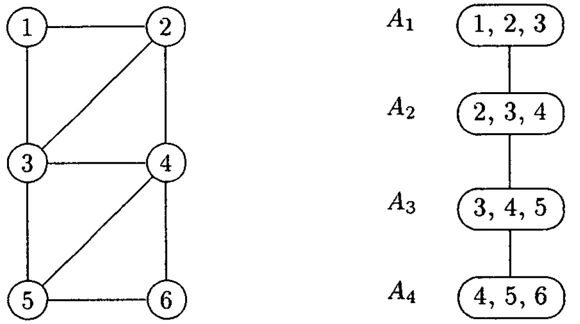

Figure 1. – An interval graph and its consecutive clique arrangement.

We denote by $\omega_{k,i}(G)$ the maximal size of a most k disjoint cliques in the interval graph $G|_{A_1 \cup \cdots \cup A_i}$ induced by the first i cliques. These values satify the following recursive relation.

LEMMA 3.1

$$\omega_{k,i}(G) = \begin{cases} \max\{\omega_{k,i-1}(G),\ \max_{1 \le j \le i-1} \omega_{k-1,j}(G) + |A_i \backslash A_j|\} & \\ \qquad\qquad\qquad\qquad \text{if} \quad i,\ k > 1 & \\ \max_{1 \le j \le i} |A_j| \qquad \text{if} \quad i > 1,\ k = 1 & \\ |A_1| \qquad i = 1, \qquad \text{if} \quad k \ge 1 & \end{cases}$$

Proof: The statement directly follows from the definition of the values $w_{k,i}(G)$ and uses the consecutivity property of the maximal cliques $A_1, \cdots, A_n$. □

The optimal value $\omega_D(G)$ is equal to $\omega_{D,n}(G)$. In the computation of $\omega_D(G)$ we have to compute $O(D \cdot |V|)$ values $\omega_{k,i}$. For our example in Figure 1, we obtain the values $\omega_{k,i}(G)$ for $1 \le k \le D = 2$ and $1 \le i \le 4$ that are shown in Table I. The optimal value $\omega_2(G)$ is equal to $\omega_{2,4}(G) = 6$.

TABLE I

The values $\omega_{k,i}(G)$.

$k \setminus i$	1	2	3	4
1	3	3	3	3
2	3	4	5	6

In the following we consider the computation of the set differences.

LEMMA 3.2: *The cardinalities of the set differences* $|A_i \backslash A_j|$ *for* $1 \leq j < i \leq n$ *can be computed in* $O(n^2)$ *time.*

Proof: For each pair j, i with $j < i$ we have to compute

$$|A_i \backslash A_j| = |A_i| - |A_i \cap A_j|.$$

Let us denote $a_{j,i} = |A_i \cap A_j|$. From the consecutive property of the cliques we know that

$$a_{j,i} = |\{v \in V | \text{for all } k,\ j \leq k \leq i : v \in A_k\}|.$$

The value $a_{j,i}$ gives the number of intervals starting before j and ending after i. To compute these values we use numbers $b_{j,i} = |\{v \in V|$ for all k, $j \leq k \leq i$: $v \in A_k \wedge v \notin A_{j-1}\}|$. A number $b_{j,i}$ gives the number of intervals starting at j and ending after i. Moreover, we generate values $c_{j,i} = |\{v \in V|$ for all k, $j \leq k \leq i$: $v \in A_k \wedge v \notin A_{j-1} \wedge v \notin A_{i+1}\}|$ where $c_{j,i}$ denotes the number of intervals starting at j and ending at i. The values $c_{j,i}$ can be computed directly from the sequence $A_1, \cdots, A_n$ in $O(n^2)$ time. Then, using the recursion

$$a_{j,i} = \begin{cases} a_{j-1,i} + b_{j,i} & \text{for } 1 < j \\ b_{1,i} & \text{for } 1 = j \end{cases}$$

and the recursion

$$b_{j,i} = \begin{cases} b_{j,i+1} + c_{j,i} & \text{for } i < n \\ c_{j,n} & \text{for } i = n \end{cases}$$

all values $b_{j,i}$ and $a_{j,i}$ with $1 \leq j < i \leq n$ are computed in $O(n^2)$ time. Therefore, the cardinalities of all set differences $A_i \backslash A_j$ can be generated in $O(n^2)$ time. □

Summarizing, we derived the following result.

THEOREM 3.3: *For an interval graph* $G = (V, E)$, $\omega_D(G)$ *can be computed in time* $O(D \cdot |V|^2)$.

Proof: For each value $\omega_{k,i}(G)$ with $1 \leq k \leq D$, $1 \leq i \leq |V|$ we need at most $O(|V|)$ comparisons. □

4. DIRECTED PATH GRAPHS

In this section, we consider the directed path graphs, a generalization of the interval graphs. We give a polynomial algorithm with time complexity $O(D^2 \cdot |V|^2)$ for these graphs.

A graph $G = (V, E)$ is a *directed path graph,* if it is the intersection graph of directed paths in a directed tree. That means, that there is a directed tree $T = (I, F)$ with all arcs oriented from its root to the leaves and for every vertex $v \in V$ there is a directed path P_v in T, such that for all pairs of vertices $u, v \in V$ with $(u \neq v)$ there is an edge $\{u, v\} \in E$, if and only if P_u and P_v have at least one node in common. Such a representation of a directed path graph can be obtained in $O(|V| + |E|)$ time [Di] and the number of nodes in such a tree can be bounded by $O(|V|)$. An example of an directed path graph with its corresponding directed tree model is given in Figure 2. Notice, that this graph is not an interval graph.

We denote by T_x the subtree of T rooted at node x, by H_x the set of vertices v whose paths P_v go throught x and by G_x the subgraph of G induced by those vertices V_x that correspond to path in T_x.

