ZOR – Methods and Models of Operations Research (1990) 34:195–206

Optimization and Optimality Test for the Max-Cut Problem

By Chr. Hohmann and W. Kern[1]

Abstract: We show that the following two problems are polynomially equivalent:

1) Given a (weighted) graph G, and a cut C of G, decide whether C is maximal or not.
2) Given a (weighted) graph G, and a cut C of G, decide whether C is maximal or not, and in case it is not, find a better solution C'.

As a consequence, an optimality testing oracle may be used to design a polynomial time algorithm for approximately solving the (weighted) Max-Cut Problem.

This in turn implies that recognizing optimal cuts in an unweighted graph is NP-hard.

Key words: Max-cut, decision problem, optimization problem, polynomial transformation.

Zusammenfassung: Wir zeigen, daß folgende Probleme polynomial äquivalent sind:

1) Gegeben ein (gewichteter) Graph G, ein cut C von G. Entscheide ob C optimal ist oder nicht.
2) Gegeben ein (gewichteter) Graph G, ein cut C von G. Entscheide, ob C optimal ist oder nicht, und falls nicht, finde einen besseren cut C'.

Eine Konsequenz hiervon ist, wie wir sehen werden, daß ein Optimalitätstest Orakel einen polynomialen Algorithmus zur approximativen Lösung von Max Cut Problemen liefert. Insbesondere folgt, daß das Erkennen von maximalen cuts in ungewichteten Graphen NP-schwer ist.

1 Introduction

The Max-Cut Problem is defined as follows: given a connected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}_+$, find a cut C in G, i.e., a minimal cutset $C \subseteq E$, of maximum weight.

Suppose we are given a polynomial time oracle which, given any weighted graph and a cut, decides whether or not the given cut is maximal. Then, as we will see, this

[1] Chr. Hohmann and Dr. W. Kern, University of Twente, Dept. of Applied Mathematics, Postbus 217, NL-7500 AE Enschede.

oracle may be used to design a polynomial time algorithm which does the following. Given a weighted graph $G = (V, E)$, it computes a cut C of G, which has the weight $w(C) \geqslant \left(1 - \frac{1}{|E|}\right) w(C^*)$ where C^* is any maximal cut of C. If every call to the oracle takes one unit of time, the algorithm will run in $O(|E|^4 \log |E|)$. Finally, we will prove a similar result for the unweighted case, showing that recognizing maximum cuts in an unweighted graph is NP-hard.

2 Finding a Better Solution

Let $G = (V, E)$ be a graph with weight-function $w : E \to \mathbb{R}_+$. The main impact of the above mentioned algorithm is a procedure which, given any suboptimal cut C of G, determines a better cut C_{new} by means of $O(|E|^2)$ calls to an optimality testing oracle. This will be desecribed in the following.

Thus let C be any suboptimal cut of G. In what follows, C will be fixed and we will refer to the edges $e \in C$ as "cut edges" and to the edges $e \notin C$ as "non-cut edges" throughout. For every $\lambda \in [0, 1]$ let G_λ denote the weighted graph obtained from G by scaling the weights of the noncut edges by a factor λ. Let w_λ denote the edge weighting of G_λ. Thus $w_\lambda(C) = w(C)$ for all λ, since only non-cut edges are scaled. Furthermore, C is optimal in G_0 and suboptimal in G_1.

W.l.o.g. we assume that $w(e) > 0$ for every $e \in E$. Let Δ denote the minimal weight of an edge $e \in E$ and let $\delta := 1/2 \cdot \Delta / w(E)$. Using the optimality testing oracle and binary search, we may compute $\alpha, \beta \in [0, 1]$ with $\alpha < \beta \leqslant \alpha + \delta$ such that C is (still) optimal in G_α, but suboptimal in G_β. In fact this can be done in linear time. In a minute, we will describe how to construct a cut C_{new} of G such that $w_\beta(C_{\text{new}}) > w_\beta(C)\ [= w(C)]$.

For simplicity, let us introduce the following notation. Let H be any graph with edge-weighting, say, u. If e is an edge of H and $0 \leqslant \epsilon \leqslant u(e)$, then $H \pm \epsilon e$ shall denote the weighted graph obtained by increasing (decreasing) the weight on e by ϵ. Furthermore, $H \backslash e$ denotes, as usual, the graph obtained from H by deleting the edge e.

