Mathematical Programming 29 (1984) 28–40
North-Holland

A POLYNOMIAL ALGORITHM FOR THE MAX-CUT PROBLEM ON GRAPHS WITHOUT LONG ODD CYCLES

Martin GRÖTSCHEL
Mathematical Institute, University of Augsburg, Augsburg, West Germany

George L. NEMHAUSER
School of OR & IE, Cornell University, Ithaca, New York, USA

Received 8 June 1982
Revised manuscript received 11 March 1983

Given a graph $G=[V, E]$ with positive edge weights, the max-cut problem is to find a cut in G such that the sum of the weights of the edges of this cut is as large as possible. Let $\mathcal{G}(K)$ be the class of graphs whose longest odd cycle is not longer than $2K+1$, where K is a nonnegative integer independent of the number n of nodes of G. We present an $O(n^{4K})$ algorithm for the max-cut problem for graphs in $\mathcal{G}(K)$. The algorithm is recursive and is based on some properties of longest and longest odd cycles of graphs.

Key words: Max-Cut Problem, Graphs without Long Odd Cycles, Polynomial Time Algorithms.

1. Introduction

The graphs we consider are finite, undirected and loopless, but they may have multiple edges. We denote a graph by $G=[V, E]$, where V is the node set and E the edge set of G. For $W \subseteq V$, $\emptyset \neq W \neq V$, the set of edges with one end node in W and the other in $V \setminus W$ is called a *cut* and is denoted by $\delta(W)$. Suppose $c_e > 0$ is the *weight* of $e \in E$. The problem of finding a cut $\delta(W^*)$ such that $c(\delta(W^*)) := \sum_{e \in \delta(W^*)} c_e$ is as large as possible is called the *max-cut problem.*

The max-cut problem is NP-complete for the class of all graphs, cf. Garey and Johnson [5]. It is also NP-complete for the class of graphs with nodes having degree at most three, and for the class of graphs having a node whose removal results in a planar graph in which all nodes have degree at most six, cf. Barahona [1, 2] and Yannakakis [12].

On the other hand, the max-cut problem is known to be solvable in polynomial time for planar graphs, cf. Hadlock [7], Orlova and Dorfman [9], and for weakly bipartite graphs, cf. Grötschel and Pulleyblank [6]. Moreover, the cardinality version, i.e. $c_e = 1$ for all $e \in E$, is solvable in polynomial time for graphs of fixed genus, cf. Barahona [3]. We will use the fact that the max-cut problem is trivial to solve for bipartite graphs. In particular, if $G=[V, E]$ is bipartite and V_1, V_2 is a 2-coloring of V, then $\delta(V_1) = \delta(V_2) = E$ is the max-cut of G.

This research was supported by National Science Foundation Grant ECS-8005350 to Cornell University.

Let $\mathcal{G}(K)$, K a nonnegative integer, denote the class of graphs whose longest odd cycle is of length not greater than $2K+1$. $\mathcal{G}(0)$ is the class of bipartite graphs. Connected components of the class $\mathcal{G}(1)\setminus\mathcal{G}(0)$ have been characterized by Hsu, Ikura and Nemhauser [8]. They also gave a polynomial-time algorithm for recognizing members of $\mathcal{G}(K)$ for any K that is independent of n, the number of nodes of the graph (also see Section 5).

Most combinatorial optimization problems on graphs can be solved efficiently for trees and many can be solved efficiently over $\mathcal{G}(0)$. Thus the absence of cycles (or odd cycles) seems to make graph optimization problems easy, and it is natural to ask the question if the absence of long cycles (or long odd cycles) also makes a graph optimization problem easy. For the maximum weight clique problem this is true since the cliques of a graph in $\mathcal{G}(K)$ have size at most $2K+1$ and thus, for fixed K, all such cliques can be enumerated in polynomial time. The maximum weight independent set problem for graphs in $\mathcal{G}(K)$ also can be solved in polynomial time for fixed K [8]. Here we describe an algorithm which, for any n-node graph $G\in\mathcal{G}(K)$, finds a maximum weight cut in $O(n^{4K})$ time. Thus, this algorithm is polynomial for every class $\mathcal{G}(K)$ with K fixed.

While results of this type are of practical interest only for small values of K, they are of theoretical significance in sharpening the delineation (if any) between the classes P and NP. Of course, it would be inefficient to develop results of this type problem-by-problem individually and so it would be very nice to have a general result of the type:

If a combinatorial optimization problem is easy for bipartite graphs, then it is also easy for $\mathcal{G}(K)$ for every fixed K. We don't, however, know whether this result is true; for example, we don't know if it is true for the minimum cardinality coloring problem of the vertices of a graph, which is a trivial problem for bipartite graphs. This study of the max-cut problem was undertaken, in part, to see if a general methodology could be developed for a large class of combinatorial optimization problems. While the design of our max-cut algorithm is based on an idea developed in [8], it requires deeper theorems on the intersections of longest cycles in a graph; these may be of interest in their own right. It still remains open whether the approach can be extended to a large class of problems.

