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 [13, 10]. This motivates the research to solve the (SIMPLE) MAX CUT problem on special classes of graphs.

In [12] 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 4 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 the references [5, 6, 15].

*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

This paper is organized as follows. In section 2 we consider the chordal graphs, and the undirected path graphs. Next, we consider the split graphs, tripartite graphs, and complements of bipartite graphs. In section 3.1, we consider cographs. An algorithm to solve SIMPLE MAX CUT on graphs with bounded treewidth is described in section 3.2. In section 3.3, the results of sections 3.1 and 3.2 are generalized. Finally, in section 4 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 NP-completeness results

In this section we analysed the SIMPLE MAX CUT problem for chordal graphs, split graphs, tripartite graphs, and complements of bipartite graphs.

2.1 Graphs, related to 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 [10]. (In [10] also a transformation from MAX 2-SAT to the SIMPLE MAX CUT problem for undirected graphs was given.) We note [9] 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 [10] 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) occurs 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 variable set, 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:

(1) for each $i \in \{1, \ldots, n\}$ take $C^{(i)} = \{c_1^{(i)}, \ldots, c_{m+1}^{(i)}\}$, $D^{(i)} = \{d_1^{(i)}, \ldots, d_{m+2}^{(i)}\}$, $E^{(i)} = \{e_1^{(i)}, \ldots, e_{m+2}^{(i)}\}$.

(2) take for each $i \in \{1, \ldots, p\}$, the set $T^{(i)} = \{t_1^{(i)}, \ldots, t_{m+2}^{(i)}\}$, and take the sets $R = \{r_1, \ldots, r_p\}$, $Q = \{q_1, \ldots, q_p\}$, $Y = \{y_1, \ldots, y_p\}$ and $S = \{s_1, \ldots, s_{m+1}\}$.

(3) take $U = \{u_1, \ldots, u_p\}$, $V = \{v_1, \ldots, v_p\}$, $W = \{w_1, \ldots, w_p\}$, and $Z = \{z_1, \ldots, z_p\}$.

In the following we define an input graph $G' = (V', E')$ and want to partition the vertex set V' into sets V_1 and V_2 where V_1 gets literals with truth value *true* and V_2 gets literals with value *false*. The sets in (1) are used to place x_i into V_1 or V_2 and the complement $\overline{x_i}$ into the other set. Furthermore, the sets in (2) place R and Q into the first set V_1 which contains the literals with value *false*. For all values to a_i, b_i with $(a_i \vee b_i) = true$ we want to have the same number of generated cut edges and for $(a_i \vee b_i) = false$ a smaller number of cut edges. To obtain this we use the sets in (3) and the set $Q = \{q_1, \ldots, q_p\}$.

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. (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 $B = 2n \cdot (m+2)^2 + n \cdot (m+1) + (n+p)^2 + p \cdot (m+1) + 2p \cdot (m+2) + 6p$. 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.

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:

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

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 can 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 assignment 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. For the details, we refer to our full paper. NP-hardness of the problem follows, because G' can be constructed in polynomial time. $\square$

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 graph 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. Then, we replace each variable x_i by $x_{i,1}, \ldots, x_{i,3}$ and $\overline{x_i}$ by $\overline{x_{i,1}}, \ldots, \overline{x_{i,3}}$ and enlarge the sets $D^{(i)}$ and $E^{(i)}$ by two vertices. $\square$

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 [8]. We analyse now 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 2.3 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 $F = \{(v, w) \mid v, w \in V,\ v \neq w\} \cup \{(v, e) \mid v \in V,\ e \in E^c,\ v \text{ is an endpoint of edge } e\}$. 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 can show 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. $\square$

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 [17].

Lemma 2.4 *Each* 2*-split graph is a double interval graph.*

Hence, as consequence we get:

Corollary 2.5 SIMPLE MAX CUT *is NP-complete for double interval graphs.*

2.2 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 2.6 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

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

and replace the edge (c_i, c_j) by the graph $G_{\{i,j\}}$. Then, $\overline{V}$ is the union of vertex sets $V_{\{i,j\}}$ and the independent set U. The edge set $\overline{E}$ is given as union of the edge sets $E_{\{i,j\}}$ and the set $\{e = \{c, u\} \in E | c \in C, u \in U\}$. 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. □

Theorem 2.7 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 can show 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. □

3 Composition of graphs

In this section we show that SIMPLE MAX CUT can be solved efficiently on a class of graphs that includes the graphs with bounded treewidth and the cographs. Independently, Wanke [18] has found a polynomial time algorithm for a class of graphs that includes the cographs. We first show the ideas of the method on cographs and on graphs with bounded treewidth.

