On the complexity of the Maximum Cut problem

Hans L. Bodlaender, Klaus Jansen

RUU-CS-91-39
October 1991

On the complexity of the Maximum Cut problem

Hans L. Bodlaender, Klaus Jansen

Technical Report RUU-CS-91-39
October 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924–3275

On the complexity of the Maximum Cut problem*

Hans L. Bodlaender[†] Klaus Jansen[‡]

Abstract

The complexity of the SIMPLE MAXCUT problem is investigated for several special classes of graphs. It is shown that this problem is NP-complete when restricted to one of the following classes: chordal graphs, undirected path graphs, split graphs, tripartite graphs, and graphs that are the complement of a bipartite graph. The problem can be solved in polynomial time, when restricted to graphs with bounded treewidth, or cographs. We also give large classes of graphs that can be seen as generalizations of classes of graphs with bounded treewidth and of the class of the cographs, and allow polynomial time algorithms for the SIMPLE MAX CUT problem.

1 Introduction

One of the best known combinatorial graph problems is the MAX CUT problem. In this problem, we have a weighted, undirected graph $G = (V, E)$ and we look for a partition of the vertices of G into two disjoint sets, such that the total weight of the edges that go from one set to the other is as large as possible. In the SIMPLE MAX CUT problem, we take the variant where all edge weights are one.

Whereas the problems where we look for a partition with a *minimum* total weight of the edges between the sets are solvable in polynomial time with flow techniques, the (decision variants of the) MAX CUT, and even the SIMPLE MAX CUT problems are NP-complete [11, 9]. This motivates the research to solve the (SIMPLE) MAX CUT problem on special classes of graphs.

*The results of section 5 of this paper have already been published in: H.L. Bodlaender, The Maximum Cut and Minimum Cut Into Bounded Sets problems on Cographs, Technical Report RUU-CS-87-12, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1987.

[†]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands. The work of this author was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (project ALCOM).

[‡]Fachbereich IV, Mathematik und Informatik, Universität Trier, Postfach 3825, W-5500 Trier, Germany

In [10] Johnson gives a table of the known results on the complexity of SIMPLE MAX CUT restricted to several classes of graphs. The most notable of the results listed there, is perhaps the fact that SIMPLE MAX CUT can be solved in polynomial time on planar graphs. Several cases however remain open. In this paper we resolve some of the open cases.

This paper is mostly concerned with the SIMPLE MAX CUT problem. In section 8 we comment on the MAX CUT problem (i.e., the problem where edges do not necessarily have unit weights.) Some applications of the MAXCUT problem are given in [4, 5, 13].

This paper is organized as follows. In section 2 we consider the chordal graphs, and the undirected path graphs. In section 3, we consider the split graphs. In section 4, we consider tripartite graphs, and complements of bipartite graphs. In section 5, we consider cographs. An algorithm to solve SIMPLE MAX CUT on graphs with bounded treewidth is described in section 6. In section 7, the results of sections 5 and 6 are generalized. Finally, in section 8 we comment on the problem with arbitrary edge weights.

We conclude this introduction with some definitions. We first give a precise description of the SIMPLE MAX CUT problem.

Problem: SIMPLE MAX CUT
Input: Undirected graph $G = (V, E)$, $k \in \mathbb{N}$.
Question: Does there exist a set $S \subset V$, such that $|\{(s,u) \in E | s \in S, u \in V - S\}| \geq k$?

If we have a partition of V into sets $S \subseteq V$, and $V - S$, then an edge $(u,v) \in E$ with $u \in S$, $v \in V - S$ is called a *cut edge.*

2 Chordal graphs

In this section we analysed the SIMPLE MAX CUT problem for chordal graphs. A graph is chordal, if and only if it does not contain a cycle of length at least four as an induced subgraph. Alternatively, a graph is chordal, if and only if there exists a tree $T = (W, F)$ such that one can associate with each vertex $v \in V$, a subtree $T_v = (W_v, E_v)$ of T, such that $(v,w) \in E$ iff $W_v \cap W_w \neq \emptyset$. This is equivalent to stating that all maximal cliques of G can be arranged in a tree T, such that for every vertex v, the cliques that contain v form a connected subtree of T. (In other words: chordal graphs are the intersection graphs of subtrees of trees.)

We will show that SIMPLE MAX CUT is NP-complete for chordal graphs. Hereto, we use the MAX 2-SAT problem, described below.

Problem: MAX 2-SAT
Input: A set of p disjunctive clauses each containing at most two literals and an integer $k \leq p$.

Question: Is there a truth assignment to the variables which satisfies at least k clauses ?

MAX 2-SAT was proven to be NP-complete by Garey, Johnson and Stockmeyer [9]. (In [9] also a transformation from MAX 2-SAT to the SIMPLE MAX CUT problem for undirected graphs was given.) We note [8] that 3-SAT remains NP-complete if for each variable there are at most five clauses that contain either the variable or its complement. Using the reduction of Garey, Johnson and Stockmeyer [9] we can obtain a similar result for MAX 2-SAT such that for each variable there are at most 20 clauses containing the variable or its complement. It is possible to replace the number 20 by the smaller constant six using a different construction. In this construction each literal (variable or its complement) occurres at most three times.

Theorem 2.1 SIMPLE MAX CUT *is NP-complete for chordal graphs.*

Proof:
(We will omit in this and all later proofs the statement that the problems are in NP.)

We give a transformation from MAX 2-SAT to SIMPLE MAX CUT for chordal graphs. Let $X = \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ be a set of variables, let $(a_1 \vee b_1), \ldots, (a_p \vee b_p)$ denote a set of clauses.

Let $m = 2p$. First we define a number of sets:

- for each $i \in \{1, \ldots, n\}$ take $C^{(i)} = \{c_1^{(i)}, \ldots, c_{m+1}^{(i)}\}$, $D^{(i)} = \{d_1^{(i)}, \ldots, d_{m+2}^{(i)}\}$, $E^{(i)} = \{e_1^{(i)}, \ldots, e_{m+2}^{(i)}\}$.
- take $S = \{s_1, \ldots, s_{m+1}\}$.
- take $R = \{r_1, \ldots, r_p\}$; take $Q = \{q_1, \ldots, q_p\}$.
- for each $i \in \{1, \ldots, p\}$, take $T^{(i)} = \{t_1^{(i)}, \ldots, t_{m+2}^{(i)}\}$.
- take $U = \{u_1, \ldots, u_p\}$, $V = \{v_1, \ldots, v_p\}$, $W = \{w_1, \ldots, w_p\}$, and $Z = \{z_1, \ldots, z_p\}$.
- take $Y = \{y_1, \ldots, y_{p+2n}\}$.