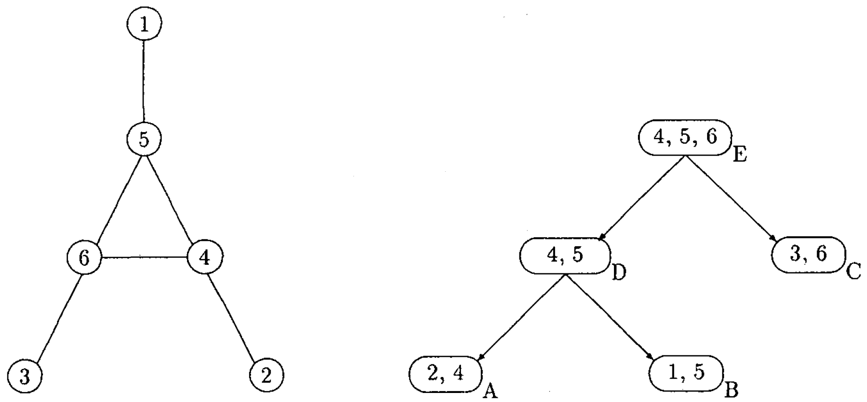

Figure 2. – A direct path graph and its corresponding directed tree.

For simplification we assume here that the tree is given as a binary tree. However, each tree which represents a directed path graph can be transformed into an equivalent binary tree. For a transformation of a general tree into a binary tree we refer to Figure 3. It shows a transformation of a node of out-degree k into a tree with only nodes of out-degree two. In the following, we show how the maximum numbers $\omega_d(G_x)$ of vertices in d disjoint cliques in G_x can be computed.

LEMMA 4.1: *Let x be a node in the tree T with two children $l(x)$ and $r(x)$. Then, $\omega_d(G_x)$ with $d \in \{1, \cdots, D\}$ can be computed by the maximum of the following two values:*

(1) $\max_{0 \leq i \leq d}[\omega_i(G_{l(x)}) + \omega_{d-i}(G_{r(x)})]$,

(2) $\max_{0\leq i\leq d-1}[\omega_i(G_{l(x)}|_{H_x^c}) + \omega_{d-i-1}(G_{r(x)}|_{H_x^c}) + |H_x|]$,
where H_x^c denotes the complement set $V_x \backslash H_x$ and $G_{l(x)}|_{H_x^c}$ is the subgraph of G induced by vertices that correspond to paths in $T_{l(x)}$ not going through the node x and $G_{r(x)}|_{H_x^c}$ is defined in $T_{r(x)}$ analogously.

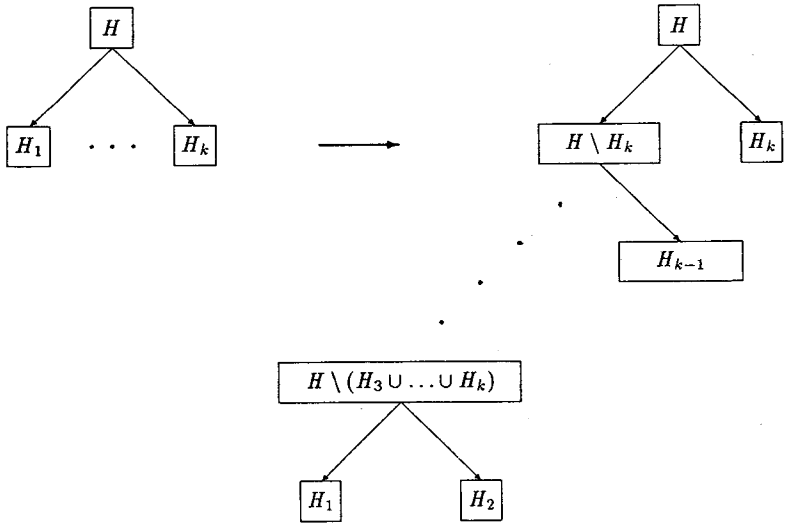

Figure 3. – Transformation of a node with out-degree k.

Proof: Note, that each set H_x is a clique in G. Let $C_1, \cdots, C_d$ be a set of d cliques in the graph G_x. For each clique C there exists a node y in T_x with $C \subset H_y$. Therefore, for each $i \in \{1,, \cdots, d\}$, if $C_i \not\subset H_x$ then $C_i \subset V_{l(x)}$ or $C_i \subset V_{r(x)}$. If $C_i \not\subset H_x$ for each $i \in \{1,, \cdots, d\}$ then the d cliques can be divided into i cliques for the left and $d-i$ cliques for the right subgraph. In this case $\omega_d(G_x)$ is equal to $\omega_i(G_{l(x)}) + \omega_{d-i}(G_{r(x)})$ with $0 \leq i \leq d$.

If $C_i \subset H_x$ for at least one $i \in \{1,, \cdots, d\}$ then another set of at most d disjoint cliques may be defined as:

(1) $C_i' = H_x$,

(2) $\forall j \in \{1,, \cdots, d\} - \{i\}$: $C_j' = C_j - H_x$.

Clearly, $\sum_{j=1}^{d} |C_j'| \geq \sum_{j=1}^{d} |C_j|$. Then, we may assume that the other $d-1$ cliques lie in the left or right subgraph. In this case, we must delete the vertices of H_x in $G_{l(x)}$ and $G_{r(x)}$ and obtain $\omega_d(G_x) = |H_x| + \omega_i(G_{l(x)}|_{H_x^c}) + \omega_{d-i-1}(G_{r(x)}|_{H_x^c})$. □

The main idea in computing the value $\omega_D(G)$ is to generate for each subtree T_x and for each predecessor y of x lying on the path from the root w to x in T the values $\omega_d(G_x|_{H_y^c})$ and the values $\omega_d(G_x)$ with $d \in \{1, \cdots, D\}$.

TABLE II
The values $\omega_i(G_x)$.

$i \setminus x$	A	B	C	D	E
0	0	0	0	0	0
1	2	2	2	2	3
2	2	2	2	4	4

The values $\omega_d(G_x|_{H_y^c})$ are computed bottom up in the tree; first we compute the values $\omega_d(G_{l(x)}|_{H_y^c})$, $\omega_d(G_{r(x)}|_{H_y^c})$ for the left and right children and then the value $\omega_d(G_x|_{H_y^c})$ for x. For a leaf we get for $d > 0$ the value $\omega_D(G_x|_{H_y^c}) = |H_x \backslash H_y|$ and for $d = 0$ the value zero. Using the fact that the graph $G_x|_{H_x^c}$ is a subgraph of $G_x|_{H_y^c}$ we take for $\omega_d(G_x|_{H_y^c})$ now the maximum of the following values:

(1) $\max_{0 \leq i \leq d}[\omega_i(G_{l(x)}|_{H_y^c}) + \omega_{d-i}(G_{r(x)}|_{H_y^c})]$,

(2) $\max_{0 \leq i \leq d-1}[\omega_i(G_{l(x)}|_{H_x^c}) + \omega_{d-i-1}(G_{r(x)}|_{H_x^c}) + |H_x \backslash H_y|]$,

In the following we consider the computation of the cardinalities of the set differences $|H_x \backslash H_y|$. In Table II we give for the example of Figure 2 the computed values $\omega_i(G_x)$ for $0 \leq i \leq 2$ and $x \in \{A, B, C, D, E\}$.