The basic approach here and in [8] is to start with a graph $G\in\mathcal{G}(K)$. If G is bipartite, we solve the max-cut problem directly, otherwise we transform G to several new graphs whose longest cycles are shorter than the longest cycle of G. The transformations are designed in such a way that a maximum weight cut in G can be recovered from the maximum cuts in the new graphs. Continuing this process, we eventually end up with bipartite graphs, and hence, a maximum cut in the original graph G can be obtained inductively. Since we can guarantee that the overall number of new graphs created in the reduction steps is not too large, we obtain the desired polynomial time bound. A cut $\delta(W)$ is called an $[s, t]$-cut if $s\in W$ and $t\in V\setminus W$. We will need to consider the *max-$[s, t]$-cut problem*: Given a graph $G=[V, E]$, two distinct nodes $s, t\in V$ and edge weights $c_e>0$ for all $e\in E$, find an $[s, t]$-cut $\delta(W^*)$

such that $c(\delta(W^*))$ is as large as possible. Clearly, by solving the max-$[s, t]$-cut problem for every pair of distinct nodes $s, t \in V$ we can find a maximum weight cut in G.

On the other hand, if we have a max-cut algorithm we can solve the max-$[s, t]$-cut problem as follows. We add to G a new edge f linking s and t and call the new graph G_{st}. Then f is given a large weight (e.g. the sum of all edge weights plus one suffices). It is obvious that all maximum weight cuts in G_{st} must contain the new edge f. Hence, all maximum cuts in G_{st} are $[s, t]$-cuts, and therefore, if we remove the edge f from any maximum weight cut in G_{st} we obtain a maximum weight $[s, t]$-cut in G. This shows that the max-cut problem and the max-$[s, t]$-cut problem are polynomially equivalent.

2. Treating graphs that are not blocks

A graph $G = [V, E]$ is *connected*, if for every two distinct nodes $u, v \in V$, G contains a $[u, v]$-path, i.e. a path whose end nodes are u and v. The maximal connected subgraphs of G are the *components* of G. Every set $W \subseteq V$ with the property that the number of components of $G - W$ is larger than the number of components of G is called an *articulation set*. A connected graph with at least three nodes is called *2-connected* if it has no articulation set of size one, i.e. no articulation node. A node-induced subgraph of a graph G which has no articulation node and which is maximal with respect to this property is called a *block* of G. Clearly, each block of a connected graph with at least two nodes is either 2-connected (if it has three or more nodes) or consists of two nodes joined by an edge.

In this section we will describe how the max-cut problem for an arbitrary graph can be reduced to a sequence of $O(n)$ max-cut problems for blocks.

Let G be a disconnected graph with components C_i, $i = 1, \ldots, s$. If $\delta(W_i)$ are the maximum weight cuts for the components C_i, $i = 1, \ldots, s$, then it is easy to see that $\delta(W)$ is a maximum weight cut of G for $W := \bigcup_{i=1}^{s} W_i$. Hence it suffices to consider connected graphs.

The blocks and articulation nodes of a connected graph $G = [V, E]$ can be found in $O(m)$ time, see Tarjan [10].

(2.1) Lemma. *Let $G = [V, E]$ be a connected graph with edge weights $c_e > 0$ for all $e \in E$. Let $B_i = [V_i, E_i]$, $i = 1, \ldots, r$, be the blocks of G. Then there is a node set $W \subseteq V$ such that $\delta_G(W)$ is a maximum weight cut of G and $\delta_{B_i}(V_i \cap W)$ is a maximum weight cut of B_i for $i = 1, \ldots, r$.*

Proof. Since $E = \bigcup_{i=1}^{r} E_i$ and $\delta_G(W) \cap E_i = \delta_{B_i}(W \cap V_i)$, we have $\delta_G(W) = \bigcup_{i=1}^{r} (\delta_G(W) \cap E_i) = \bigcup_{i=1}^{r} \delta_{B_i}(W \cap V_i)$. This immediately implies the statement of the lemma. □

Lemma (2.1) and algorithm (2.2) given below show that the maximum weight cut of a graph is determined by the maximum weight cuts of its blocks.

(2.2) Algorithm. Suppose $G=[V, E]$ is a connected graph with edge weights $c_e>0$ for all $e\in E$, and $B_i=[V_i, E_i]$, $i=1,\ldots,r$, are the blocks of G. Let $\delta_{B_i}(W_i)$ be a maximum weight cut of B_i, $i=1,\ldots,r$. We determine a node set $W\subseteq V$ recursively such that $\delta_G(W)$ is a maximum weight cut of G. Initially all blocks of G are unlabeled.

Step 1. Pick any block, say B_1, of G, and set $W:=W_1$. Label B_1.

Step 2. If there is no unlabeled block of G left, STOP.

Step 3. Pick an unlabeled block B_i of G which has a node, say v, in common with a labeled block. If $v\notin W\cup W_i$ or $v\in W\cap W_i$, set $W:=W\cup W_i$. If ($v\in W$ and $v\notin W_i$ or ($v\notin W$ and $v\in W_i$), set $W:=W\cup(V_i\setminus W_i)$. Label B_i and go to Step 2. □

It is easy to see that the final node set W constructed in (2.2) determines a maximum weight cut $\delta(W)$ of G. Moreover, using well-known search techniques, algorithm (2.2) can be implemented in $O(|E|)$ time, provided that a maximum cut for every block is known. Thus algorithm (2.2) shows that in order to prove that the max-cut problem is solvable in polynomial time for a class $\mathcal{G}$ of graphs, it is sufficient to prove that the max-cut problem is solvable in polynomial time for the graphs in $\mathcal{G}$ which are 2-connected.