We now define the input graph $G' = (V', E')$ for the SIMPLE MAX CUT problem. V' is the disjoint union of all sets: X, $C^{(i)}$ $(1 \leq i \leq n)$, $D^{(i)}$ $(1 \leq i \leq n)$, $E^{(i)}$ $(1 \leq i \leq n)$, S, R, Q, $T^{(i)}$ $(1 \leq i \leq p)$, U, V, W, and Z. There is an edge between a pair of vertices in G', if and only if at least one of the following sets contains both vertices, i.e. each of the following sets forms a clique in G':

- for each $i \in \{1, \ldots, n\}$:
  - $\{x_i\} \cup C^{(i)} \cup E^{(i)}$,

  - $\{x_i\} \cup C^{(i)} \cup D^{(i)}$,
  - $\{x_i, \overline{x_i}\} \cup C^{(i)}$,

- for each $j \in \{1, \ldots, m+1\}$, take a set (clique) $R \cup \{s_j\}$.
- $X \cup R \cup Y$.
- for each $i \in \{1, \ldots, p\}$, and each $j \in \{1, \ldots, m+2\}$, take a set (clique) $\{r_i, q_i, t_j^{(i)}\}$.
- for each $i \in \{1, \ldots, p\}$: if the ith clause is $(a_i \vee b_i)$, then take sets (cliques)
  - $\{a_i, b_i, r_i, q_i\}$
  - $\{a_i, b_i, v_i, w_i\}$
  - $\{a_i, u_i, v_i\}$
  - $\{b_i, v_i, z_i\}$

First, we claim that the graph G', formed in this way is chordal. This follows because we can arrange all cliques in a tree T, such that every vertex belongs to a set of trees that forms a connected subtree of T, illustrated in figure 1. (Alternatively, one can check by tedious case analysis that G' does not contain an induced cycle of length more than 3.)

In order to count the maximum number of possible cut edges, we consider six types of edges:

1. Edges between vertices in $C^{(i)} \cup D^{(i)} \cup E^{(i)} \cup \{x_i\}$, for some $i \in \{1, \ldots, n\}$.
2. Edges of the form $(\overline{x_i}, c_j^{(i)})$.
3. Edges between vertices in $X \cup R \cup Y$.
4. Edges of the form (r_i, s_j).
5. Edges of the form (r_i, q_i), $(r_i, t_j^{(i)})$, $(q_i, t_j^{(i)})$, for some $i \in \{1, \ldots, n\}$.
6. Edges of the form (q_i, a_i), (q_i, b_i), (a_i, u_i), (a_i, v_i), (a_i, w_i), (b_i, v_i), (b_i, w_i), (b_i, z_i), (u_i, v_i), (v_i, w_i), (w_i, z_i), for some $i \in \{1, \ldots, n\}$, where the ith clause is $(a_i \vee b_i)$.

Note that each edge of G' has exactly one type.

Write $B = 2n \cdot (m+2)^2 + n \cdot (m+1) + (2n+p)^2 + p \cdot (m+1) + 2p \cdot (m+2) + 6p$. We now claim that G' has a partition with at least $B + 2k$ cut edges, if and only there is a truth assignment, that verifies at least k clauses.

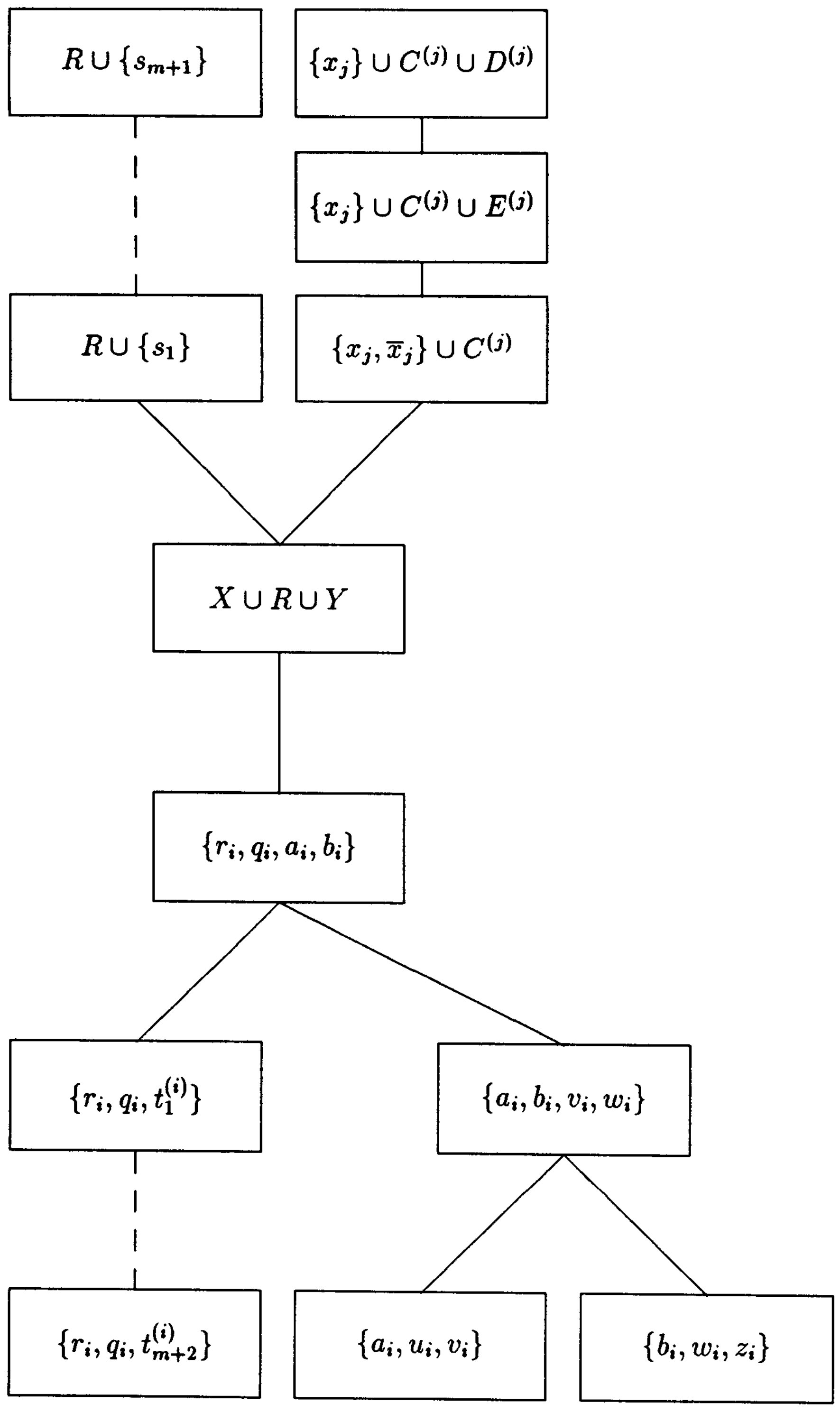
$R \cup \{s_{m+1}\}$
$\{x_j\} \cup C^{(j)} \cup D^{(j)}$
$\{x_j\} \cup C^{(j)} \cup E^{(j)}$
$R \cup \{s_1\}$
$\{x_j, \overline{x}_j\} \cup C^{(j)}$
$X \cup R \cup Y$
$\{r_i, q_i, a_i, b_i\}$
$\{r_i, q_i, t_1^{(i)}\}$
$\{a_i, b_i, v_i, w_i\}$
$\{r_i, q_i, t_{m+2}^{(i)}\}$
$\{a_i, u_i, v_i\}$
$\{b_i, w_i, z_i\}$

Figure 1: Clique arrangement of G'.