LEMMA 4.2: *The cardinalities of the set differences $|H_x \backslash H_y|$ can be computed in $O(|I|^2)$ time for all pairs of nodes x and y, where y is a predecessor of x in the tree $T = (I, F)$.*

Proof: Let us denote $a_{x,y} = |H_x \cap H_y|$. To compute these values we use numbers $c_{x,y}$ and $b_{x,y}$. The value $c_{x,y}$ gives the number of paths P_v starting at y and ending at x and the value $b_{x,y}$ gives the number of paths P_v starting above to y and ending at x. The values $c_{x,y}$ can be computed directly from the tree representation in $O(|I|^2)$ time. The computation of the values $b_{x,y}$ can be done using the following recursion also in quadratic time. Let $p(y)$ be the direct predecessor of a vertex y in the tree with y unequal to the root w.

$$b_{x,y} = \begin{cases} b_{x,p(y)} + c_{x,y} & \text{if } y \neq w \\ c_{x,y} & \text{if } y = w \end{cases}$$

Moreover, the values $a_{x,y}$ can be generated using the recursion:

$$a_{x,y} = \begin{cases} a_{r(x),y} + a_{l(x),y} + b_{x,y} & \text{if } x \text{ is not a leaf,} \\ b_{x,y} & \text{if } x \text{ is a leaf.} \end{cases}$$

Therefore, the sizes of set differences $|H_x \backslash H_y| = |H_x| - |H_x \cap H_y|$ can be generated in $O(|I|^2)$ time. □

THEOREM 4.3: *For a directed path graph* $G = (V, E)$, *the problem* DUC *is solvable in time* $O(D^2 \cdot |V|^2)$.

Proof: Use that at most $O(D \cdot |V|^2)$ values must be computed and that the composition can be done in $O(D)$ time for each recursion step. The sizes of the set differences $|H_x \backslash H_y|$ can be computed by preprocessing in $O(|V|^2)$ time. Therefore, the complete algorithm needs $O(D^2 \cdot |V|^2)$ time steps. □

5. UNDIRECTED PATH GRAPHS

An *undirected path graph* is a generalization of a direct path graph introduced in the preceding section. Undirected path graphs are the intersection graphs of undirected paths in an unrooted and undirected tree. In Figure 4 we give an example of an undirected path graph which is not a directed path graph.

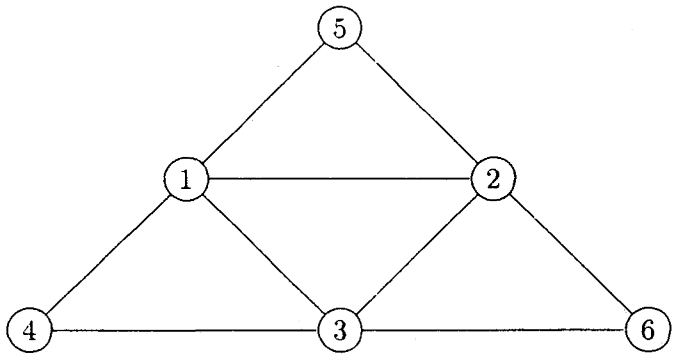

Figure 4. – An undirected path graph which is not a directed path graph.

The class of undirected path graphs, directed path graphs and split graphs all are subclasses of the *chordal* graphs. Hence, the NP-completeness result of DUC for split graphs (*see* [YG]) implies that DUC is NP-complete for chordal graphs. Whereas for directed path graphs there exists a polynomial time algorithm, we show in this section that DUC is NP-complete for undirected path graphs.

THEOREM 5.1: *The problem* DUC *is NP-complete for undirected path graphs.*

Proof: We prove this by reduction from the NP-complete 3-SAT problem. We use the restricted version where each literal occurs at most three times in the clauses (*cf.* [GJ]). Assume that an instance of 3-SAT is given. Let

$X = \{\overline{x_1}, \cdots, x_n, \overline{x_n}\}$ be its set of variables and $\{c_1, \cdots, c_m\}$ be its set of clauses of size three. We construct now an undirected path graph that has $n + m$ cliques which cover $(n + m)(6n + 3)$ vertices if and only if there is a truth assignment that verifies the given formula.

For each variable x_i we define six vertices $x_i^{(1)}, \overline{x_i^{(1)}}, \cdots, x_i^{(3)}, \overline{x_i^{(3)}}$. That means, that we have one vertex for every occurrence of a literal in the formula. We denote by $Y_j = \{y_{j,1}, y_{j,2}, y_{j,3}\}$ the set of the vertices corresponding to the variables in the j-th clause c_j with $y_{j,k} = x_i^{(l)}$ or $y_{j,k} = \overline{x_i^{(l)}}$. Here l stands for the number of the occurrence of x_i rsp. $\overline{x_i}$, so $l \in \{1, 2, 3\}$. Using this setting, we define now some additional sets of vertices that we need in order to get cliques of equal size in our graph:

- for each $i \in \{1,, \cdots, n\}$ we take one set L_i of size n and two sets $K_i^{(1)}$ and $K_i^{(2)}$, each of size $5n$.
- for each $j \in \{1,, \cdots, m\}$ we take three additional vertices $c_j^{(1)}$, $c_j^{(2)}$ and $c_j^{(3)}$ and three sets $D_j^{(1)}$, $D_j^{(2)}$ and $D_j^{(3)}$, each of size $6n$.

We denote $A_i = \{x_i^{(1)}, x_i^{(2)}, x_i^{(3)}\}$, $B_i = \{\overline{x_i^{(1)}}, \overline{x_i^{(2)}}, \overline{x_i^{(3)}}\}$ and $X_i = A_i \cup B_i$ for each $i \in \{1,, \cdots, n\}$. Furthermore, we set $X' = \bigcup_{i=1}^{n} X_i$.