An alternative way to find a maximum weight cut in G (having the same time complexity as (2.2)) is to first construct the graph $G'=[V, \bigcup_{i=1}^{r}\delta_{B_i}(W_i)]$ which is bipartite and then to determine a bipartition of V.

3. $\{u, v\}$-Fragments and node set shrinking

We will now describe two techniques for reducing the max-cut problem on a graph G to a sequence of max-cut problems on related graphs.

Let G be a 2-connected graph that has an articulation set of size two and suppose $\{u, v\}$ is such an articulation set. Let $H_1=[V_1, E_1],\ldots,H_r=[V_r, E_r]$ be the components of $G-\{u, v\}$.

For $i=1, 2,\ldots,r$ let $H_i'=[V_i', E_i']$ be obtained from the subgraph of G induced by $V_i'=V_i\cup\{u, v\}$ where a new (possibly parallel) edge is added linking u and v. The new edge is assigned a large weight given by $\min\{c(\delta(u)), c(\delta(v))\}+1$. The graphs H_i' are called the *edge-added* $\{u, v\}$*-fragments* of G. Let $H_i''=[V_i'', E_i'']$ be the graph obtained from H_i' by identifying u and v into a node w and removing any loops that occur. The graphs H_i'' are called the *condensed* $\{u, v\}$*-fragments* of G.

The edge-added and the condensed $\{u, v\}$-fragments $H_1',\ldots,H_r'$, $H_1'',\ldots,H_r''$ are called the $\{u, v\}$*-fragments* of G. Figure 3.1(a) shows a graph G with nodes u, v, Fig. 3.1(b) the edge added $\{u, v\}$-fragments, and Fig. 3.1(c) the condensed $\{u, v\}$-fragments of G.

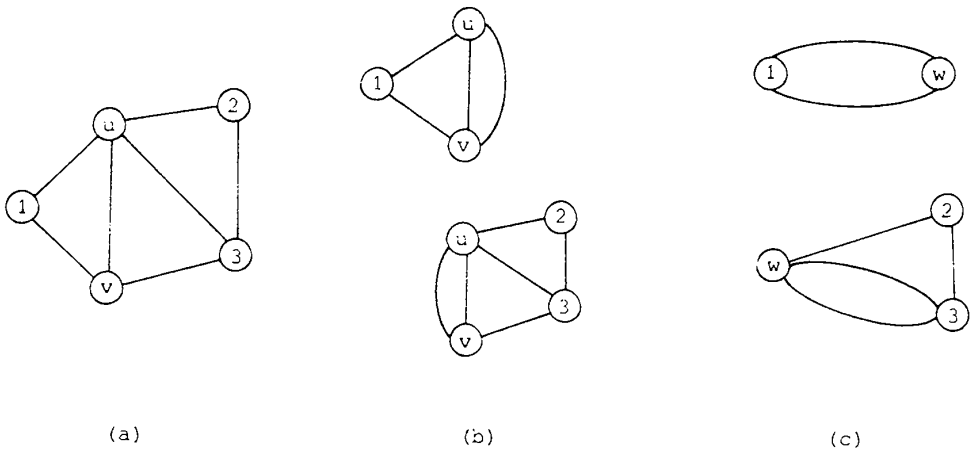

Fig. 3.1.

We claim that the max-cut problem for G can be solved by solving the max-cut problem for all condensed $\{u, v\}$-fragments and for all edge-added $\{u, v\}$-fragments.

Because $\delta(W) = \delta(V \setminus W)$, there is always a maximum weight cut $\delta(W)$ of G which either satisfies $u, v \in W$ or $u \in W$, $v \notin W$. Hence, if we can construct a set W'' with $u, v \in W''$ such that $c(\delta(W''))$ is as large as possible among the sets W with $u, v \in W$ and a set W' with $u \in W'$ and $v \notin W'$ such that $c(\delta(W'))$ is as large as possible among the sets W with $u \in W$, $v \notin W$, we can find a maximum weight cut of G by comparing the weights of these two cuts.

Suppose $\delta(W_i'')$ is a maximum weight cut of the condensed $\{u, v\}$-fragment H_i'' of G such that W_i'' contains the new node w, $i = 1, \ldots, r$. Set $W'' := \bigcup_{i=1}^{r} (W_i'' \setminus \{w\}) \cup \{u, v\}$. Then it is easy to see that among all cuts $\delta(W)$ of G with $u, v \in W$, $\delta(W'')$ is one with maximum weight.

Suppose $\delta(W_i')$ is a maximum weight cut of the edge-added $\{u, v\}$-fragment H_i' of G. Then because H_i' contains the new edge uv of large weight, $\delta(W_i')$ is a maximum weight $[u, v]$-cut. We can suppose that $u \in W_i'$, $i = 1, \ldots, r$. Again it is trivial to show that for $W' := \bigcup_{i=1}^{r} W_i'$, $\delta(W')$ is a cut in G whose weight is maximum among all cuts $\delta(W)$ with $u \in W$ and $v \notin W$.

Therefore by calculating $\delta(W')$ and $\delta(W'')$ from $\{u, v\}$-fragments of G and comparing their weights, we obtain a maximum weight cut of G.