Suppose we have a truth assignment, that verifies at least k clauses. We construct a partition $V' = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ in the following way:

$$\begin{aligned} V_1 = \; & R \cup Q \cup \{x_i, c_j^{(i)} | x_i \text{ false } \} \\ & \cup \{\overline{x_i}, d_j^{(i)}, e_j^{(i)} | x_i \text{ true } \} \\ & \cup \{u_i, z_i | a_i \text{ false } \wedge b_i \text{ false } \} \\ & \cup \{v_i, z_i | a_i \text{ false } \wedge b_i \text{ true } \} \\ & \cup \{u_i, w_i | a_i \text{ true } \wedge b_i \text{ false } \} \\ & \cup \{v_i, w_i | a_i \text{ true } \wedge b_i \text{ true } \} \\ & \cup \{y_1, \ldots, y_n\} \\ V_2 = \; & V \setminus V_1. \end{aligned}$$

We have $2n \cdot (m+2)^2$ cut edges of type 1, $n \cdot (m+1)$ cut edges of type 2, $(2n+p)^2$ cut edges of type 3, $p \cdot (m+1)$ cut edges of type 4, and $2p \cdot (m+2)$ cut edges of type 5. For a clause that is true, the number of type 6 cut edges corresponding to that clause is eight, and for a clause that is false, this number is six. Hence, the total number of cut edges of type 6 is $6p + 2k$. The total number of cut edges of all types is precisely $B + 2k$.

We now show, that when we have a partition of V' in sets V_1, V_2 with at least $B + 2k$ cut edges, then there must be a truth assigment with at least k true clauses. We consider for each type of edges the maximum number of cut-edges. We compare these numbers with the numbers obtained in the partition formed above. We often use that the maximum number of cut edges in a clique of size $2r$ is r^2.

Consider edges of type 1. There are at most $(m+2)^2$ cut edges between vertices in a set $C^{(i)} \cup D^{(i)} \cup \{x_i\}$, and similarly for a set $C^{(i)} \cup D^{(i)} \cup \{x_i\}$. Hence, we cannot gain cut edges with respect to the partition defined above. Moreover, if not $C^{(i)} \cup \{x_i\} \subseteq V_1$ or $C^{(i)} \cup \{x_i\} \subseteq V_2$, then we loose at least $m + 1$ cut edges of type 1.

As every edge of type 2 is in the partition defined above a cut edge, we cannot gain cut edges of type 2. However, if for some i, x_i and $\overline{x_i}$ belong to the same set (V_1 or V_2), then we loose at least $m + 1$ cut edges of type 1 and 2.

There are at most $(|X \cup R \cup Y|/2)^2$ cut edges of type 3. Hence, no cut edges of type 3 can be gained.

Again, as every edge of type 4 is a cut edge in the partition defined above, no gain of type 4 cut edges is possible. However, if not $R \subseteq V_1$ or $R \subseteq V_2$, then every vertex in S is adjacent to only one cut edge, and we have a loss of at least $m + 1$ cut edges.

With a similar argument, it follows that no gain of cut edges of type 5 is possible, but if for some i, q_i and r_i do not belong to the same set, there is a loss of at least $m + 1$ cut edges.

A simple case analysis shows, that the number of cut edges of type 6, for one clause $(a_i \vee b_i)$ is at most eight. Hence, the maximum number of type 6 edges that can be gained is at most $2p$.

It follows that we cannot have a partition with at least $B + 2k$ (and even a partition with at least B) cut edges, if we loose somewhere $m + 1 = 2p + 1$ edges. It follows that $R \cup Q \subseteq V_1$ or $R \cup Q \subseteq V_2$. It also follows that for each i, x_i and $\overline{x_i}$ belong to a different set. Call x_i true, if it does not belongs to the same set in the partition as $R \cup Q$, otherwise call x_i false.

We claim that this truth assignment satisfies at least k clauses. We have shown that the number of cut edges of types 1 to 5 is at most $B - 6p$. So, there must be at least $6p + 2k$ cut edges of type 6. If the clause $(a_i \vee b_i)$ is false, then a_i, b_i and q_i belong to the same set. In this case, at most six cut edges of type 6 can be obtained in the subgraph formed by the eleven edges of type 6 for this value of i. If the clause is true, the maximum number of cut edges that can be obtained in the subgraph is precisely eight. Hence, at least k clauses must be true. This proves the claim. NP-hardness of the problem follows, because G' can be constructed in polynomial time. □

Now we analyse a subclass of the chordal graphs, the undirected path graphs. A graph is an undirected path graph, if it is the intersection graphs of paths in an (unrooted, undirected) tree. In other words, $G = (V, E)$ is an undirected path graph, if and only if there exists a tree $T = (W, F)$, and for every vertex $v \in V$ a path P_v in T, such that for all pairs of vertices $v, w \in V$, $v \neq w$: $(v, w) \in E$, if and only if P_v and P_w have at least one vertex in common.

Theorem 2.2 SIMPLE MAX CUT *is NP-complete for undirected path graphs.*

Proof:
We can show this by changing the construction from the proof above. We use that MAX 2-SAT remains NP-complete, when for each variable, the number of clauses that contains the variable is bounded by the constant 3. Note that for almost every type of vertex in the proof above, its corresponding subtree of T is a path. The only exception to this are the vertices in X. Hence, we must change the construction with respect to the vertices in X. This is done in the following way.

We replace each variable x_i by $x_{i,1}, \ldots, x_{i,3}$ and $\overline{x_i}$ by $\overline{x_{i,1}}, \ldots, \overline{x_{i,3}}$. We enlarge the sets $D^{(i)}$ and $E^{(i)}$ by two vertices. It then can be argued, that if we do not have $\{x_{i,1}, \ldots, x_{1,3}\} \subseteq W_1$ and $\{\overline{x_{i,1}}, \ldots, \overline{x_{1,3}}\} \subseteq W_2$, or $\{x_{i,1}, \ldots, x_{1,3}\} \subseteq W_2$ and $\{\overline{x_{i,1}}, \ldots, \overline{x_{1,3}}\} \subseteq W_1$, then there is a loss of at least $m + 1$ edges.

For the jth occurrence of x_i in a clause we use instead of x_i the vertex $x_{i,j}$. The same arguments as in the previous proof now apply for this new graph. However, the vertices of this graphs can be represented as undirected paths in the tree corresponding to the set of cliques, hence we have an undirected path graph. □

3 Split graphs

A graph $G = (V, E)$ is a split graph, if and only if there is a partition of the vertices V of G into a clique C and an independent set U. Another necessary and sufficient condition for a graph G to be a split graph is that G and its complement G^c are chordal graphs, see also Földes and Hammer [7]. We analyse in this section a subclass of the split graphs, namely the class of those split graphs where each vertex of the independent set U is incident to exactly two vertices of the clique C. We call these graphs the 2-split graphs.