We now define the input graph $G = (V, E)$ for the DUC problem. Its vertices set is the union

$$V = X' \cup \bigcup_{i=1}^{n} (L_i \cup K_i^{(1)} \cup K_i^{(2)})$$
$$\cup \bigcup_{j=1}^{m} (D_j^{(1)} \cup D_j^{(2)} \cup D_j^{(3)} \cup \{c_j^{(1)}, c_j^{(2)}, c_j^{(3)}\}).$$

We take an edge between a pair of vertices if and only if one of the following sets contains both vertices. Each of the following sets forms a clique in G.

- the set X'.
- for each $i \in \{1,, \cdots, n\}$:
  - $X_i \cup L_i$.
  - $A_i \cup L_i \cup K_i^{(1)}$.
  - $B_i \cup L_i \cup K_i^{(2)}$.
- for each $j \in \{1,, \cdots, m\}$:
  - $Y_j \cup \{c_j^{(1)}, c_j^{(2)}, c_j^{(3)}\}$.
  - $\{y_{j,1}\} \cup \{c_j^{(2)}, c_j^{(3)}\} \cup D_j^{(1)}$.

– $\{y_{j,2}\} \cup \{c_j^{(1)},\ c_j^{(3)}\} \cup D_j^{(2)}$.

– $\{y_{j,3}\} \cup \{c_j^{(1)},\ c_j^{(2)}\} \cup D_j^{(3)}$.

First, we note that the graph G, given in this way is an undirected path graph. This follows from the fact that we can arrange the cliques in a tree T, such that each vertex in G lies in cliques of an undirected path in T. A possible arrangement of the cliques in a tree is illustrated in Figure 5.

Now we prove the equivalence that G contains $D = n + m$ cliques of size $B = (6n + 3)\cdot(n + m)$ if and only if there is a truth assignment that verifies all m clauses.

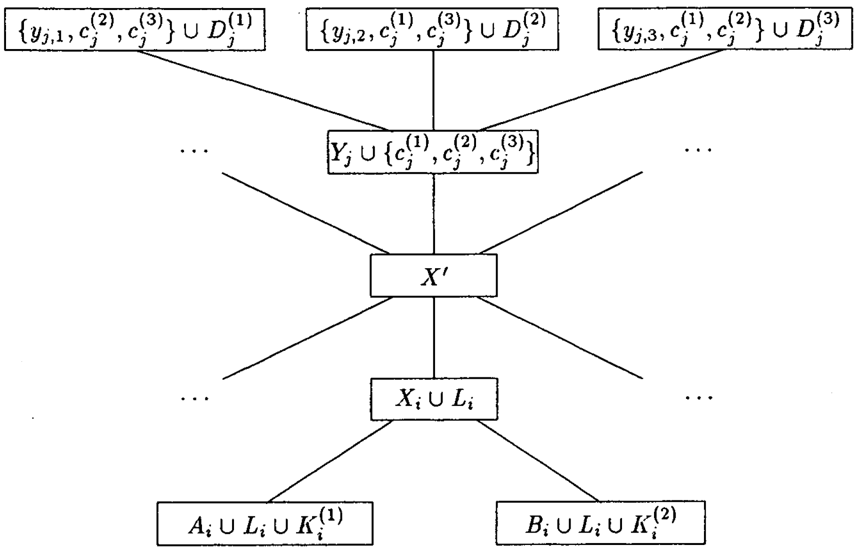

Figure 5. – An arrangement of the cliques in a tree.

Suppose we have a truth assignment, that verifies the clauses. Let y_{j,i_j} with $i_j \in \{1, 2, 3\}$ be a literal which satisfies the j-th clause. In dependence whether a variable x_i is true or false and whether a clause index i_j is 1, 2 or 3 we take the following $n+m$ cliques with total size $B = (6n+3)\cdot(n+m)$.

- if x_i is true, take $B_i \cup L_i \cup K_i^{(2)}$.
- if x_i is false, take $A_i \cup L_i \cup K_i^{(1)}$.
- if i_j is one, take $\{y_{j,1}\} \cup \{c_j^{(2)},\ c_j^{(3)}\} \cup D_j^{(1)}$.
- if i_j is two, take $\{y_{j,2}\} \cup \{c_j^{(1)},\ c_j^{(3)}\} \cup D_j^{(2)}$.
- if i_j is three, take $\{y_{j,3}\} \cup \{c_j^{(1)},\ c_j^{(2)}\} \cup D_j^{(3)}$.

We show now, if a set of $(n+m)$ cliques of total size $(6n+3)\cdot(n+m)$ is given, then there must be a truth assignment for all m clauses. For that, consider the sizes of the cliques. The maximum cliques have exactly the size $6n+3$ and can only be found under the following:

CLAUSE-CLIQUES: for $j \in \{1, \cdots, m\}$:

- $\{y_{j,1}\} \cup \{c_j^{(2)}, c_j^{(3)}\} \cup D_j^{(1)}$.
- $\{y_{j,2}\} \cup \{c_j^{(1)}, c_j^{(3)}\} \cup D_j^{(2)}$.
- $\{y_{j,3}\} \cup \{c_j^{(1)}, c_j^{(2)}\} \cup D_j^{(3)}$.

VARIABLE-CLIQUES: for $i \in \{1, \cdots, n\}$:

- $A_i \cup L_i \cup K_i^{(1)}$.
- $B_i \cup L_i \cup K_i^{(2)}$.

All other cliques in the graph have size less than $(6n+3)$ and cannot be chosen. But also the chosen sets must be pairwise disjoint, and we must take $(n+m)$ of them. So it is not possible to take for an index $i \in \{1, \cdots, n\}$ or $j \in \{1, \cdots, m\}$ more than one of the cliques; otherwise we loose at least one vertex. Therefore, the $(n+m)$ cliques $C_1, \cdots, C_{n+m}$ consists of one clause clique for each $j \in \{1, \cdots, m\}$ and one variable clique for each $i \in \{1, \cdots, n\}$.