A further (trivial) reduction method uses the shrinking of node sets. For the graph $G = [V, E]$ and node set $T \subseteq V$, the graph obtained from G by *shrinking* T is the graph obtained by replacing the node set T of G by a new node t and linking a node $v \in V \setminus T$ with t by as many edges as there are edges linking v to a node in T.

Suppose $G = [V, E]$ and a node set $S \subseteq V$ of cardinality $l \geq 2$ are given. We partition S into two nonempty sets S_1 and S_2 and denote the graph obtained from G by shrinking S_1 into a node s_1 and S_2 into a node s_2 by $G_{s_1 s_2}$. (Note that $G_{s_1 s_2}$ is independent of the order of shrinking S_1 and S_2.) Now consider all $2^{l-1} - 1$ possible partitions of S into two nonempty subsets S_1 and S_2, and suppose that $\delta(W_{s_1 s_2})$ is

a maximum weight cut of $G_{s_1 s_2}$ such that $s_1 \in W$. Suppose that $\delta(W_{t_1 t_2})$ is a cut among all these cuts with largest weight. Then $\delta(W)$ is a maximum weight cut of G, where if $t_1, t_2 \in W_{t_1 t_2}$, $W := (W_{t_1 t_2} \setminus \{t_1, t_2\}) \cup T_1 \cup T_2$, and if $t_1 \in W_{t_1 t_2}$, $t_2 \notin W_{t_1 t_2}$, $W := (W_{t_1 t_2} \setminus \{t_1\}) \cup T_1$.

The last reduction is, of course, useful only if l is small compared to n and if the shrunken graphs $G_{s_1 s_2}$ have some interesting properties. In our case, we shall shrink the node set of a longest cycle of G. Therefore we have to show that by shrinking such a node set we obtain graphs in which the max-cut problem is 'easier' than in G.

4. Properties of longest cycles

Our algorithm exploits several properties of the intersections of longest cycles that will be described in this section. Throughout this section we assume that G is a 2-connected graph with at least three nodes.

The following definitions are needed in this section. A *path* P in $G = [V, E]$ is a sequence of edges $e_1, e_2, \ldots, e_k$ such that $e_1 = v_0 v_1$, $e_2 = v_1 v_2, \ldots, e_k = v_{k-1} v_k$ and such that $v_i \neq v_j$ for $i \neq j$. The nodes v_0 and v_k are the *end nodes* of P and we say that P is a $[v_0, v_k]$-path. The nodes v_i, $i = 1, \ldots, k-1$ are the *internal nodes* of P.

The number k of edges of P is called the *length* of P. If $P = e_1, e_2, \ldots, e_k$ is a *$[v_0, v_k]$-path* and $e_{k+1} = v_0 v_k \in E$ then the sequence $e_1, e_2, \ldots, e_k, e_{k+1}$ is called a *cycle* of length $k+1$. A cycle (path) is called *odd* if its length is odd, otherwise it is called *even*. The length of a path or cycle C is denoted by $|C|$. The length of a longest cycle of a graph G is called the *circumference* of G. For convenience of notation we shall sometimes consider paths or cycles as subgraphs of a graph.

Two paths P_1, P_2 are called *internally disjoint* if the set of internal nodes of P_1 (resp. P_2) is disjoint from the set of nodes of P_2 (resp. P_1). If two $[u, v]$-paths P_1, P_2 are internally disjoint then $P_1 \cup P_2$ denotes the cycle obtained by concatenating P_1 and P_2.

Two nodes u, v are called *adjacent on a cycle* C if uv is an edge of C. Note that u and v may be adjacent in G without being adjacent on C.

The following proposition is obvious.

(4.1) Proposition. (a) *Every pair of longest cycles of G meets in at least two nodes.*

(b) *If two longest cycles of G meet in exactly two nodes, then these two nodes are not adjacent on either of the two cycles.* □

(4.2) Theorem. *Let C_1 and C_2 be two longest cycles of G whose intersection is the set $\{u, v\}$. Suppose $C_1 = P_1 \cup Q_1$ and $C_2 = P_2 \cup Q_2$, where P_1, P_2, Q_1, Q_2 are internally disjoint $[u, v]$-paths. Then*

(a) *paths P_1, P_2, Q_1, Q_2 have the same length (which implies that $|C_1| = |C_2|$ is even).*

(b) *$\{u, v\}$ is an articulation set of G and every truncated path $\bar{P}_1$, $\bar{P}_2$, $\bar{Q}_1$, $\bar{Q}_2$ obtained from P_1, P_2, Q_1, Q_2 by removing the two endpoints u and v belongs to a different component of $G-\{u, v\}$.*

Proof. Note first that because of (4.1)(b) all paths P_1, P_2, Q_1, Q_2 have length at least two.

(a) Clearly, $P_1 \cup Q_2$ and $P_2 \cup Q_1$ are cycles of G which by assumption cannot be longer than $|C_1| = |C_2|$. This means $|P_1 \cup Q_2| \leq |C_1| = |P_1 \cup Q_1|$ and hence $|Q_2| \leq |Q_1|$; similarly $|P_2 \cup Q_1| \leq |C_2| = |P_2 \cup Q_2|$ and hence $|Q_1| \leq |Q_2|$, which proves $|Q_1| = |Q_2|$. By symmetry we get $|P_1| = |P_2|$.