Now let us come back to G. Recall that C is optimal in G_α, suboptimal in G_β, and we want to compute a cut C_{new} as above. Essentially, the idea is to slightly change the weights of the edges in G_α and G_β and check whether this makes C become suboptimal in G_α or optimal in G_β. This will give us information about which edges are possible candidates for C_{new}. More precisely, we are going to construct a sequence of graphs $G_\alpha^{(i)}$ and $G_\beta^{(i)}$; starting with $G_\alpha^{(0)} = G_\alpha$ and $G_\beta^{(0)} = G_\beta$, such that C is optimal in $G_\alpha^{(i)}$ and suboptimal in $G_\beta^{(i)}$ for all i.

The sequence $G_\beta^{(i)}$ will be such that in each step the number of candidates for C_{new} is reduced until there is just one left. This one will then be easy to determine.

The whole procedure is divided into two parts: One for handling the non-cut edges and one for handling the cut edges. We present the former first:

```
       PROCEDURE NON-CUT EDGES
       Let $G_\alpha^{(0)} := G_\alpha$ and $G_\beta^{(0)} := G_\beta$ and let $i = 0$.
LOOP:  FOR every non cut edge $e$ DO
       Define $e$ to be $\beta$-sensitive, iff $C$ is optimal in $G_\beta^{(i)} - \Delta e$
                        $\alpha$-sensitive, iff $C$ is suboptimal in $G_\alpha^{(i)} + \Delta e$
                        insensitive, iff $e$ is neither $\alpha$- nor $\beta$-sensitive.
       IF every edge is $\beta$-sensitive or insensitive, STOP.
       ELSE, let $e$ be an $\alpha$-sensitive edge, which is not $\beta$-sensitive.
       Let $G_\alpha^{(i+1)} := G_\alpha^{(i)} \backslash e$, $G_\beta^{(i+1)} := G_\beta^{(i)} \backslash e$,
       $i := i + 1$,
       GOTO LOOP.
```

When the algorithm terminates, there are only β-sensitive and insensitive edges left. The β-sensitive will be precisely those non-cut edges contained in C_{new}.

First note, that since Δ is the smallest weight of an edge in G, the edge weighting of $G_\beta^{(i)} - \Delta e$ is (still) nonnegative. Thus the "testgraphs" considered during the algorithm are feasible inputs for our optimality-testing oracle.

Claim 1: C remains optimal in $G_\alpha^{(i)}$ for all i.

Proof: This is clear, since we only remove non cut edges. □

The fact that C remains suboptimal in $G_\beta^{(i)}$ will be proved by induction on i. For notational convenience, let us call any cut C' a "β-candidate" of $G_\beta^{(i)}$, if the weight of C' in $G_\beta^{(i)}$ is larger than weight of C in $G_\beta^{(i)}$. (Of course, any β-candidate in any $G_\beta^{(i)}$ induces a cut C_{new} in G_β as required.) Thus we have to show that the set of β-candidates is non-empty for all i. Since C is suboptimal in $G_\beta^{(0)}$ this is certainly true for $i = 0$.

Now, let $i \geqslant 0$ and assume that

(*) there exists a β-candidate of $G_\beta^{(i)}$, i.e. C is suboptimal in $G_\beta^{(i)}$.

Let $w_\alpha^{(i)}$ and $w_\beta^{(i)}$ denote the weight functions of $G_\alpha^{(i)}$ and $G_\beta^{(i)}$. Thus in particular $w_\alpha^{(i)}(C) = w_\beta^{(i)}(C) = w(C)$ for all i.

Claim 2: Let C' be any β-candidate of $G_\beta^{(i)}$. Then $w_\beta^{(i)}(C') < w(C) + \Delta$ and $w_\alpha^{(i)}(C') > w(C) - \Delta$.