Theorem 3.1 SIMPLE MAX CUT *is NP-complete for 2-split graphs.*

Proof:
We use a transformation from the (unrestricted) SIMPLE MAX CUT problem. Let a graph $G = (V, E)$ be given. Let $G^c = (V, E^c)$ be the complement of G. Let $H = (V \cup E^c, F)$, where $F = \{(v, w) \mid v, w \in V,\ v \neq w\} \cup \{(v, e) \mid v \in V,\ e \in E^c,\ v$ is an endpoint of edge $e\}$. In other words, we take a vertex in H for every vertex in G and every edge in the complement of G. V forms a clique, E^c forms an independent set in H. Every edge-representing vertex is connected to the vertices, representing its endpoints.

We claim that G allows a partition with at least K cut edges, if and only if H allows a partition with at least $2 \cdot |E^c| + K$ cut edges.

Suppose we have a partition W_1, W_2 of G with at least K cut edges. We partition the vertices of H as follows: partition V as in the partition of G; for every $e \in E^c$: if both endpoints of e belong to W_1, then put e in W_2, otherwise put e in W_1. It is easy to see that this partition gives the desired number of cut edges.

Now suppose we have a partition of H into sets W_1, W_2, that has at least $2 \cdot |E^c| + K$ cut edges. Partition the vertices of G in sets $W_1 \cap V$, $W_2 \cap V$. This partition gives the desired number of cut edges. This can be noted as follows: for every edge $(v, w) \in E$, we have one cut edge in H if (v, w) is a cut edge in G, otherwise we have no cut edge. For every edge $e = (v, w) \in E^c$, we have that of the three edges (v, w), (e, v), and (e, w), exactly two will be a cut edge. Hence, the total number of cut edges in H equals the number of cut edges in G plus $2 \cdot |E^c|$.

The theorem now follows, because H can be constructed from G is polynomial time. □

Double interval graphs are the intersection graphs of sets $A_1, \ldots, A_n$ such that for all i, $1 \leq i \leq n$, A_i is the union of two closed intervals of the real line. These graphs are introduced by Harary and Trotter in [15].

Lemma 3.2 *Each 2-split graph is a double interval graph.*

Proof:
Let $G = (V, E)$ be a 2-split graph with a clique $C = \{c_1, \ldots, c_n\}$ and an independent

set $U = \{u_1, \dots, u_m\}$ where each element u_j is connected with two vertices a_j, b_j of the clique. Define for each vertex a set of two intervals such that G is the intersection graph of the corresponding sets. For each vertex $c_i \in C$ take $T(c_i) = \{[m(i-1)+1, mi], [mn+1, mn+1]\}$ and for each vertex $u_j \in U$ take $T(u_j) = \{[m(a_j-1)+j, m(a_j-1)+j], [m(b_j-1)+j, m(b_j-1)+j]\}$. □

Hence, as consequence we get:

Corollary 3.3 SIMPLE MAX CUT *is NP-complete for double interval graphs.*

4 Graphs, related to bipartite graphs

Bipartite graphs are graphs $G = (V, E)$ in which the vertex set can be partitioned into two sets V_1 and V_2 such that no edge joins two vertices in the same set. Simple MAXCUT is trivial for bipartite graphs. Thus, it is interesting to look at related graph classes. We consider the tripartite graphs, and the graphs that are the complement of a bipartite graph. The latter graphs we call the co-bipartite graphs.

A generalization of bipartite graphs are the tripartite graphs $G = (V, E)$. A graph is tripartite, if and only if the vertex set can be partitioned into three independent sets V_1, V_2 and V_3. In other words, a graph is tripartite if its chromatic number is at most three.

Theorem 4.1 SIMPLE MAX CUT *is NP-complete for tripartite graphs.*

Proof:
By transformation from SIMPLE MAX CUT for split graphs to tripartite graphs. Let $G = (V, E)$ be a split graph, where the vertex set is partitioned into a clique C and an independent set U, and define a graph $\overline{G} = (\overline{V}, \overline{E})$. For each pair $c_i, c_j \in C$ with $i \neq j$ define a graph $G_{\{i,j\}}$ with vertex set

$$\begin{aligned} V_{\{i,j\}} = \ & \{c_i, c_j, x_{\{i,j\}}, z_{\{i,j\}}, y_{\{i,j\}}, u_{\{i,j\}}\} \\ E_{\{i,j\}} = \ & \{(x_{\{i,j\}}, c_i), (z_{\{i,j\}}, c_i), (y_{\{i,j\}}, c_i), \\ & (z_{\{i,j\}}, c_j), (y_{\{i,j\}}, c_j), (u_{\{i,j\}}, c_j), \\ & (x_{\{i,j\}}, y_{\{i,j\}}), (z_{\{i,j\}}, y_{\{i,j\}}), (z_{\{i,j\}}, u_{\{i,j\}})\} \end{aligned}$$

and replace the edge (c_i, c_j) by the graph $G_{\{i,j\}}$. The resulting graph is tripartite and we can show that G allows a partition with at least k cut edges if and only if $\overline{G}$ allows a partition with at least $k + 3|C|(|C|-1)$ cut edges.

If $c_i \in S$ and $c_j \notin S$, we have in G one edge and get in $G_{\{i,j\}}$ seven edges, if we put $y_{\{i,j\}}, u_{\{i,j\}}$ into $\overline{S}$. If for $i \neq j$ both vertices $c_i, c_j \in S$, we can get at most six edges in $G_{\{i,j\}}$. We can obtain this by putting $y_{\{i,j\}}$ into $\overline{S}$. □

Theorem 4.2 SIMPLE MAX CUT *is NP-complete for co-bipartite graphs.*

Proof:
We use a transformation from the SIMPLE MAX CUT problem, restricted to split graphs. Suppose $G = (C \cup U, E)$ is a split graph, U forms an independent set, and C forms a clique in G. Take a set U', disjoint from $C \cup U$, with $|U'| = |U|$. Let $H = (C \cup U \cup U', E \cup \{(v, w) \mid v \neq w,\ v, w \in U \cup U'\})$. In other words, H is obtained from G by adding the vertices in U', and putting a clique on $U \cup U'$. Clearly, H is a co-bipartite graph.

We claim that G has a partition with at least K cut edges, if and only if H has a partition with at least $|U|^2 + K$ cut edges. Suppose we have a partition W_1, W_2 of G with at least K cut edges. Extend this partition by putting exactly $|W_1 \cap U|$ vertices from U' in W_2, and putting the other $|W_2 \cap U|$ vertices from U' in W_1. Now $|W_1 \cap (U \cup U')| = |W_2 \cap (U \cup U')| = |U|$. Hence, there are in H $|U|^2$ cut edges in the clique on $U \cup U'$, and at least K cut edges in the remainder of H.