For each $i \in \{1, \cdots, n\}$, we can define a truth assignment for the variables as follows: If $A_i \cup L_i \cup K_i^{(1)}$ is one of the $n+m$ cliques, we set x_i false, and if $B_i \cup L_i \cup K_i^{(2)}$ is one of the $n+m$ cliques we set x_i true. Consider now the clause $Y_j = \{y_{j,1}, y_{j,2}, y_{j,3}\}$. Without loss of generality we assume that $\{y_{j,1}\} \cup \{c_j^{(2)}, c_j^{(3)}\} \cup D_j^{(1)}$ is the choosen clause clique. If $y_{j,1} = x_i^{(k)}$ then B_i must be a choosen variable clique, and if $y_{j,1} = \overline{x_i^{(k)}}$ then A_i must be a choosen variable clique. In both cases, we obtain that $y_{j,1}$ is true. $\square$

6. COGRAPHS

In this section, we show that DUC is solvable in $O(|V|^2)$ time when restricted to cographs. Since each cograph is a comparability graph and since DUC is solvable in $O(D\cdot|V|^2)$ time for comparability graphs, this gives an improvement eliminating the factor D. *Cographs* are generated by disjoint union and product operations on graphs (starting with single vertex graphs) and they can be represented by a parse tree according to these operations.

For graphs $G_i = (V_i, E_i)$ with $V_1 \cap V_2 = \emptyset$, the union of G_1 and G_2, $\cup(G_1, G_2)$ is given by $(V_1 \cup V_2, E_1 \cup E_2)$. The product of G_1 and G_2, denoted by $\times(G_1, G_2)$, is obtained by first taking the union of G_1 and G_2, and then adding all the edges $\{v_1, v_2\}$ with $v_i \in V_i$.

To each cograph G, one can associate a rooted binary tree, called a *cotree* of G. Each non-leaf node in the tree is labelled either by $\cup$ (union) or by $\times$ (product) and has two children. Each node in the cotree corresponds to a cograph and a leaf corresponds to a single-vertex graph. An example of a cograph and its corresponding cotree is given in Figure 6. Corneil, Perl and Stewart [CPS] showed that it is linear time $O(|V| + |E|)$ decidable, whether a graph is a cograph. Moreover, within the same time a corresponding cotree can be constructed. We investigate here the recursion of the values $\omega_D(G)$ and the complexity of this problem restricted to cographs.

LEMMA 6.1: *Let $G = (V, E)$ be a cograph and let $d \in \mathbb{N}$.*

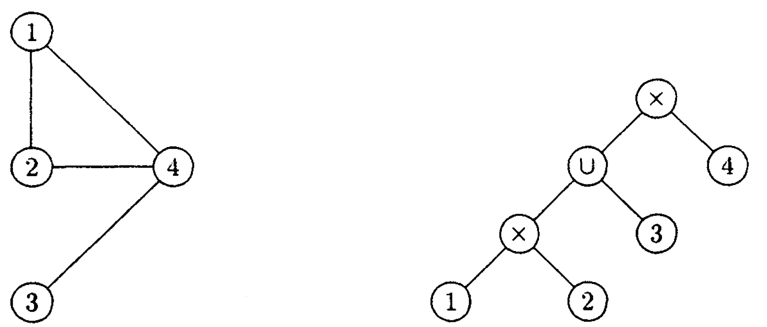

Figure 6. – A cograph and its corresponding cotree.

- *If $V = \{v\}$, then $\omega_d(G) = 1$ for $d \geq 1$ and $\omega_0(G) = 0$.*
- *If $G = \times(G_1, G_2)$, then $\omega_d(G) = \omega_d(G_1) + \omega_d(G_2)$.*
- *If $G = \cup(G_1, G_2)$, then $\omega_d(G) = \max_{0 \leq i \leq d}[\omega_i(G_1) + \omega_{d-i}(G_2)]$.*

Proof: For $V = \{v\}$, the size of d cliques can only be one (for $d > 0$). If we have a product of two cographs, we can combine each pair of cliques C_1 and C_2 where C_i is a clique in G_i. Hence, the maximum size of d cliques is given as the sum of both values for G_1 and G_2. If we have a union of two cographs, a clique only lies in one of the graphs G_1 or G_2. Then, a choice of d cliques in G equals a choice if i cliques in G_1 and $d - i$ cliques in G_2. Therefore, the maximum over all $\omega_i(G_i) + \omega_{d-i}(G_2)$, $0 \leq i \leq d$ gives $\omega_d(G)$. □

THEOREM 6.2: *For a cograph $G = (V, E)$,* DUC *can be solved in polynomial time* $O(|V|^2)$.

Proof: For each node x of the cotree T which corresponds to a cograph $G_x = (V_x, E_x)$ we must only compute the values $\omega_d(G_x)$ for $d \leq |V_x|$. Given a union of two cographs $\cup(G_1, G_2)$ with sets of vertices V_1 and V_2 we get

$$\omega_d(G) = \max\{\omega_i(G_1) + \omega_{d-i}(G_2)/0 \leq i \leq d,\ i \leq |V_1|,\ d - i \leq |V_2|\}$$

where $d \leq |V|$. Then, it follows that one can compute these values for a union and (also for a product) in at most $O(|V_1|\cdot|V_2|)$ time. Let $t(n)$ denote the maximum total time to compute all values for cotrees corresponding to cographs with n vertices. Then, we have for all $n > 1$, $t(n) \leq \max_{1 \leq i \leq n-1} c\cdot i\cdot(n-i) + t(i) + t(n-i)$, for some constant c. This follows, because if G is the union or product of two disjoint cographs G_1 and G_2 with i and $n - i$ vertices, then we get as computing time $t(i)$ for G_1, $t(n-i)$ for G_2 and at most $c\cdot i\cdot(n-i)$ for the root of the cotree. From this formula, it can be proved by induction, that there exists a constant c' with $t(n) \leq c'\cdot n^2$ for each $n \geq 1$. □

7. PARTIAL k-TREES

In this section we present a polynomial time algorithm for DUC on partial k-trees. For any integer k, *partial k-trees* are the subgraphs of k-trees. A k-trees is a graph that can be reduced to a k-clique (*i.e.* a complete graph on k vertices) by consecutively eliminating vertices of degree k with a completely connected neighbourhood. The k-trees are a natural generalization of trees. We have $k = 1$ for trees. (A tree can be reduced to a single vertex by eliminating leaves.) In Figure 7 we show an example of a 2-tree. Partial k-trees are well studied [Arn, Bo2, Go]. We give an alternative definition of partial k-trees as the graphs that have a tree-decomposition of width k below.