Proof: Obviously,

$w_\beta^{(i)}(C') \leqslant w_\alpha^{(i)}(C') + (\beta - \alpha)w(C) \leqslant w_\alpha^{(i)}(C') + (\beta - \alpha)w(E) < w_\alpha^{(i)}(C') + \Delta$; and since C is optimal in $G_\alpha^{(i)}$, we get

$$w_\alpha^{(i)}(C') \leqslant w_\alpha^{(i)}(C) = w(C).$$

This proves the first inequality. The second follows from

$$w_\alpha^{(i)}(C') > w_\beta^{(i)}(C') - \Delta > w(C) - \Delta. \qquad \square$$

Claim 3: An edge e in $G_\beta^{(i)}$ is β-sensitive if and only if it is a non-cut edge contained in every β-candicate of $G_\beta^{(i)}$.

Proof: "$\Rightarrow$": Let e be β-sensitive. Then C is optimal in $G_\beta^{(i)} - \Delta e$. But, by the inductive assumption (*), C is suboptimal in $G_\beta^{(i)}$. Since e is a non-cut edge, changing the weight on e cannot effect the value of C, and hence is must affect the value of every β-candidate of $G_\beta^{(i)}$.

"$\Leftarrow$": Let e be contained in every β-candidate of $G_\beta^{(i)}$. Then decreasing the weight of e by Δ decreases the weight of every β-candidate by Δ. By claim 2, this makes C become optimal.

Claim 4: An edge e in $G_\beta^{(i)}$ is α-sensitive, if it is a non-cut edge contained in some β-candidate of $G_\beta^{(i)}$.

Proof: Let e be contained in some β-candidate C' of $G_\beta^{(i)}$. Increasing the weight of e by Δ has no effect on the weight of C, but increases the weight of C' in $G_\alpha^{(i)}$ by Δ. By claim 2, this makes C suboptimal in $G_\alpha^{(i)}$, i.e., e is α-sensitive. $\square$

Since $G_\alpha^{(i+1)}$ and $G_\beta^{(i+1)}$ arise from $G_\alpha^{(i)}$ and $G_\beta^{(i)}$ by deleting non-cut edges which are α-sensitive, but not β-sensitive, Claims 3 and 4 imply that $G_\beta^{(i+1)}$ (still) contains β-candidates, i.e. that C is (still) suboptimal in $G_\beta^{(i+1)}$. Thus, our inductive assumption (*) remains valid for $i + 1$. Furthermore, if the above procedure runs into STOP in the i-th iteration, we have the following situation:

1) C is (still) optimal in $G_\alpha^{(i)}$ and suboptimal in $G_\beta^{(i)}$.

2) If C' is any β-candidate of $G_\beta^{(i)}$, then the set of non-cut edges contained in C' is precisely the set of β-sensitive edges in $G_\beta^{(i)}$.

The pair $G_\alpha^{(i)}$ and $G_\beta^{(i)}$ now becomes (together with C) the input for the second procedure, which handles the cut edges in a similar way:

```
        PROCEDURE CUT EDGES
LOOP:   FOR every cut edge $e$ DO
        Define $e$ to be $\beta$-sensitive, iff $C$ is optimal in $G_\beta^{(i)} + \Delta e$
                         $\alpha$-sensitive, iff $C$ is suboptimal in $G_\alpha^{(i)} - \Delta e$
                         insensitive, iff $e$ is neither $\alpha$- nor $\beta$-sensitive.
        IF every edge is either $\beta$-sensitive or insensitive: STOP.
        ELSE, let $e$ be an $\alpha$-sensitive edge, which is not $\beta$-sensitive.
        Let $G_\alpha^{(i+1)} := G_\alpha^{(i)} + \Delta e$ and let $G_\beta^{(i+1)} := G_\beta^{(i)} + \Delta e$
        $i := i + 1$,
        GOTO LOOP.
```

When this procedure runs into STOP, only β-sensitive and insensitive cut edges are left. We will see, that the insensitive cut edges together with the β-sensitive non cut edges (determined previously) make up a cut C_{new} as required. Again the following is obvious:

Claim 1': C remains optimal in every $G_\alpha^{(i)}$.

Next, let us assume that the following inductive assumption

(*') C is suboptimal in $G_\beta^{(i)}$

holds. (This is of course true for the initial value of i.) As before, let us call a cut in $G_\beta^{(i)}$ a β-candidate, if its weight in $G_\beta^{(i)}$ is strictly larger than the weight of C in $G_\beta^{(i)}$. Furthermore, let $w_\alpha^{(i)}$, $w_\beta^{(i)}$ denote the weight functions associated to $G_\alpha^{(i)}$ and $G_\beta^{(i)}$.