Now $P_1 \cup P_2$ and $Q_1 \cup Q_2$ are also cycles which are not longer than $|C_1| = |C_2|$, i.e. $|P_1 \cup P_2| \leq |C_1| = |P_1 \cup Q_1|$, which implies $|P_2| \leq |Q_1|$, and $|Q_1 \cup Q_2| \leq |C_2| = |P_2 \cup Q_2|$, which implies $|Q_1| \leq |P_2|$. Thus we get $|P_1| = |P_2| = |Q_1| = |Q_2|$.

(b) Suppose not all of the truncated paths $\bar{P}_1$, $\bar{P}_2$, $\bar{Q}_1$, $\bar{Q}_2$ are in different components of $G-\{u, v\}$. Without loss of generality, we may assume that $\bar{P}_1$ and $\bar{Q}_1$ are in the same component of $G-\{u, v\}$ and that there is a node x on $\bar{P}_1$, a node y on $\bar{Q}_1$ and an $[x, y]$-path R in $G-\{u, v\}$, which, except for x and y, has no other node in common with $\bar{P}_1$, $\bar{P}_2$, $\bar{Q}_1$, $\bar{Q}_2$.

In G, node x splits P_1 into a $[u, x]$-path P_1' and an $[x, v]$-path P_1'', and node y splits Q_1 into a $[u, y]$-path Q_1' and a $[y, v]$-path Q_1''; see Figure 4.1. Note that

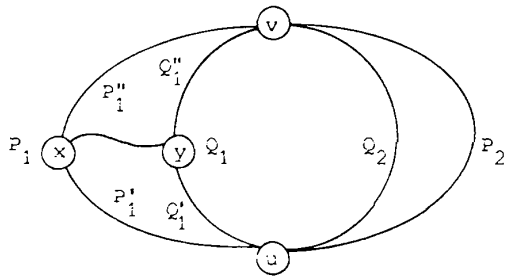

Fig. 4.1.

$Q_2 \cup P_1' \cup R \cup Q_1''$ and $Q_2 \cup Q_1' \cup R \cup P_1''$ are cycles in G not longer than $|C_1| = |C_2| = |Q_2 \cup P_1|$. Hence

$$|Q_2 \cup P_1' \cup R \cup Q_1''| \leq |Q_2 \cup P_1' \cup P_1''| \quad \text{implies} \quad |R \cup Q_1''| \leq |P_1''|,$$

and

$$|Q_2 \cup Q_1' \cup R \cup P_1''| \leq |Q_2 \cup P_1' \cup P_1''| \quad \text{implies} \quad |Q_1' \cup R| \leq |P_1'|.$$

This gives

$$|Q_1| < 2|R| + |Q_1'| + |Q_1''| \leq |P_1'| + |P_1''| = |P_1| = |Q_1|,$$

which is a contradiction. □

(4.3) Theorem. *Suppose two longest cycles C_1 and C_2 of G meet in exactly two nodes, say u and v.*

(a) *Then all longest cycles of G contain u and v, but not the edge uv.*

(b) *For every longest cycle C of G, the two pieces of $C-\{u, v\}$ in $G-\{u, v\}$ belong to different components of $G-\{u, v\}$.*

Proof. (a) We may assume as in (4.2) that $C_1 = P_1 \cup Q_1$ and $C_2 = P_2 \cup Q_2$ and that $\bar{P}_1, \bar{P}_2, \bar{Q}_1, \bar{Q}_2$ are the four truncated paths in $G-\{u, v\}$, which by (4.2)(b) are in different components of $G-\{u, v\}$.

Suppose C is a longest cycle of G not containing $\{u, v\}$, say u is not on C. This implies that $C-\{u, v\}$ is either a path or a cycle in $G-\{u, v\}$, and hence $C-\{u, v\}$ belongs to one component of $G-\{u, v\}$.

By (4.1)(a) C and C_1 meet in at least two nodes. Since u is not on C, there is a node on $C_1-\{u, v\}$, say on $\bar{P}_1$, that is also on $C-\{u, v\}$. Similarly, there is a node on $C_2-\{u, v\}$, say on $\bar{P}_2$, which is also on $C-\{u, v\}$. Since $C-\{u, v\}$ is connected in $G-\{u, v\}$, this implies that $\bar{P}_1$ and $\bar{P}_2$ are in the same component of $G-\{u, v\}$, contradicting (4.2)(b).

If C meets C_1 or C_2 only at u and v, then by (4.1)(b), u and v are not adjacent on C. Thus if uv is an edge of C so that $C-\{u, v\}$ belongs to one component of $G-\{u, v\}$, then $C-\{u, v\}$ intersects $C_1-\{u, v\}$ and $C_2-\{u, v\}$, which we have just shown to be impossible.

(b) Let $C = P \cup Q$ be a longest cycle of G, where P and Q are $[u, v]$-paths. Let $\bar{P}$ and $\bar{Q}$ be the paths in $G-\{u, v\}$ obtained by truncating P and Q. Then two of the paths $\bar{P}_1, \bar{P}_2, \bar{Q}_1, \bar{Q}_2$, say $\bar{P}_i$ and $\bar{Q}_j$, are in different components of $G-\{u, v\}$ then $\bar{P}$ and $\bar{Q}$. This implies that $P_i \cup P$ and $Q_j \cup Q$ are cycles in G. Moreover, they are longest cycles having exactly two nodes in common, namely u and v. Hence by (4.2)(b), $\bar{P}$ and $\bar{Q}$ are in different components of $G-\{u, v\}$. □

For our purposes, the following corollary bounding the circumference of the $\{u, v\}$-fragments of G (cf. Section 3) is of particular importance.