Next, suppose we have a partition W_1, W_2 of H with at least $|U|^2 + K$ cut edges. There are at most $|U|^2$ of these cut edges that go between two vertices in $U \cup U'$. Hence, the partition $(C \cup U) \cap W_1$, $(C \cup U) \cap W_2$ of G contains at least K cut edges.
□

5 Cographs

In this section, we show that the SIMPLE MAX CUT problem can be solved in $O(n^2)$ time, when restricted to cographs. First, we need some definitions.

Definition 5.1 *Let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be two graphs, with V_1 and V_2 disjoint sets. The disjoint union of G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The product of G_1 and G_2 is the graph $G_1 \times G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(v, w) \mid v \in V_1,\ w \in V_2\})$.*

Definition 5.2 *The class of cographs is the smallest set of graphs, fulfilling the following rules:*

1. *Every graph $G = (V, E)$ with one vertex and no edges ($|V| = 1$ and $|E| = 0$) is a cograph.*

2. *If $G_1 = (V_1, E_1)$ is a cograph, $G_2 = (V_2, E_2)$ is a cograph, and V_1 and V_2 are disjoint sets, then $G_1 \cup G_2$ is a cograph.*

3. *If $G_1 = (V_1, E_1)$ is a cograph, $G_2 = (V_2, E_2)$ is a cograph, and V_1 and V_2 are disjoint sets, then $G_1 \times G_2$ is a cograph.*

To each cograph G one can associate a corresponding rooted binary tree T, called the *cotree* of G, in the following way. Each non-leaf node in the tree is labeled with either "$\cup$" (union-nodes) or "$\times$" (product-nodes). Each non-leaf node has exactly two children. Each node of the cotree corresponds to a cograph. A leaf node corresponds to a cograph with one vertex and no edges. A union-node (product-node) corresponds to the disjoint union (product) of the cographs, associated with the two children of the node. Finally, the cograph that is associated with the root of the cotree is just G, the cograph represented by this cotree.

We remark that the most usual definition of cotrees allows for arbitrary degree of internal nodes. However, it is easy to see that this has the same power, and can easily be transformed in cotrees with two children per internal node. In [6], it is shown that one can decide in $O(n+e)$ time, whether a graph is a cograph, and build a corresponding cotree.

Our algorithm has the following structure: first find a cotree for the input graph G, which is a cograph. Then for each node of the cotree, we compute a table, called $maxc_H$, where H is the cograph corresponding to the node. These tables are computed 'bottom-up' in the cotree: first all tables of leaf-nodes are computed, and in general a table of an internal node is computed after the tables of its two children are computed.

Let $H = (V', E')$ be a cograph. The table $maxc_H$ has entries for all integers i, $0 \leq i \leq |V'|$, that denote the maximum size of a cut of H into a set of size i and a set of size $|V'| - i$, in other words:

$$maxc_H(i) = \max\{|\{(v,w) \mid v \in W_1,\ w \in W_2\}| \mid W_1 \cup W_2 = V',\ W_1 \cap W_2 = \emptyset,\ |W_1| = i\}$$

Clearly, the size of the maximum cut of G is $\max_{0 \leq i \leq |V|} maxc_G(i)$, hence, when we have the table $maxc_G$, i.e., the table of the root node of the cotree, then we know the size of the maximum cut. It remains to show that the tables can be computed efficiently.

For leaf nodes, the corresponding cograph H has one vertex and no edges, hence the tables associated with leaf nodes are all of the form: $maxc_H(0) = 0$, $maxc_H(1) = 0$.

The following lemma shows how a table $maxc_{G_1 \cup G_2}$ or a table $maxc_{G_1 \times G_2}$ can be computed, after the tables $maxc_{G_1}$ and $maxc_{G_2}$ are computed.

Lemma 5.3 *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs, with V_1 and V_2 disjoint sets. Then:*
(i) $maxc_{G_1 \cup G_2}(i) = \max\{maxc_{G_1}(j) + maxc_{G_2}(i-j) \mid 0 \leq j \leq i,\ j \leq |V_1|,\ i-j \leq |V_2|\}$.
(ii) $maxc_{G_1 \times G_2}(i) = \max\{maxc_{G_1}(j) + maxc_{G_2}(i-j) + j \cdot (|V_2| - (i-j)) + (|V_1| - j) \cdot (i-j) \mid 0 \leq j \leq i,\ j \leq |V_1|,\ i-j \leq |V_2|\}$.

Proof:
Consider a partition of $V_1 \cup V_2$ into two disjoint sets W_1 and W_2, with $|W_1| = i$. If $V_1 \cap W_1$ contains exactly j vertices, then $0 \leq j \leq i$; $j \leq |V_1|$; there are exactly $i-j$ vertices in $V_2 \cap W_1$, and hence $i - j \leq |V_2|$. The maximum number of cut edges

in the graph $G_1 \cup G_2$, under the assumption that there are j vertices in $V_1 \cap W_1$, and $i - j$ vertices in $V_2 \cap W_1$ is $maxc_{G_1}(j) + maxc_{G_2}(i - j)$. In case of the graph $G_1 \times G_2$, there are additionally $j \cdot (|V_2| - (i - j))$ edges between vertices in $V_1 \cap W_1$ and vertices in $V_2 \cap W_2$, and $(i - j) \cdot (|V_1| - j)$ edges between vertices in $V_2 \cap W_1$ and vertices in $V_1 \cap W_2$. □

It directly follows that one can compute the table $maxc_{G_1 \cup G_2}$ and $maxc_{G_1 \times G_2}$ in $O(|V_1| \cdot |V_2|)$ time.

Let $t(n)$ denote the maximum total time to compute all tables $maxc$ for cotrees corresponding to cographs with n vertices. We have that 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. (G must be the product or union of two cographs G_1, G_2. In case G_1 has i vertices, computing all tables of the subtree rooted at the node representing G_1 costs at most $t(i)$ time, computing all tables of the subtree rooted at the other child of the root costs at most $t(n - i)$ time, and computing the table of the root-node costs $O(i(n - i))$ time.) From this formula, it follows by induction, that there exists a constant c', such that for all integers $n \geq 1$, $t(n) \leq c' \cdot n^2$.