Claim 2': If C' is any β-candidate of $G_\beta^{(i)}$, then $w_\beta^{(i)}(C') < w_\alpha^{(i)}(C) + \Delta$ and $w_\alpha^{(i)}(C') > w_\alpha^{(i)}(C) - \Delta$.

Proof: This is proved in exactly the same way as Claim 2. □

Claim 3': A cut edge e of $G_\beta^{(i)}$ is β-sensitive if and only if no β-candidate of $G_\beta^{(i)}$ contains e.

Proof: If e is contained in a β-candidate, say C', of $G_\beta^{(i)}$, then obviously C can not become optimal by increasing the weight on e. Thus e is not β-sensitive.

Conversely, assume that e is not contained in any β-candidate. Then, by Claim 2', increasing the weight on e makes C become optimal. □

Claim 4': A cut edge e of $G_\beta^{(i)}$ is α-sensitive, if some β-candidate of $G_\beta^{(i)}$ does not contain e.

Proof: Let C' be a β-candidate of $G_\beta^{(i)}$ not containing e. Then, by Claim 2', decreasing the weight on e in $G_\alpha^{(i)}$ makes C become suboptimal. Thus e is α-sensitive. □

Claim 3' and 4' show that our inductive assumption (*') remains valid for $i + 1$. In fact if e is an α- but not β-sensitive edge in the i-th iteration, this means that some β-candidate of $G_\beta^{(i)}$, say C', contains e, and some other, say C'', does not. Increasing the weight of e in $G_\beta^{(i)}$ rules out C'' as a β-candidate in $G_\beta^{(i+1)}$ (as can be seen from Claim 2').

In fact, a cut is a β-candidate in $G_\beta^{(i+1)}$ if and only if it has been a β-candidate in $G_\beta^{(i)}$ and contains e. From this it is clear that after at most $|E|$ iterations, the procedure will run into STOP, thereby producing a pair $G_\alpha^{(j)}$, $G_\beta^{(j)}$ such that the following holds

1') C is still optimal in $G_\alpha^{(j)}$ and suboptimal in $G_\beta^{(j)}$

2') If C' is any β-candidate in $G_\beta^{(j)}$ then the set of cut edges contained in C' is precisely the set of insensitive cut edges in $G_\beta^{(j)}$.

Since during the whole procedure CUT EDGES, the set of β-candidates in each iteration is a (proper) subset of the set of β-candidates in the previous iteration, we may combine 1') and 2') together with the corresponding results 1) and 2) for the non-cut edges, to obtain the following.

Proposition 2.1: Let the procedure NON CUT EDGE and CUT EDGE run into STOP in iteration i and j, resp. Then $G_\beta^{(j)}$ contains a unique β-candidate C' consisting of the β-sensitive non cut edges in the i-th iteration of procedure NON CUT EDGES and the insensitive cut edges in the j-th iteration of the procedure CUT EDGES. □

Corollary 2.2: Based on the optimality testing oracle O, there exists a $O(|E|^2)$-algorithm which, given a suboptimal cut C of G, produces a better solution C_{new}. The difference between the weights of C and C_{new} is at least $(1-\beta)\Delta$ (recall that Δ was defined to be the smallest nonzero weight of an edge).

Proof: The time bound is trivial. Let C' be the β-candidate of $G_\beta^{(j)}$ as in Proposition 2.1. Then C' corresponds to a β-candidate C_{new} in $G_\beta^{(0)} = G_\beta$.

Let N be the set of non-cut edges of C_{new}. Then

$$w(C_{\text{new}}) \geqslant (1-\beta)w(N) + w_\beta(C_{\text{new}}) > (1-\beta)w(N) + w_\beta(C) \geqslant (1-\beta)\Delta + w(C),$$

since N is nonempty. □

3 An Approximation Algorithm Based on the Optimality Testing Oracle

From Corollary 2.2. one easily derives an approximation algorithm, based on the optimality testing oracle O. Let C be any cut of G, let C_{new} be the cut obtained by applying the procedures of section 2 and let C^* be any max cut of G. Let w, w_{new} and w^*, resp., denote the weight of these cuts.