(4.4) Corollary. *Let u and v be two nodes that form the intersection of two longest cycles of G. Then the circumference of each of the $\{u, v\}$-fragments $H'_1, \ldots, H'_r$, $H''_1, \ldots, H''_r$ of G is smaller than the circumference of G, and all $\{u, v\}$-fragments are blocks.*

Proof. Immediate from the construction and Theorem (4.3). □

5. Finding a longest cycle

For every graph constructed in our algorithm, we must know a longest cycle explicitly. The algorithm showing that such a cycle can be found in polynomial time is based on a result of Voss.

(5.1) Theorem [11]. *Let G be a 2-connected, nonbipartite graph whose longest odd cycle has length l_0. Then the circumference of G is at most $2(l_0-1)$.* □

Let $\mathcal{H}(l)$ denote the class of graphs whose circumference is at most l, then Theorem (5.1) implies that every graph $G \in \mathcal{G}(K)$ is contained in $\mathcal{H}(4K)$. Therefore we can use brute force to find a longest cycle in G.

Our algorithm works as follows. Suppose a graph $G=[V, E] \in \mathcal{H}(l)$ is given. We first use depth-first search to check whether G is a forest. If G is a forest, we stop. Otherwise, let C be a cycle of G found by depth-first search. Set $p := |C|$.

Next we construct a sequence $W_1, W_2, \ldots, W_t$, $t=\binom{n}{l}$, containing all l-element subsets of V.

In the general step, we pick a set W_i not yet examined and for every s, $p \leq s \leq l$, we generate every sequence $w_1, w_2, \ldots, w_s$ of s different nodes of W_i. Then we check whether this sequence forms a cycle in G.

If it forms a cycle, say D, then we proceed as follows. In case $s=p$ we check whether C and D have exactly two nodes in common. If this is the case, we label the two nodes on C also contained in D. If $s>p$, then D is longer than C and we set $p := s$, and $C := D$, and continue.

Note that the algorithm either shows that G is a forest, or if not, produces a longest cycle C. Moreover, if G contains a cycle D with $|C|=|D|$ and C and D meet in exactly two nodes, then these two nodes are labeled (but D is not recorded). By Theorem (4.3)(a) all longest cycles of G contain the two labeled nodes.

The enumeration of all $\binom{n}{l}$ l-element subsets of V can be done in $O(n^l)$ time. For l fixed, the time needed for the enumeration of all sequences of nodes $w_1, \ldots, w_s$ in W_i is independent of n. Thus, the overall running time of our enumeration algorithm to find a longest cycle in a graph $G \in \mathcal{H}(l)$ is $O(n^l)$. For $G \in \mathcal{G}(K)$ we therefore need $O(n^{4K})$ time to construct a longest cycle.

6. The algorithm

In the preceding sections we have described all aspects of the algorithm and all the results on which it is based. We will now put the pieces together and estimate the complexity of the procedure.

The algorithm works inductively. Either the present graph G is bipartite and we can solve the max-cut problem trivially, or we construct several new graphs $G_1, G_2, \ldots, G_r$ from G. G is called the *father* of the new graphs G_i, the graphs G_i are the *sons* of G, and the G_i's are called *brothers*. The sons are created in such a way that a maximum weight cut in G can be read from the maximum weight cuts in the graphs G_i.

(6.1) The input to our algorithm is a graph $G=[V, E]$ with n nodes, m edges and with edge weights $c_e > 0$ for all $e \in E$, and a positive integer K. ($K=0$ is the bipartite case which is trivial.)

(6.2) We first use depth-first search to find all components of G and all blocks of the components of G. (This can be done in $O(m)$ time. We have shown in Section 2 how a maximum weight cut of G can be recovered from the maximum weight cuts of its components and blocks.)

(6.3) We use the algorithm described by Hsu, Ikrua and Nemhauser [8] to check whether the blocks of the components of G are members of $\mathcal{G}(K)$. If one of the blocks is not in $\mathcal{G}(K)$, we stop and report that G has an odd cycle longer than $2K+1$. (This procedure requires $O(n^{2K+3})$ time, cf. [8].)

(6.4) We now open a list $\mathcal{L}$ of unscanned graphs where each graph has a label l stating an upper bound on the circumference of the graph. The first members of the list $\mathcal{L}$ are the blocks of the components of the original graph G with a label $l := 4K$. (Note that because of (6.3) every block is in $\mathcal{G}(K)$ and hence, by Theorem (5.1), is contained in $\mathcal{H}(4K)$.)

(6.5) If the list $\mathcal{L}$ of unscanned graphs is empty, we go to step (6.12).

(6.6) We pick an unscanned graph, say $G=[V, E]$, from $\mathcal{L}$ with a label l and remove it from $\mathcal{L}$. (By construction every graph on the list is connected.)