3.1 Cographs

Definition 3.1 *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 3.2 *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.*

Alternatively, a graph is a cograph, if it does not contain a path with four vertices P_4 as an induced subgraph. Many NP-complete problems are polynomial time solvable on cographs; there are only a few notable expections, e.g. achromatic number [2] and list coloring [11] are NP-complete for cographs.

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. 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 [7], 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. The tables can be computed efficiently, starting with the tables at the leaves, and computing tables in an order such that when we compute the table a node, then the tables of its children have already been computed. efficiently

The tables associated with leaf nodes are clearly 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. A more general result will be shown in section 5.3.

Lemma 3.3 *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) $maxc_{G_1\times G_2}(i) = \max\{maxc_{G_1}(j)+maxc_{G_2}(i-j)+j\cdot(|V_2|-(i-j))+(|V_1|-j)\cdot(i-j) \mid 0 \leq j \leq i,\ j \leq |V_1|,\ i-j \leq |V_2|\}$.

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. By standard arguments, the following result now can be derived:

Theorem 3.4 *There exists an $O(n^2)$ algorithm for* SIMPLE MAX CUT *on cographs.*

It is easy to modify this algorithm such that it also *yields a partition* with the maximum number of cut edges, and uses also $O(n^2)$ time.

3.2 Graphs with bounded treewidth

It is well know that the SIMPLE MAX CUT problem can be solved in linear time on graphs with bounded treewidth (see e.g. [19]). We sketch the method here, as it will be generalized hereafter. The notion of treewidth of a graph was introduced by Robertson and Seymour [16], and is equivalent to several other interesting graph theoretic notions, for instance the notion of partial k-trees (see e.g., [1, 4]).

Definition 3.5 *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 not difficult 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 $O(n)$ time [3].

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, which is easy to proof, shows how the tables can be computed efficiently:

Lemma 3.6 *(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 children j_1, j_2 in T, with $X_i = X_{j_1} = X_{j_2}$. For all $S \subseteq X_i$, $maxc_i(S) = maxc_{j_1}(S) + maxc_{j_2}(S) - |\{(v,w) \in E \mid v \in S,\ w \in X_i - S\}|$.

It follows 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 3.7 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.

3.3 Composition of graphs

We now generalize and combine the previous results in this section.

Definition 3.8 *Let $H_0 = (V_0, E_0)$ be a graph with r vertices; $V_0 = \{v_1, v_2, \cdots, v_r\}$. 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 factor 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 factor graph $G = H_0[H_1, H_2, \cdots, H_r]$, for some suitable choice of $H_0, \ldots, H_r$. Such a 'factorization', where H_0 is as small as possible, $r \geq 2$, can be found in polynomial time [14]. (Clearly, a trivial factorization, where $G = H_0$ and all graphs $H_1, \cdots, H_n$ consist of one vertex always exists, but is not really useful.) Then, it is often useful to factorize the graphs H_1, H_2, $\ldots$, H_r again, and then possibly factorize the formed parts of these graphs again, etc.

In this way, one can associate with a graph a factor tree. A factor 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 factor 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 factor tree.

The notion of factor 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 3.9 *For all constants k, the* SIMPLE MAX CUT *problem is solvable in polynomial time for graphs, with a factor tree, where every label graph has treewidth at most k.*

The first step of the algorithm is to find the factor tree. By using the results from [14], it follows that the factor 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 factor 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 factor 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 factor tree, and $H_1 = (V_1, E_1)$, $\ldots$, $H_r = (V_r, E_r)$ are the factor graphs, associated with the children of that node.) We have already computed all tables $maxc_{H_1}$, $\ldots$, $maxc_{H_r}$.

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.

For $\alpha \in I$, 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 function $f : X_\alpha \to \{0, 1, 2, \ldots\}$, such that $f(i) \leq |V_i|$, where for all such functions f:

> $maxc'_\alpha(f, s)$ 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 $|W_1| = s$.

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, in the tree-decomposition. The next lemma shows how this can be done. The proof of this result is given in the full paper. Note that we are working with two types of trees: we have one factor tree, and with every node of this factor tree, we have associated a tree-decomposition.