Let $q := (w^* - w_{\text{new}})/(w^* - w) = 1 - (w_{\text{new}} - w)/(w^* - w)$. Thus the relative difference $r(C_{\text{new}}) = (w^* - w_{\text{new}})/w^*$ is obtained by multiplying the relative difference $r(C) = (w^* - w)/w^*$ with q.

Lemma 3.1: We may assume that $q \leqslant 1 - \dfrac{\Delta}{2w(E)}$.

Proof: By corollary 2.2, $w_{\text{new}} - w \geqslant (1-\beta)\Delta$. On the other hand, $w^* - w \leqslant (1-\alpha)w(E)$, since α has been chosen such that C is optimal in G_α. Therefore,

$$q = 1 - (w_{\text{new}} - w)/(w^* - w) \leqslant 1 - (1-\beta)\Delta/[(1-\alpha)w(E)]$$

Choosing β sufficiently close to α, we may assume that $(1-\beta)/(1-\alpha) \geqslant 1/2$. Thus $q \geqslant 1 - \dfrac{\Delta}{2w(E)}$. □

Thus starting with $C^{(0)} = C$, we may construct a sequence of cuts $C^{(1)}, C^{(2)}, \dots$ by successively applying the two procedures described in section 2, each time reducing the relative difference by a factor of $q \leqslant 1 - \Delta/2w(E)$. If $w^{(k)}$ denotes the weight of $C^{(k)}$, then

$$(w^* - w^{(k)})/w^* \leqslant q^k (w^* - w^{(0)})/w^* \leqslant q^k.$$

Now suppose we are given any $\epsilon > 0$ (presenting an upper bound for the relative difference we want to accept). Let G_{red} denote the graph obtained from G by deleting all edges of weight $\leqslant \epsilon \cdot w(E)/|E|$. Let w^*_{red} denote the weight of any max cut in G_{red}. Then, performing the above iterations in G_{red}, instead of G, we produce a sequence of cuts $C^{(0)}_{\text{red}}, C^{(1)}_{\text{red}}, \dots$, whose weights $w^{(0)}_{\text{red}}, w^{(1)}_{\text{red}}, \dots$, satisfy

$$(w^*_{\text{red}} - w^{k}_{\text{red}})/w^*_{\text{red}} \leqslant q^k_{\text{red}} \leqslant (1 - \Delta_{\text{red}}/2w(E_{\text{red}}))^k$$

where q_{red}, Δ_{red} and E_{red} are understood to have the obvious meaning. Thus $\Delta_{\text{red}} \geqslant \epsilon \cdot w(E)/|E| \geqslant \epsilon \cdot w(E_{\text{red}})/|E|$. Hence

$$(1)\quad (w^*_{\text{red}} - w^{(k)}_{\text{red}})/w^*_{\text{red}} \leqslant q^k_{\text{red}} \leqslant \left(1 - \frac{\epsilon}{2|E|}\right)^k \leqslant e^{\frac{-k\epsilon}{2|E|}}.$$

Furthermore, the relative difference between max cuts in G and G_{red} is given by

$$(2)\quad \frac{(w^* - w^*_{\text{red}})}{w^*} \leqslant \frac{w(E) - w(E_{\text{red}})}{w^*} \leqslant \frac{\epsilon \cdot w(E)}{1/2w(E)} = 2\epsilon$$

since obviously $w^* \geqslant 1/2w(E)$ always holds.

Combining (1) and (2), we get

$$\begin{aligned}(w^* - w^{(k)}_{\text{red}})/w^* &\leqslant (w^* - w^*_{\text{red}})/w^* + (w^*_{\text{red}} - w^{(k)}_{\text{red}})/w^* \\ &\leqslant 2\epsilon + (w^*_{\text{red}} - w^{(k)}_{\text{red}})/w^*_{\text{red}} \\ &\leqslant 2\epsilon + e^{-k\epsilon/2|E|}.\end{aligned}$$

Thus for example, if $\epsilon = 1/|E|$ we get a cut of relative difference $\leqslant 2/|E|$ by choosing $k \geqslant 2|E|^2 \ln |E|$.

Since each iteration takes $O(|E|^2)$ units of time, this yields a $O(|E|^4 \ln |E|)$ algorithm for computing a cut of weight at least $\left(1 - \dfrac{1}{|E|}\right) w^*$.