Theorem 5.4 *There exists an $O(n^2)$ algorithm for* SIMPLE MAX CUT *on cographs.*

It is possible to modify this algorithm such that it also *yields a partition* with the maximum number of cut edges. This modification uses also $O(n^2)$ time. The modification can be carried out in the following way: for each table $maxc_H$ associated with an internal node, and for each entry in such a table, store, when computing this table, also the value of j where the maximum as described in lemma 5.3. More formal, store values $maxval_{H_1 \cup H_2}$ and $maxval_{H_1 \times H_2}$, with if $maxval_{H_1 \cup H_2} = j$, then $maxc_{G_1 \cup G_2}(i) = maxc_{G_1}(j) + maxc_{G_2}(i - j)$, and if $maxval_{H_1 \times H_2} = j$, then $maxc_{G_1 \times G_2}(i) = \max\{maxc_{G_1}(j) + maxc_{G_2}(i-j) + j \cdot (|V_2| - (i-j)) + (|V_1| - j) \cdot (i-j)\}$. These values are obtained as a by-product of the algorithm described above. When all tables are computed, the partition of G with the maximum cut can easily be constructed. Suppose the maximum value in the table $maxc_G$ is $maxc_G(i_0)$, and $maxval_G(i_0) = j_0$. We construct the desired partition with i_0 vertices in W_1 as follows. If G consists of one vertex, then put this vertex in W_1, if and only if $i_0 = 1$. Otherwise, G is disjoint union or product of two graphs H_1, H_2. We know the maximum cut is achieved if we choose j_0 vertices of W_1 in H_1, and $i_0 - j_0$ vertices of W_1 in H_2. Recursively construct the partitions of H_1 and H_2 with these numbers of vertices in W_1. The resulting partition gives the maximum cut. It is straightforward, that the extra overhead is bounded by $O(n^2)$.

6 Graphs with bounded treewidth

In this section we show how the SIMPLE MAX CUT problem can be solved on graphs with bounded treewidth. This result is already well-known (see e.g. [16] and neither are the techniques used here very novel. We include the result with proof here, as it will be generalized in the next section. The notion of treewidth of a graph was introduced by Robertson and Seymour [14], and is equivalent to several other interesting graph theoretic notions, for instance the notion of partial k-trees (see e.g., [1, 2]).

Definition 6.1 *A tree-decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of V, and $T = (I, F)$ is a tree, such that the following conditions hold:*

1. $\bigcup_{i \in I} X_i = V$.
2. *For all edges $(v, w) \in E$, there exists a node $i \in I$, with $v, w \in X_i$.*
3. *For every vertex $v \in V$, the subgraph of T, induced by the nodes $\{i \in I \mid v \in X_i\}$ is connected.*

The treewidth of a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph is the minimum treewidth over all possible tree-decompositions of the graph.

It is possible to make small modifications to a tree-decomposition, without increasing its treewidth, such that one can see T as a rooted tree, with root $r \in I$, and the following conditions hold:

1. T is a binary tree.
2. If a node $i \in I$ has two children j_1 and j_2, then $X_i = X_{j_1} = X_{j_2}$.
3. If a node $i \in I$ has one child j, then either $X_j \subset X_i$ and $|X_i - X_j| = 1$, or $X_i \subset X_j$ and $|X_j - X_i| = 1$.

We will assume in the remainder that a tree-decomposition of G of this type is given, with treewidth at most k, for some constant k. Note that a tree-decomposition of G with treewidth $\leq k$ can be found, if it exists, in $O(n \log^2 n)$ time [3]. We omit the description of the transformations to the above form from this version of the paper.

For every node $i \in I$, let Y_i denote the set of all vertices in a set X_j with $j = i$ or j is a descendant of i in the rooted tree T. Our algorithm is based upon computing for every node $i \in I$ a table $maxc_i$. For every subset S of X_i, there is an entry in the table $maxc_i$, fulfilling

$$maxc_i(S) = \max_{S' \subseteq Y_i,\ S' \cap X_i = S} |\{(v, w) \in E \mid v \in S',\ w \in Y_i - S'\}|.$$

In other words, for $S \subseteq X_i$, $maxc_i(S)$ denotes the maximum number of cut edges for a partition of Y_i, such that all vertices in S are in one set in the partition, and all vertices in $X_i - S$ are in the other set in the partition.

The tables are again computed in a bottom-up manner: start with computing the tables for the leaves, then always compute the table for an internal node later than the tables of its child or children are computed. The following lemma shows how the tables can computed efficiently:

Lemma 6.2 *(i) Let i be a leaf in T. Then for all $S \subseteq X_i$, $maxc_i(S) = |\{(v,w) \in E \mid v \in S,\ w \in X_i - S\}|$.*
(ii) Let i be a node with one child j in T. Suppose $X_i \subseteq X_j$. Then for all $S \subseteq X_i$, $maxc_i(S) = \max_{S' \subseteq X_j,\ S' \cap X_i = S} maxc_j(S')$.
(iii) Let i be a node with one child j in T. Suppose $X_j \cup \{v\} = X_i$, $v \notin X_j$. For all $S \subseteq X_i$, if $v \in S$, then $maxc_i(S) = maxc_j(S - \{v\}) + |\{(s,v) \mid v \in X_i - S\}|$, and if $v \notin S$, then $maxc_i(S) = maxc_j(S) + |\{(s,v) \mid v \in S\}|$.
(iv) Let i be a node with two childs j_1, j_2 in T, with $X_i = X_{j_1} = X_{j_2}$. For all $S \subseteq X_i$, $maxc_i(S) = maxc_{j_1}(S) + maxc_{j_2}(S) - |\{(v,w) \in E \mid v \in S,\ w \in X_i - S\}|$.

Proof:
It is easy to see that these conditions hold from the definitions. The subtraction in case (iv) is needed because these edges are otherwise counted twice. □

From the lemma, one can see that computing a table $maxc_i$ can be done in $O(1)$ time. So, in $O(n)$ time, one can compute the table of the root r. The size of the maximum cut is $\max_{S \subseteq X_r} maxc_r(S)$.

Theorem 6.3 SIMPLE MAX CUT *can be solved in $O(n)$ time on graphs, given with a tree-decomposition of constant bounded treewidth.*

Again, it is possible to modify the algorithm, such that it also yields a partition with the maximum number of cut edges. Note that the result of this section is not new, but included here because it is generalized in the next section.

7 Composition of graphs

In this section, we give a result, that generalizes the results of the previous two sections.

Definition 7.1 *Let $H_0 = (V_0, E_0)$ be a graph with r vertices; $V_0 = \{v_1, v_2, \cdots, v_2\}$. Let $H_1 = (V_1, E_1)$, $H_2 = (V_2, E_2)$, $\ldots$, $H_r = (V_r, E_r)$ be r disjoint graphs. The composition graph $H_0[H_1, H_2, \cdots, H_r]$ is the graph, obtained by taking the disjoint union of H_1, H_2, $\ldots$, H_r, and adding all edges between pairs of vertices v, w, with $v \in V_i$, $w \in V_j$, and $(i,j) \in E_0$: $H_0[H_1, H_2, \cdots, H_r] = (\bigcup_{1 \leq i \leq r} V_i, \bigcup_{1 \leq i \leq r} E_i \cup \{(v,w) \mid \exists i,j : \ 1 \leq i,j \leq r,\ v \in V_i, w \in V_j,\ (i,j) \in E_0\})$.*

It often is useful to try to write a graph $G = (V, E)$ as a composition graph $G = H_0[H_1, H_2, \cdots, H_r]$, for some suitable choice of $H_0, \ldots, H_r$. Such a composition, where H_0 is as small as possible, $r \geq 2$, can be found in polynomial time [12]. (Clearly, a trivial composition, where $G = H_0$ and all graphs $H_1, \cdots, H_n$ consist of one vertex allways exists, but is not really usefull.) Then, it is often useful to decompose the graphs H_1, H_2, $\ldots$, H_r again, and then possibly decompose the formed parts of these graphs again, etc.