Lemma 3.10 *(i) Let α be a leaf in T. Then for all $f : X_\alpha \to \{0, 1, 2, \ldots\}$, with for all $i \in X_\alpha : f(i) \leq |V_i|$, $s = \sum_{i \in X_\alpha} f(i)$:*

$$maxc'_\alpha(f, s) = \sum_{i \in X_\alpha} maxc_{H_i}(f(i)) + \sum_{(i,j) \in E_0,\ i,j \in X_\alpha} (f(i) \cdot (|V_j| - f(j)) + f(j) \cdot (|V_i| - f(i)))$$

For all other values of s,

$$maxc'_\alpha(f, s) = -\infty$$

(ii) Let α be a node with one child β in T. Suppose $X_\alpha \subseteq X_\beta$. Then for all $s \geq 0$, $f : X_\alpha \to \{0, 1, 2, \ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$:

$$maxc'_\alpha(f, s) = \max\{maxc'_\beta(f', s) \mid \forall i \in X_\alpha : f(i) = f'(i) \wedge \forall i \in X_\beta : f'(i) \leq |V_i|\}$$

(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 $s \geq 0$, $f : X_\alpha \to \{0, 1, 2, \ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$: let f' be the function f restricted to X_β. Then

$$maxc'_\alpha(f, s) = maxc'_\beta(f, s - f(i_0)) + \sum_{(i_0,j) \in E_0, j \in X_\alpha} (f(i_0) \cdot (|V_j| - f(j)) + (|V_{i_0}| - f(i_0)) \cdot f(j))$$

(iv) Let α be a node with two children β_1, β_2 in T, with $X_\alpha = X_{\beta_1} = X_{\beta_2}$. Then for all $s \geq 0$, $f : X_\alpha \to \{0,1,2,\ldots\}$ with for all $i \in X_\alpha : f(i) \leq |V_i|$:

$$\begin{aligned} maxc'_\alpha(f,s) \quad = \quad & \max_{s_1,s_2 \geq 0,\ s_1+s_2-\sum_{i\in X_\alpha} f(i)=s} maxc'_{\beta_1}(f,s_1) + maxc'_{\beta_2}(f,s_2) \\ & - \sum_{(i,j)\in E_0, i,j\in X_\alpha} (f(i)\cdot(|V_j| - f(j)) + (|V_i| - f(i))\cdot f(j)) \end{aligned}$$

From lemma 3.10, 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 γ 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 3.11 *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 factor 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 3.9.

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.

4 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 [19], MAX CUT can be solved in linear time on graphs given with a tree-decomposition of bounded treewidth. (It is possible to modify the results of section 3.2 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, Achromatic Number is NP-complete for cographs and interval graphs. *Information Processing Letters*, 31:135–138, 1989.

[3] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth. Technical Report RUU-CS-92-27, Department of Computer Science, Utrecht University, Utrecht, the Netherlands, 1992.

[4] H.L. Bodlaender, A tourist guide through treewidth. Technical Report RUU-CS-92-12, Department of Computer Science, Utrecht University, Utrecht, 1992.

[5] K. Chang and D. Du, Efficient algorithms for the layer assignment problem, IEEE Trans. CAD **6** (1987), pp. 67 – 78.

[6] 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.

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

[8] 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.

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

[10] M.R. Garey, D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, Theo. Comput. Sci **1** (1976), pp. 237 – 267.

[11] K. Jansen and P. Scheffler, Some coloring results for tree like graphs, Workshop on Graph Theoretic Concepts in Computer Science, LNCS, 657, 1992, p. 50-59.

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

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

[14] J.H. Muller and J. Spinrad, Incremental modular decomposition, J. ACM, **36** (1989), pp. 1 – 19.

[15] R. Pinter, Optimal layer assignment for interconnect, Proc. Int. Symp. Circuit Syst. (ISCAS) (1982), pp. 398 – 401.

[16] N. Robertson and P.D. Seymour, Graph minors. II. Algorithmic aspects of tree-width, J. Algorithms **7** (1986), pp. 309-322.

[17] W.T. Trotter, Jr. and F. Harary, On double and multiple interval graphs, J. Graph Theory **3** (1979), pp. 205 – 211.

[18] E. Wanke, k-NLC graphs and polynomial algorithms. Bericht, Reihe Informatik 80, Universität Paderborn, 1991.

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