Note that in case of max cardinality cut problems ($w \equiv 1$), the above approximation algorithm will indeed find an optimum solution in polynomial time. However, it might look somewhat unfair using an optimality testing oracle for weighted graphs in order to solve problems in *unweighted* grpahs. Therefore we have studied the unweighted case separately. It turned out that a similar result as obtained above (i.e. polynomial equivalence of optimality test and improving a given suboptimal solution) also holds in the unweighted case. We sketch the main ideas in the following section.

4 The Unweighted Case

If one tries to imitate the procedure for finding a better solution by means of an optimality testing oracle, as described in the previous section, for the unweighted case, two problems will arise:

(1) The scaled graphs G_α and G_β are not feasible inputs for an optimality testing oracle for unweighted graphs.

(2) The modifications (decreasing or increasing the weight of an edge) made in the two procedures CUT EDGES and NON CUT EDGES yield infeasible inputs for the testing oracle (provided you are not willing to allow multiple edges in the testgraphs).

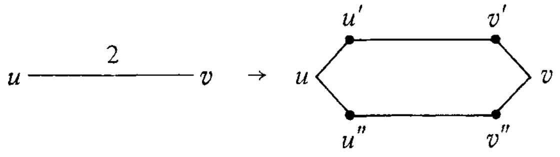
Fortunately, it turns out that (1) doesn't cause any trouble. In fact the cardinality problem allows a much simpler method to be used. This will be sketched below.

Problem (2) can be resolved as follows. Recall that in the two procedures CUT EDGES and NON CUT EDGES the weight of every edge is increased or decreased at most once. Thus, if the original graph has edges weighted by 1, then the testgraphs will have edges weighted by 1 or 2. Therefore, to handle the case of unweighted graphs, it will be sufficient to have an optimality testing oracle which is able to treat weighted graphs in which every edge has weight 1 or 2. Such an oracle, however can be build up from an arbitrary unweighted graph oracle as shown in the following.

Lemma 4.1: Given an optimality testing oracle for unweighted graphs, one can build up (in linear time) an optimality testing oracle for graphs with edge weightings 1 or 2.

Proof: Suppose we are given a graph $G = (V, E)$ with every edge weighted by 1 or 2, and a cut $C \subseteq E$, providing an instance for the "1–2 weighted" optimality testing oracle. This is transformed to an instance $\tilde{G} = (\tilde{V}, \tilde{E})$ and $\tilde{C} \subseteq \tilde{E}$ for the ordinary (unweighted) oracle as follows:

Replace every edge $e = (u, v) \in E$ of weight 2 by a subgraph as indicated below:



The unweighted graph obtained in this way (by replacing every edge of weight 2) is already the object graph $\tilde{G} = (\tilde{V}, \tilde{E})$. The cut $\tilde{C} \subseteq \tilde{E}$ is defined as follows:

If an edge $e = (u, v)$ of weight 2 belongs to C, then $\tilde{C}$ shall contain all six edges of the auxiliary subgraph in $\tilde{G}$ which replaces e. (Note that this is possible: Let $V = V_1 \dot{\cup} V_2$ define the bipartition of V corresponding to C. Since $e \in C$, we must have that, say, $u \in V_1$ and $v \in V_2$. Hence we may add u' and u'' to V_2 and v' and v'' to V_1 for constructing the bipartition of $\tilde{V}$ corresponding to $\tilde{C}$.)

If an edge $e = (u, v)$ of weight 2 does not belong to C then only 4 of the six edges of the auxiliary subgraph corresponding to e in $\tilde{G}$ shall be contained in $\tilde{C}$. (In this case we may assume that both u and v are in V_1 and we may add u', u'', v' and v'' to V_2 in order to get the bipartition corresponding to $\tilde{C}$.)

Finally, $\tilde{C}$ shall contain precisely those edges of weight 1 which are also included in C.