(6.7) First we check whether G is bipartite. If G is bipartite, we put G on a list $\mathcal{L}'$ of scanned graphs recording a set $W \subseteq V$ with $\delta(W)=E$ determining the maximum weight cut. (This step requires $O(m)$ time.) Then we go to step (6.5).

(6.8) Now we check whether G is 2-connected. If G is not, we determine the blocks of G, put the blocks on the list $\mathcal{L}$ with a label l and go to step (6.5). (This step has complexity $O(m)$.)

(6.9) We run the $O(n^l)$ enumeration procedure described in Section 5 to find a longest cycle C of length $p \leq l$ in G. We must consider two cases.

(6.10) If C contains no labeled nodes (cf. Section 5), let S be the set of nodes of C. We partition S in all possible $2^{p-1}-1$ ways into two nonempty subsets S_1 and S_2, shrink S_1 into s_1, S_2 into s_2 and put all the graphs $G_{s_1 s_2}$ produced this way on the list $\mathcal{L}$, cf. Section 3. All graphs $G_{s_1 s_2}$ receive the label $p-1$ and we go to step (6.5). (Note that in this case each longest cycle, say D, of G has at least three nodes in common with C. Thus in every partition S_1, S_2 of S one of these two sets must contain two nodes, say u, v, of D. By shrinking S_1 and S_2 the nodes u and v are identified. Hence all longest cycles of G are destroyed by the shrinking procedure and therefore $G_{s_1 s_2}$ belongs to $\mathcal{H}(p-1)$. Since l (and hence p) is independent of n, step (6.10) requires only $O(m)$ time.)

(6.11) If C contains two labeled nodes, say u and v, then we construct all the $\{u, v\}$-fragments of G, cf. Section 3, put these on the list $\mathcal{L}$ with a label $p-1$ and go to step (6.5). (The label $p-1$ can be given because of corollary (4.4). The complexity of this step is $O(n)$.)

(6.12) Using the list $\mathcal{L}'$ of scanned graphs, we recover the maximum weight cut of the original graph G by determining inductively the maximum weight cut of a father from the maximum weight cuts of its sons. (These methods have been described in the previous sections.) □

To prove that the algorithm produces a maximum weight cut of the original graph and to estimate its complexity we may assume that the original graph is connected. Then there are three ways of creating sons of a father G having label l.

(6.8) Splitting G into its blocks which obtain label l.

(6.10) Shrinking the node set of a longest cycle in various ways into graphs all of which obtain the label $p-1$, where $p \leq l$.

(6.11) Constructing the $\{u, v\}$-fragments of G with label $p-1$, $p \leq l$.

Note that in steps (6.10) and (6.11) the sons created have a circumference that is strictly less than the circumference of the father. Only if G is not a block, can its sons have the same circumference as G. However, since in this case the sons are blocks, the sons of the sons (the grandsons) of G are either bipartite or have a smaller circumference than G. This shows that all grandsons of a father G have a circumference that is strictly smaller than G (or they are bipartite). Since our original graph G has label $l = 4K$, we can conclude that after at most $8K$ generations all sons will have circumference zero.

The graphs of circumference zero are the forests. These are bipartite and thus the maximum weight cut can be obtained easily. Hence after at most $8K$ generations all sons are bipartite and we can construct a maximum weight cut of the original graph by backward induction. This shows that our algorithm works.

The application of step (6.9) to the original graph (resp. its blocks) gives a time bound of $O(n^{4K})$. We claim that for $K \geq 2$ this time bound is the overall running time of the whole algorithm. We prove this claim inductively on the label l by showing that if G is a graph with label l then it takes at most $d(l)n^l$ time to find a maximum weight cut of G.

Note that the labels 0 and 1 never occur, so suppose G is a graph with label $l = 2$. Then we recognize in step (6.7) that G is bipartite. In this case a maximum weight cut of G can be found in $O(n^2)$ time, so that our claim is true for $l = 2$.

Now assume that the running time of our algorithm for a graph with r nodes and label $2 \leq k < l$ is at most $d(k)r^k$. Consider a graph G with n nodes and label l constructed by the algorithm.

If G is a bipartite graph we are done. So suppose G is nonbipartite and that the algorithm constructs t sons G_i with n_i nodes and label $l_i \leq l$, $i = 1, \ldots, t$. During the generation of the sons of G some of the steps (6.6), ..., (6.11) are executed. Checking bipartiteness, finding the blocks and recovering the maximum cut of G from the maximum cuts of its sons takes at most $d'(l)n^2$ time altogether. The most costly step is (6.9) where at most $d''(l)n^l$ time is needed.

We now use our induction hypothesis to estimate the total amount of work needed to find the maximum weight cuts of all sons of G. Recall that sons are created in either step (6.8) or (6.10) or (6.11).