In this way, one can associate with a graph a decomposition tree. (Note that this is an entirely different notion as the notion of tree-decomposition!) A decomposition tree is a rooted tree, where every non-leaf node is labeled with a graph. We call this graph a *label graph.* The number of vertices in a label graph equals the number of children of the node to which the graph is labeled; these vertices are always numbered 1,2,... To each node of the decomposition tree, one can associate then a graph, called the *factor graph*, in the following way. To a leaf node, associate a graph with one vertex and no edges. To a non-leaf node, with label graph $H_0 = (\{1, 2, \cdots, r\}, E_0)$, associate the graph $H_0[H_1, \cdots, H_r]$, where for all i, $1 \leq i \leq r$, H_i is the factor graph associated to the i'th child of the node. The factor graph associated to the root of the tree is the graph, represented by this decomposition tree.

The notion of decomposition tree generalizes the notion of cotree: in a cotree the only label graphs are K_2 (a graph with two vertices and one edge — the label of product nodes), and K_2^c (a graph with two vertices and no edges — the label of union nodes).

The following result generalizes the results of the previous two sections.

Theorem 7.2 *For all constants k, the* SIMPLE MAX CUT *problem is solvable in polynomial time for graphs, with a decomposition tree, where every label tree has treewidth at most k.*

The first step of the algorithm is to find the decomposition tree. By using the results from [12], it follows that the decomposition tree can be found in polynomial time, such that the size and also the treewidth of label graphs are minimal. Also, a tree-decomposition of treewidth at most k of the type as described in the previous section is computed for every label graph.

For each factor graph $H = (V', E')$, associated with a node of the decomposition tree, we compute — just as we did for cographs — a table $maxc_H$, which has entries for all integers i, $0 \leq i \leq |V'|$, that denote the maximum size of a cut of H into a set of size i and a set of size $|V'| - i$, in other words:

$$maxc_H(i) = \max\{|\{(v, w) \mid v \in W_1,\ w \in W_2\}| \mid W_1 \cup W_2 = V',\ W_1 \cap W_2 = \emptyset,\ |W_1| = i\}$$

These tables are easily computed for factor graphs, associated with leaves. Again, the tables are computed bottom up in the decomposition tree.

Suppose we want to compute the table for a factor graph $H = H_0[H_1, \cdots, H_r]$, ($H_0 = (\{1, 2, \cdots, r\}, E_0)$ is the label graph of some non-leaf node of the decomposition tree, and $H_1 = (V_1, E_1)$, ..., $H_r = (V_r, E_r)$ are the factor graphs, associated with the children of that node.) We have already computed all tables $maxc_{H_1}$, ..., $maxc_{H_r}$.

As in the previous section, for every node $\alpha \in I$, let Y_α denote the set of all vertices in a set X_β with $\beta = \alpha$ or β is a descendant of α in the rooted tree T.

Let $H_\alpha = (Z_\alpha, F_\alpha)$ denote the graph, obtained by removing all vertices from H, that are not in a graph H_i, with $i \in Y_\alpha$, or in other words, H_α is the subgraph of H, induced by all vertices in $Z_\alpha = \bigcup_{i \in Y_\alpha} V_i$.

In order to compute the table $maxc_H$, we compute now for every node $\alpha \in I$ of the tree-decomposition $(\{H_\alpha \mid \alpha \in I\}, T = (I, F))$ of label graph H_0 a table $maxc'_\alpha$, which has an entry for every funtion $f : X_\alpha \to \{0, 1, 2, \ldots\}$, such that $f(i) \leq |V_i|$, where for all such funtions f:

> $maxc'_\alpha(f, r)$ denotes the maximum cut size of a partition of Z_α into two disjoint sets W_1, W_2, such that for all $i \in X_\alpha$, $|W_1 \cap V_i| = f(i)$, and $|W_1| = r$.

In other words, we look for the maximum cut of H_α, such that f describes for all graphs H_i with i an element of the set X_α, how many vertices of H_i are in the set W_1.

We compute the tables $maxc'_\alpha$ bottom up. (Note that we work with two types of trees: we have one decomposition tree, and with every node of the decomposition tree, we have associated a tree-decomposition.)

Lemma 7.3 *(i) Let α be a leaf in T. Then for all $f : X_\alpha \to \{0, 1, 2, \ldots\}$, with for all $i \in X_\alpha : f(i) \leq |V_i|$, $r = \sum_{i \in X_\alpha} f(i)$: $maxc'_\alpha(f, r) = \sum_{i \in X_\alpha} maxc_{H_i}(f(i)) + \sum_{(i,j) \in E_0,\ i,j \in X_\alpha} (f(i) \cdot (|V_j| - f(j)) + f(j) \cdot (|V_i| - f(i)))$. For all other values of r, $maxc'_\alpha(f, r) = -\infty$.*
(ii) Let α be a node with one child β in T. Suppose $X_\alpha \subseteq X_\beta$. Then for all $r \geq 0$, $f : X_\alpha \to \{0, 1, 2, \ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$: $maxc'_\alpha(f, r) = \max\{maxc'_\beta(f', r) \mid \forall i \in X_\alpha : f(i) = f'(i) \vee \forall i \in X_\beta : f'(i) \leq |V_i|\}$.
(iii) Let α be a node with one child β in T. Suppose $X_\beta \cup \{i_0\} = X_\alpha$, $i_0 \notin X_\beta$. Then for all $r \geq 0$, $f : X_\alpha \to \{0, 1, 2, \ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$: let f' be the function f restricted to X_β. Then $maxc'_\alpha(f, r) = maxc'_\beta(f, r - f(i_0)) + \sum_{(i_0,j) \in E_0, j \in X_\alpha} (f(i_0) \cdot (|V_j| - f(j)) + (|V_{i_0}| - f(i_0)) \cdot f(j))$.
(iv) Let α be a node with two childs β_1, β_2 in T, with $X_\alpha = X_{\beta_1} = X_{\beta_2}$. Then for all $r \geq 0$, $f : X_\alpha \to \{0, 1, 2, \ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$: $maxc'_\alpha(f, r) = \max_{r_1, r_2 \geq 0,\ r_1 + r_2 - \sum_{i \in X_\alpha} f(i) = r} maxc'_{\beta_1}(f, r_1) + maxc_{\beta_2}(f, r_2) - \sum_{(i,j) \in E_0, i,j \in X_\alpha} (f(i) \cdot (|V_j| - f(j)) + (|V_i| - f(i)) \cdot f(j))$.