DEFINITION 7.1: *A* **tree-decomposition** *of width k for a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where T is an oriented tree and $\mathcal{X} = \{X_t \subseteq V,\ t \in V(T)\}$ with:*

(i) $\bigcup_{t \in V(T)} X_t = V(G)$, *and* $|X_t| \leq k+1$ *for every* $t \in V(t)$,

(ii) *for every* $\{u, v\} \in E(G)$, *there is a node* $t \in V(T)$ *such that* $\{u, v\} \subseteq X_t$,

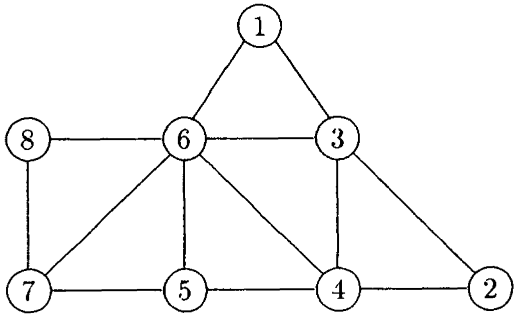

Figure 7. – A 2-tree. The vertices are numbered in elimination order.

(iii) $X_i \cap X_l \subseteq X_j$ *for all* $\{i, j, l\} \subseteq V(T)$ *such that* j *is on the path from* i *to* l *in* T.

Many intractable problems are solvable in linear time for partial k-trees. Clique and Partition into Cliques are two of them (*cf.* [Sch]). So it is natural to investigate the complexity of the generalization of these problems. It turns out, that there is a polynomial-time algorithm for Duc, too. The solution is found step by step in larger and larger subgraphs of G that are determined by the tree-decomposition. This idea was applied in [Sch] for many other problems and here it proves to be useful once more. We use a special kind of tree-decompositions to represent partial k-trees.

Definition 7.2: *A tree-decomposition* $(T, \mathcal{X})$ *is called* **nice** *if*

(i) T *is an oriented binary tree,*

(ii) $X_i = X_k = X_j$ *if* $i \in V(T)$ *has two children* j *and* k. *If* j *is the only child of* i, *then there is a vertex* $v \in V(G)$ *such that either* $X_i = X_j \cup \{v\}$ *or* $X_i = X_i \cup \{v\}$.

We give an example of a tree-decomposition in Figure 8. Nice tree-decompositions are appropriate to handle partial k-trees efficiently. In fact, the given notion is no restriction compared with the original definition by Robertson and Seymour in [RS]. Indeed, the following fact is easy to prove, *see* [Sch]:

Lemma 7.3: *If a tree-decomposition of width* k *for a graph* $G = (V, E)$ *is given, then a nice tree-decomposition of the same width can be construted in linear time. Its tree* T *has at most* $O(|V|)$ *nodes.*

It is known, that a tree-decomposition of constant width k can be constructed in polynomial time for a graph if one exists. The best algorithm

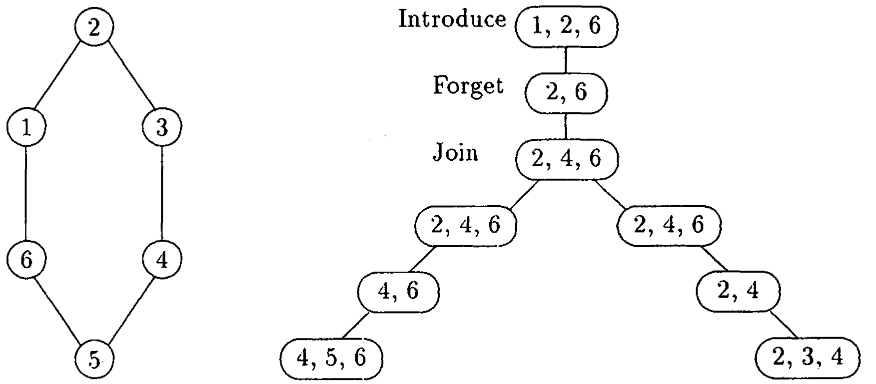

Figure 8. – The graph C_6 together with a nice tree-decomposition of width 2.

has complexity $O(|V|)$, this is a recent result of Boldaender (*see* [Bol]). Hence, given a partial k-tree, we can obtain a nice tree-decomposition with width k in linear time.

A nice tree-decomposition gives a method to reconstruct the graph consecutively by simple operations, starting with small graphs of size at most $k+1$. Using these simple operations we give an algorithm for the problem DUC. Similarly to Bern, Lawler and Wong [BLW], we consider $k+1$-**terminal graphs** that are pairs (G, X) consisting of a graph G together with a set of at most $k+1$ terminals $X \subseteq V(G)$. They are constructed by the following four operations:

Start: Take a (G, X) with $X = V(G)$. This operation corresponds to the leaves in a decomposition-tree: For a leaf t we have the subgraph of G induced by the terminal set X_t.

Forget: Take a (G, X), where $X = Y \backslash \{v\}$ for a given terminal graph (G, Y). This operation corresponds to an edge (s, t) in the decomposition tree T, where s is the only child of t and $Y = X_s = X_t \cup \{v\}$. In this case, we have the same graph as in the child, but one of its vertices is no longer a terminal (and hence, it cannot be adjacent to any subsequent vertex of G).

Introduce: Take a (G, X), where $X = Y \cup \{v\}$ and $V(G) = V(H) \cup \{v\}$ and $E(G) = E(H) \cup F$, where $F = \{\{v, y\} \in E : y \in Y\}$ for a given terminal graph (H, Y). This operation corresponds to an edge (s, t) in the decomposition tree T, where s is the only child of t and $Y = X_s = X_t \backslash \{v\}$. Here a new vertex is added to the graph corresponding to the child s of t and to its terminal set.

Join: Take a (G, X), where $G = G_1 \cup G_2$ for two given terminal graphs (G_1, X) and (G_2, X) with the same set of terminals X. This operation corresponds to a node with two children in the decomposition tree, where the two subgraphs corresponding to the children are joined by identifying their terminals pairwise.

Let $(T, \mathcal{X})$ be a tree-decomposition of a graph G. Then, we get a sequence of terminal graphs (G_t, X_t) constructed according to the four composition operations, starting with small graphs with at most $k+1$ vertices and proceeding in post-order.

Observe, that each set of terminals nodes X_t is a separator that separates the graph G_t from the rest of G. Most algorithms on partial k-trees are based on this crucial property. Moreover, we use the fact that every clique C must occur as subset $C \subseteq X_t$ in at least one node t of the decomposition tree. These basic properties of tree-decompositions are proved e.g. in [RS] and [Sch].