It is then straightforward to check that C is optimal in G if and only if $\tilde{C}$ is optimal in $\tilde{G}$. (Note that the weight of the non cut edges in G equals the weight of the non cut edges in $\tilde{G}$.) □

Thus assume that an optimality testing oracle is available which can treat $\{1, 2\}$-weighted graphs. Then, given any unweighted graph $G = (V, E)$ and a suboptimal cut $C \subseteq E$, we will find a better solution C_{new} by applying two procedures, similar to those described in section 3.

```
        PROCEDURE NON CUT EDGES
        Let $G^{(0)} := G; i = 0$
WHILE   there exists a non cut edge $e$ which can be removed without affecting sub-
        optimality of $C$:
        Let $G^{(i+1)} := G^{(i)} \backslash e; i := i + 1$,
STOP
```

When this procedure stops, say in the i-th iteration, then obviously C is still suboptimal in $G^{(i)}$, i.e. $G^{(i)}$ contains cuts which are larger than C. Moreover, any of these cuts contains precisely those non cut edges that are (still) contained in $G^{(i)}$. Finally, any of these cuts contains $|C| + 1$ edges in total.

```
        PROCEDURE CUT EDGES
WHILE   there exists a cut edge $e$ of weight 1 that can be weighted by 2 without af-
        fecting suboptimality of $C$:
        Let $G^{(i+1)} := G^{(i)} + 1 \cdot e; i := i + 1$,
STOP    (Output $C_{\text{new}}$ := the set of all cut edges of weight 2, together with all non cut
        edges left.)
```

As in section 3, let us call any cut in $G^{(k)}$ which is larger than C, simply a "candidate". Since this procedure starts with a graph the candidates of which surpass the weight of C exactly by 1, this remains true throughout the procedure: For the weight of C increases in each step and the candidates' weight can not grow faster. After each step, we still have a nonempty set of candidates. Their weight must have been increased in each step, so they contain all cut edges the weight of which has been put to 2 so far. This is still true when the procedure stops. Now let us look at the remaining cut-edges: C becomes optimal if we double the weight of any of these, so none of them can belong to a candiate. In fact, it follows, that there is just one candidate left, which is precisely the cut C_{new} that is output at the end of PROCEDURE CUT EDGES.

Thus we have proved

Proposition 4.2: Given an optimality testing oracle for the unweighted Max Cut Problem one can design a polynomial time algorithm for solving the Max Cut Problem. □

Hence, in particular, recognizing optimality even in the unweighted case is NP-hard.

5 Remarks

1. The main idea of the approximation algorithm appears to be a kind of homotopy-method, successively transforming an easy problem (provided by G_0) into a more difficult one (provided by G_1). We think that it would not be surprising if the method could be adopted to various other problems, (e.g., it works for the Max Clique Problem, too.)

2. As mentioned at the end of section 4, our results may be seen as a strengthening of the wellknown NP-completeness result for Max Cut: Given a graph G and integer k, the problem of deciding whether there exists a cut of size $>k$ remains NP-hard even if (in addition to the input G and k) we are given a cut C of size equal to k. As indicated in Remark 1, the same is true for the Maximum Clique Problem. Furthermore, a similar result has been proved for the Longest Path Problem (cf. [3]). Is there any general result hidden behind these special cases?

3. The results we obtained answer – to some extent – a question that has been considered in [1]. There, the notion of *depth* of a combinatorial optimization problem was studied for some selected examples. Intuitively, the depth of a feasible solution x_0 of some combinatorial optimization problem is defined to be the minimum net amount of "hill climbing" that has to be done in order to go from x_0 to any better solution. The results in [1] indicate that, from a computational complexity point of view, there might be a relationship between the following two problems:

1) computing the depth

2) solving the optimization problem

The relation of our work to [1] becomes obvious, once you notice that recognizing optimal solutions is equivalent to deciding whether a given solution has depth = 0 or >0. In fact, a solution is a "global" optimum (w.r.t. some prespecified neighbourhood structrue) if and only if it is a "local" optimum (which can be checked easily) and has depth equal to zero. The interested reader is referred to [1] for more details and further references.

References

1. Kern W (1988) On the depth of combinatorial optimization problems. Discr Appl Math (to appear)
2. Garey MR, Johnson DS (1979) Computers and intractibility. Freemans Co, San Fransisco
3. Papadimitriou CM, Steiglitz K (1977) On the complexity of local search for the traveling salesman problem. SIAM J Comp 6:76–83

Received September 1988
Revised version received March 1989