Proof:
(i) In each V_i, $i \in X_\alpha$, we must put $f(i)$ vertices in the first set of the partition.

In the subgraph, induced by a set V_i, the maximum number of cut edges is then $maxc_{H_i}(f(i))$. For each edge (i,j) with both i and j in X_α, there are $f(i) \cdot (|V_j| - f(j)) + (|V_i| - f(i)) \cdot f(j)$ cut edges.

(ii) Note that $H_\alpha = H_\beta$. We take the maximum over all possible ways to extend the function f to the domain X_β.

(iii) Consider a partition of Z_α into two disjoint sets of W_1, W_2, with for all $i \in X_\alpha$, $|W_1 \cap V_i| = f(i)$, and $|W_1| = r$. $W_1' = W_1 \cap Z_\beta$, $W_2' = W_2 \cap Z_\beta$ is a partition of Z_β, with for all $i \in X_\beta$, $|W_1' \cap V_i| = f(i)$, and $|W_1'| = r - f(i_0)$. For every edge $(i_0, j) \in E_0$, $j \in X_\alpha$, there are exactly $f(i_0) \cdot (|V_j| - f(j)) + (|V_{i_0}| - f(i_0)) \cdot f(j)$ cut edges between vertices in V_{i_0} and V_j.

(iv) Consider a partition of Z_α into two disjoint sets of W_1, W_2, with for all $i \in X_\alpha$, $|W_1 \cap V_i| = f(i)$, and $|W_1| = r$. $W_1^\gamma = W_1 \cap Z_{\beta_\gamma}$, $W_2^\gamma = W_2 \cap Z_{\beta_\gamma}$ is a partition of Z_{β_γ}, $\gamma \in \{1,2\}$. Note that $|W_1^1| + |W_1^2| = r - \sum_{i \in X_\alpha}$. There are precisely $\sum_{(i,j) \in E_0, i,j \in X_\alpha} (f(i) \cdot (|V_j| - f(j)) + (|V_i| - f(i)) \cdot f(j))$ edges that are both a cut edge in the partition W_1^1, W_1^2 of Z_{β_1}, and a cut edge in the partition W_2^1, W_2^2 of Z_{β_2}. □

From lemma 7.3, it follows directly how all tables $maxc'$ can be computed in a bottom up manner, given all tables $maxc_{H_i}$. The time, needed per table is linear in the size of the table, plus the sizes of the tables of its children, hence is polynomial in n (but exponential in k.) When we have the table $maxc'_\gamma$ with gamma the root-node of the tree-decomposition, then we can compute the table $maxc_H$ (remember that $H = H_0[H_1, \cdots, H_r]$), using the following lemma.

Lemma 7.4 *For all $r \geq 0$, $r \leq |V_H|$, $maxc_H(r) = \max\{maxc'_\gamma(f,r) \mid \forall i \in X_\gamma : f(i) \leq |V_i|\}$.*

Proof:
Note that $H = H_\gamma$. We just take the maximum over all possible numbers of vertices in W_1 that are in each of the sets V_i with $i \in X_\gamma$. □

We now are one level higher in the decomposition tree. The processes are repeated until the table $maxc_G$ is obtained, from which the answer to the simple max cut problem can be determined. As each table computation can be done in polynomial time, and a linear number of tables must be computed, the whole algorithm takes time, polynomial in n, when k is a fixed constant. We now have proved theorem 7.2.

It is also possible to construct the partition which gives the maximum number of cut edges, without increasing the running time of the algorithm by more than a constant factor. This can be done similarly as in the previous two sections. We omit the details.

8 Weighted Max Cut

We conclude this paper with some small observations on the weigthed variant of the problem. First, observe that MAX CUT is NP-complete, when restricted to cliques, when only edge weigths 0 and 1 are allowed. (The problem in this form is equivalent to the SIMPLE MAX CUT problem.) So, MAX CUT is NP-complete for all classes of graphs that contain all cliques, (e.g., for the class of cographs.) Secondly, as first shown by Wimer [16], MAX CUT can be solved in linear time on graphs given with a treedecomposition of bounded treewidth. (It is possible to modify the results of section 6 and obtain an algorithm, quite similar to the algorithm of Wimer.)

References

[1] S. ARNBORG, Efficient algorithms for combinatorial problems on graphs with bounded decomposability — A survey, BIT **25** (1985), pp. 2 - 23.

[2] H.L. BODLAENDER, Classes of graphs with bounded treewidth, Technical Report RUU-CS-86-22, Dept. of Computer Science, Utrecht University, Utrecht, 1986.

[3] H.L. BODLAENDER AND T. KLOKS, Better algorithms for the pathwidth and treewidth of graphs, Proc. 18th Int. Coll. on Automata, Languages and Programming, Springer Verlag Lecture Notes on Comp. Science, vol. 510, (1991) pp. 544 - 555.

[4] K. CHANG AND D. DU, Efficient algorithms for the layer assignment problem, IEEE Trans. CAD **6** (1987), pp. 67 - 78.

[5] R. CHEN, Y. KAJITANI AND S. CHAN, A graph theoretic via minimization algorithm for two layer printed circuit boards, IEEE Trans. Circuit Syst. (1983), pp. 284 - 299.

[6] D.G. CORNEIL, Y. PERL, AND L.K. STEWART, A linear recognition algorithm for cographs, SIAM J. Comput. **4** (1985), pp. 926 - 934.

[7] S. FÖLDES AND P.L. HAMMER, Split graphs, Proc. 8th Southeastern Conf. on Combinatorics, Graph Theory and Computing, Louisiana State University, Baton Rouge, Louisiana, pp. 311 - 315.

[8] M.R. GAREY AND D.S. JOHNSON, Computers and Intractability: A Guide to the Theory of NP-Completness, Freeman, San Francisco, 1979.

[9] M.R. GAREY, D.S. JOHNSON AND L. STOCKMEYER, Some simplified NP-complete graph problems, Theo. Comput. Sci **1** (1976), pp. 237 - 267.

[10] D.S. JOHNSON, The NP-completeness column: an ongoing guide, J. Algorith. **6** (1985), pp. 434 - 451.

[11] R.M. KARP, Reducibility among combinatorial problems, in: Miller and Thatcher: Complexity of Computer Computations, Plenum Press (1972), pp. 85 - 104.

[12] J.H. MULLER AND J. SPINRAD, Incremental modular decomposition, J. ACM, **36** (1989), pp. 1 - 19.

[13] R. PINTER, Optimal layer assignment for interconnect, Proc. Int. Symp. Circuit Syst. (ISCAS) (1982), pp. 398 - 401.

[14] N. ROBERTSON AND P.D. SEYMOUR, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms **7** (1986), pp. 309-322.

[15] W.T. TROTTER, JR. AND F. HARARY, On double and multiple interval graphs, J. Graph Theory **3** (1979), pp. 205 - 211.

[16] T.V. WIMER, Linear algorithms on k-terminal graphs, PhD thesis, Department of Computer Science, Clemson University, 1987.