Suppose the sons are created in step (6.10). Here we generate at most 2^{l-1} sons where each son has fewer than n vertices and a label $l_i < l$. Thus, by the induction hypothesis, to obtain a maximum cut for one of these sons the time needed is at most $d(l_i)n_i^{l_i}$. Hence to obtain a maximum cut for G, an upper bound on the running

time is

$$\sum_{i=1}^{t} d(l_i)n_i^l + d'(l)n^2 + d''(l)n^l \leq 2^{l-1}d(l-1)(n-1)^{l-1} + (d'(l)+d''(l))n^l$$
$$\leq (2^{l-1}d(l-1) + d'(l) + d''(l))n^l.$$

Suppose the sons are created in step (6.11). Here we generate at most $2n$ sons each on fewer than n nodes and each having a label less than l. Thus to obtain a maximum cut for G, the time needed is at most

$$2nd(l-1)(n-1)^{l-1} + (d'(l)+d''(l))n^l \leq (2d(l-1) + d'(l) + d''(l))n^l.$$

From the above analysis we can conclude that if G is a block with label l then at most $d^*(l)n^l$ operations are required to find a max cut of G, where

$$d^*(l) = 2^{l-1}d(l-1) + d'(l) + d''(l).$$

Finally we consider the case where G is not a block and step (6.8) is applied. In this case G is split into t blocks G_i with $n_i < n$ nodes and label $l_i = l$. We have that $\sum_{i=1}^{t} n_i < 2n$. Thus the work required to find a max cut of G is at most

$$\sum_{i=1}^{t} d^*(l)n_i^l + d'(l)n^2 \leq d^*(l) \sum_{i=1}^{t} n_i^l + d'(l)n^2 \leq d^*(l)\left(\sum_{i=1}^{t} n_i\right)^l + d'(l)n^2$$
$$\leq d^*(l)2^l n^l + d'(l)n^2 \leq (2^l d^*(l) + d'(l))n^l.$$

Setting

$$d(l) = 2^l d^*(l) + d'(l)$$

completes the induction.

In addition to counting the number of steps of the algorithm, we also must establish that the numbers involved in the computations do not grow too large. Recall that the only part of the algorithm in which the data may be modified is in step (6.11) where we may at most double the sum over all edge capacities. This does not affect the complexity bound of the algorithm.

Hence we have proved that the running time of the complete algorithm is $O(n^{4K})$ ($4K$ is the largest label) provided that $K \geq 2$. If $K = 1$, then the membership test in (6.3) is more costly than the max-cut algorithm and we get a time bound of $O(n^5)$.

(6.13) Theorem. *There is an $O(n^{4K})$ algorithm which finds a maximum weight cut for every graph $G \in \mathcal{G}(K)$, $K \geq 2$, with n nodes.* □

7. Conclusions

There are several ways to improve the algorithm described in Section 6. For instance instead of shrinking and splitting until a bipartite graph is reached, we can

stop the generation of sons whenever we have found a planar or a weakly bipartite graph since for these graphs polynomial time algorithms are known.

Moreover, we have also devised a method in which by splitting and shrinking, sons of a father are created such that the longest *odd* cycle of every son is shorter than the longest odd cycle of its father. This algorithm has a time bound of $O(n^{2K+1})$, $K \geq 1$, not counting the membership test. However, its description is much more complicated and uses deeper results of graph theory than the ones presented in Section 4 regarding intersections of longest cycles. (Note that when we consider longest odd cycles, the theorems of Section 4 are no longer true. In particular, 2 longest odd cycles of length 5 can intersect in exactly one edge uv, and the resulting edge added $\{u, v\}$-fragments will be cycles of length 5.)

Acknowledgment

We are grateful to an anonymous referee who provided a very careful analysis of an earlier version of the paper and several useful comments.

References

[1] F. Barahona, "Sur la complexité du problème du verre de spins", Rapport de Recherche No. 177, Mathématiques Appliquées et Informatiques. Université Scientifique et Medicale de Grenoble, France, October 1979.

[2] F. Barahona, "On the complexity of max cut", Rapport de Recherche No. 186, Mathématiques Appliquées et Informatiques, Université Scientifique et Medicale de Grenoble, France, February 1980.

[3] F. Barahona, "Balancing signed graphs of fixed genus in polynomial time", Depto. de Matematicas, Universidad de Chile, Santiago, Chile, June 1981.

[4] P. Erdös and H. Hajnal, "On chromatic number of graphs and set systems", *Acta Mathematica Academiae Scientiarum Hungaricae* **17** (1966) 61–69.

[5] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness* (Freeman, San Francisco, 1979).

[6] M. Grötschel and W.R. Pulleyblank, "Weakly bipartite graphs and the max-cut problem", *Operations Research Letters* **1** (1981) 23–27.

[7] F.O. Hadlock, "Finding a maximum cut of a planar graph in polynomial time", *SIAM Journal on Computing* **4** (1975) 221–225.

[8] W.L. Hsu, Y. Ikura and G.L. Nemhauser, "A polynomial algorithm for maximum weighted vertex packings on graphs without long odd cycles", *Mathematical Programming* **20** (1981) 225–232.

[9] G.I. Orlova and Y.G. Dorfman, "Finding the maximum cut in a planar graph", *Engineering Cybernetics* **10** (1975) 502–506.

[10] R.E. Tarjan, "Depth-first search and linear graph algorithms", *SIAM Journal on Computing* **1** (1972) 146–159.

[11] H.-J. Voss, "Graphs with prescribed maximal subgraphs and critical chromatic graphs", *Commentationes Mathematicae Universitatis Carolinae* **18** (1977) 129–142.

[12] M. Yannakakis, "Node- and edge-deletion NP-complete problems", *Proc. 10th Annual ACM Symposium on Theory of Computing, Association of Computing Machinery* (New York, 1978) pp. 253–264.