In Figure 8, the decomposition-tree with the topmost node choosen as is root is a parse tree reflecting the construction of the graph $G = C_6$ according to the defined operations. For example, the indicated join node corresponds to the union of two induced subgraphs $G[\{2, 4, 5, 6\}]$ and $G[\{2, 3, 4, 6\}]$ both with the terminal set $X = \{2, 4, 6\}$. The terminals are pairwise identified by the join operation. The graph G_t is a path on five vertices at this node t. In the indicated forget node, the graph remains the same but the vertex 4 is not considered as a terminal any more. In the introduce node, the new vertex 1 is added to the graph together with two incident edges. Notice, that the new vertex is always a terminal and so are its neighbors.

While solving the optimization version of DUC, we are looking for a disjoint union of cliques $C_1, \cdots, C_D$ in a graph G given with a tree-decomposition. We construct larger and larger partial solutions in the sequence of subgraphs determined by the tree-decomposition in a dynamic programming manner. Thus, we have to show that a Principle of Optimality holds. Clearly, every solution for G induces at most D disjoint cliques in every subgraph $H \subseteq G$. But notice, that this union of cliques may not be maximal in an arbitrary subgraph H. An example for this fact is shown in Figure 9. The graph G has a disjoint union of two cliques that cover the eight vertices $G[\{2, 4, 5, 6, 7, 8, 9\}]$. Only four of them are contained in the subgraph $H \subset G$. But a maximal union of two cliques in H is given by the two triangles that cover the six vertices $\{1, 3, 4, 7, 8, 10\}$.

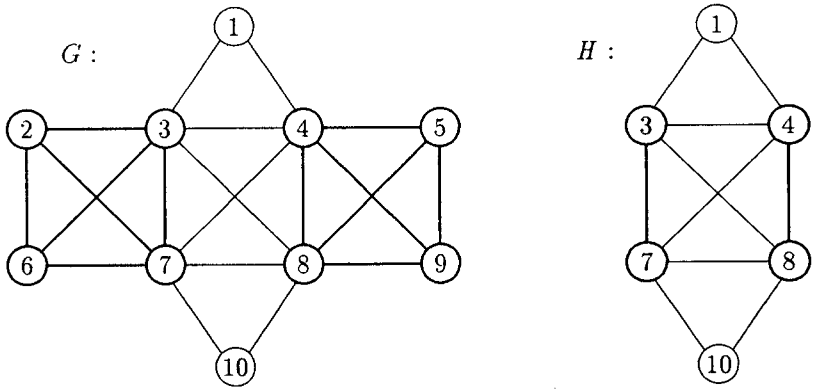

Figure 9. – A graph G with a maximal disjoint union of two cliques that is not maximal in its subgraph H.

Hence, it is not sufficient to solve DUC in the subgraphs corresponding to nodes in the decomposition tree. Nevertheless, a dynamic programming approach may be used for DUC on partial k-trees. The algorithm should consider all possibilities for a solution to cover the terminals X_t in a partial graph. For the description of the method we need some more notation.

A **partial solution** of DUC in a node t of the decomposition tree is a disjoint union of $d_t \leq D$ cliques $\{C_{t,1}, \cdots, C_{t,d_t}\}$ in the terminal graph (G_t, X_t). Denote the set of covered vertices by $K_t = \bigcup_{i=1}^{d_t} C_{t,i}$ and the set of covered terminals by $Y_t(K_t) = K_t \cap X_t$. Denote the number of vertices contained in K_t by $b_t(K_t) = |K_t|$. Clearly, the set of covered vertices K_t determines the set of cliques in the partial solution completly. So, we will identify the partial solution with this set K_t.

The Principle of Optimality holds in the following form:

LEMMA 7.4: *A partial solution K_t in the terminal graph (G_t, X_t) is contained in a solution for* DUC *in G, if and only if there is a solution L in G such that for its restriction $L_t = L \cap V(G_t)$ the following holds:*

(i) $L \cap X_t = Y_t(K_t)$,

(ii) $d_t(L_t) = d_t(K_t)$ *and*

(iii) $b_t(L_t) = b_t(K_t)$.

This is obvious because of the separator property of X_t: We may simply replace the parts K_t and L_t in the solutions. This makes it possible to examine all partial solutions for the sequence of terminal graphs (G_t, X_t) given by a tree-decomposition of G. We consider two partial solutions as equivalent, if they have the same head:

DEFINITION 7.5: *Two disjoint unions of cliques K_t and L_t in a terminal graph (G_t, X_t) are called* **equivalent** *if they fulfill the conditions $d_t(L_t) = d_t(K_t)$ and $Y_t(L_t) = Y_t(K_t)$. An equivalence class of partial solutions is represented by a pair (Y_t, d_t) that is called its* **head**. *Here we denote $d_t := d_t(K_t)$ (the number of cliques) and $Y_t := Y_t(K_t) \subseteq X_t$.*

LEMMA 7.6: *The number of equivalence classes (Y_t, d_t) for any fixed number d_t is bounded by* $2^{|X_t|}$, i.e. *it is a constant depending only on the tree-width of G.*

THEOREM 7.7: *The optimization version of* DUC *can be solved in time $O(D^2 \cdot |V|)$ for partial k-trees.*

Proof: First, we find a tree-decomposition of width k for G. This needs time $O(|V|)$ by the algorithm of Bodlaender [Bo1]. Now, we describe the dynamic programming algorithm that solves DUC in a partial k-tree G. It proceeds in post order all nodes t of the decomposition tree and computes for all equivalence classes of partial solutions with at most D cliques the following functions:

$$b_t(Y_t, d_t) = \max\{b_t(K_t) : K_t \text{ is a partial solution in } (G_t, X_t) \text{ with head } (Y_t, d_t)\}$$

The computation of the functions b_t is done recursively, starting at the leaves. We calculate the values for every pair (Y_t, d_t) that is appropriate as head of a partial solution, *i.e.* that satisfies $d_t \leq D$ and $Y_t \subseteq X_t$. Depending on the local structure of the decomposition tree the following cases occur:

Start:

$$b_t(Y_t, d_t) = \begin{cases} |Y_t| & \text{if there are } d_t \text{ disjoint cliques in } G_t \\ & \text{that cover exactly } Y_t, \\ -\infty & \text{otherwise.} \end{cases}$$

Forget: Let s be the child of t in the decomposition tree and $v \in X_s \backslash X_t$ the unique vertex of G that we forget at this node. This vertex v may be covered by a clique or not. Hence we get:

$$b_t(Y_t, d_t) = \max\{b_s(Y_t, d_t), b_s(Y_t \cup \{v\}, d_t)\}.$$

Introduce: $b_t(Y_t, d_t) = \max\{b_s(Y_t \backslash C, d_t - 1) + |C| : C \subseteq Y_t$ is a clique with $v \in C\}$ if $v \in Y_t$ (otherwise $b_t(Y_t, d_t) = b_s(Y_t, d_t)$). There s is the child of t in the decomposition tree and v is the only vertex from $X_t \backslash X_s$.

The new vertex v can improve a partial solution for the child s. In this case, one clique must be changed.

Join: Let the two children of t in the decomposition tree be s and s'. Every clique of a partial solution must be contained in one of the two subgraphs (G_s, X_s) and $(G_{s'}, X_{s'})$ (where $X_s = X_{s'} = X_t$ holds). The cliques that are in both are contained also in X_t. They are considered only once. Hence, we get: $b_t(Y_t, d_t) = \max\{b_s(Y_s, d_s) + b_{s'}(Y_{s'}, d_{s'}) : Y_t = Y_s \cup Y_{s'}, Y_s \cap Y_{s'} = \emptyset, d_t = d_s + d_{s'}\}$.

This computation needs no more time than $O(D \cdot 2^k)$ at start and forget nodes, $O(D \cdot 2^{2k})$ at introduce nodes and a most $O(D^2 \cdot 2^{2k})$ at join nodes. Since the decomposition tree has at most $|V(G)|$ nodes and k is a constant, we get at all $O(D^2 \cdot |V(G)|)$ time for these calculations.

The last step of the algorithm is to compare all partial solutions for the root r of the decomposition tree. This needs constant time. The answer for the given DUC is $b(G) = \max\{b(Y_r, D) : Y_r \subseteq X_r\}$. □

With the same approach as in the case of cographs (*see* Section 6), we get the time bound $O(|V|^2)$ for this algorithm. Furthermore, it is clear that the construction problem can be solved easily by backtracking. Here we ask not only for the maximal number of vertices that may be covered by D cliques but also for the cliques realizing this value. For this we store one feasible extension to the child (or children) maximizing function b_t in all appropriate cases during the original algorithm. Then we walk once more through the decomposition tree, now starting at the root r, to get a disjoint union of cliques in G that covers $b(G)$ vertices.

8. CONCLUSION

We have proposed an approximation algorithm for a scheduling problem with worst case ratio $O(\log |J|)$ for graph classes on which a graph theoretical problem called DUC is polynomially solvable. Using the results in this paper, the method can be applied to interval graphs, directed path graphs, cographs, comparability graphs, co-comparability graphs and partial k-trees.

Recent results on the intractibility of obtaining approximation results imply that an algorithm with an asymptotically better guarantee is unlikely to exist for the considered scheduling problem. Bellare, Goldwasser, Lund and Russel [BGLR] proved that approximating set covering within any constant factor is NP-complete. Moreover, Lund and Yannakakis [LY] showed that set covering cannot be approximated with ratio $c \cdot \log(n)$ for any constant $c < \frac{1}{4}$ unless NP is contained in DTIME$[n^{\mathrm{polylog}(n)}]$.

REFERENCES

[Arn] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability. A survey, *BIT,* 1985, *25*, pp. 2-23.

[BGLR] M. Bellare, S. Goldwasser, C. Lund and A. Russell, Efficient probabilistically checkable proofs and applications to approximation, *25th ACM Symposium on the Theory of Computing,* 1993, pp. 294-304.

[BLW] M. W. Bern, E. L. Lawler and A. L. R. Wong, Linear-time computations of subgraphs of decomposable graphs, *Journal of Algorithms,* 1987, *8*, pp. 216-235.

[Bo1] H. L. Bodlaender, A linear time algorithm for finding tree-decomposition of small treewidth, *25th Symposium on the Theory of Computing,* 1993, pp. 226-234.

[Bo2] H. L. Bodlaender, A tourist guide through treewidth, *Acta Cybernetica,* 1993, *11*, pp. 1-23.

[BL] S. Booth and S. Lueker, Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *J. Comput. Syst. Sci.,* 1976, *13*, pp. 335-379.

[Br] A. Brandstädt, Special graph classes - a survey. Report SM-DU-199, Universität-Gesamthochschule Duisburg, 1991.

[CPS] D. G. Corneil, Y. Perl and L. K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput,* 1985, *4*, pp. 926-934.

[Di] P. F. Dietz, Intersection graph algorithms, Ph. D. thesis, Comput. Sci. Dept., Cornell Univ., Ithaka, NY 1984.

[Fr] A. Frank, On chain and antichain families of a partially ordered set, *J. Combinatorial Theory (B),* 1980, *29*, pp. 176-184.

[GJ] M. R. Garey and D. S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[Ga] F. Gavril, Algorithms for maximum k-colorings and k-coverings of transitive graphs, *Networks,* 1987, *17*, pp. 465-470.

[Go] M. C. Golumbic, Algorithm graph theory and perfect graphs, Academic Press, New York, 1980.

[Ja] K. Jansen, Scheduling of incompatible jobs on unrelated machines, *International Journal of Foundations of Computer Science,* 1993, *4*, pp. 275-291.

[LY] C. Lund and M. Yannakakis, On the hardness of approximating minimization problems, *25th ACM Symposium on the Theory of Computing,* 1993, pp. 286-293.

[MV] S. Micali and V. V. Vazirani, An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matchings in general graphs, in: *Proc. 21st Ann. IEEE Symp. Foundations of Computer Science,* Long Beach, California, 1980, pp. 17-27.

[RS] N. Robertson and P. Seymour, Graph Minors. II. Algorithmic aspects of tree-width, *Journal of Algorithms,* 1986, *7*, pp. 309-322.

[Sch] P. Scheffler, Die Baumweite von Graphen als Maβ für die Kompliziertheit algorithmischer Probleme. Report RMATH-04/89, K-Weierstraβ-Institut für Mathematik, Berlin, 1989.

[YG] M. Yannakakis and F. Gavril, The maximum k-colorable subgraph problem for chordal graphs, *Information Processing Letters,* 1987, *24*, pp. 133-137.