PDMI PREPRINT — 11/2002

A $2^{|E|/4}$-TIME ALGORITHM FOR MAX-CUT

Sergey S. FEDIN
St.Petersburg State University
Department of Mathematics and Mechanics
St. Petersburg
Russia
E-mail: fss@SF8469.spb.edu

Alexander S. KULIKOV
St.Petersburg State University
Department of Mathematics and Mechanics
St.Petersburg
Russia
E-mail: ask@AK8436.spb.edu

June 11, 2002

Abstract

In this paper we present an exact algorithm solving MAX-CUT in time $poly(|E|) \cdot 2^{|E|/4}$, where $|E|$ is the number of edges (there can be multiple edges between two vertices). This bound improves the previously known bound $poly(|E|) \cdot 2^{|E|/3}$ of [GHNR00].

1 Introduction

Given an undirected graph $G = (V, E)$ (where there can be multiple edges between two vertices) we ask for a cut of maximum weight, i.e., for a partition of V into V_1 and V_2 that maximizes the number of edges $(u_1, u_2) \in E$ for which $u_1 \in V_1$ and $u_2 \in V_2$. This *MAX-CUT* problem is one of the Karp's original NP-complete problems [Kar72], and has long been known to be NP-complete. Furthermore, MAX-CUT is MAX-SNP-hard [PY91], and so it is known that there exists a constant $c < 1$ such that a c-approximation algorithm for this problem would imply that $P = NP$ [ALMSS92]. Besides its theoretical importance, the MAX-CUT problem has applications in circuit layout design and statistical physics (Barahona et al. [BGJR98]). For a comprehensive survey of the MAX-CUT problem, the reader is referred to the paper of Poljak and Tuza [PT95].

Similarly to the algorithm of [GHNR00], our algorithm is a typical splitting algorithm. Namely, it uses a procedure, called splitting procedure, which reduces the problem for a graph G to the problem for two graphs with smaller number of edges. Before the algorithm splits each of the obtained two graphs, it can simplify them using transformation rules.

The algorithm of [GHNR00] transforms an instance of MAX-CUT into an instance of MAX-2-SAT, and then solves the latter. While the analysis of their algorithm uses the fact that the input formula originates from MAX-CUT instance, the algorithm itself does not make use of this fact. Contrary to this algorithm, our algorithm solves MAX-CUT directly. Since formulas in 2-CNF corresponding to graphs have specific structure, instead of common MAX-2-SAT transformation rules we can use rules that can be applied to these formulas only.

Our algorithm works with a slight generalization of MAX-CUT problem: we attach two numbers Δ_1 and Δ_2 to each vertex of our graph, where Δ_i denotes the weight that would be added to the maximum cut if the vertex is put into the corresponding set V_i. Informally speaking, these numbers correspond to the 1-clauses of the 2-CNF formula obtained after splitting an instance of MAX-CUT problem formulated as a formula in 2-CNF. However, while 1-clauses are not counted in the algorithm of [GHNR00], they cannot be eliminated in MAX-2-SAT (and this puts certain restrictions on formula simplification rules). On the contrary, Δ_1 and Δ_2 are not harmful and allow to simplify graphs with Δ_i's similarly to ordinary graphs. In particular, our algorithm uses a new transformation rule which eliminates end vertices (in a graph with Δ_i's). This rule can be also formulated in terms of MAX-2-SAT: it eliminates the variable if it appears only in two 2-clauses with the same weight (these two 2-clauses correspond to an edge) and in several 1-clauses. This rule helps us to solve the MAX-CUT problem for some special graphs. For example, we can solve in polynomial time the problem for the graph (with Δ_i's) in which each vertex has degree 2.

We show that at almost every step of our algorithm we can make a splitting corresponding to the recurrence inequality $T(|E|) \le 2 \cdot T(|E| - 4) + poly(|E|)$ for the running time. The only graph for which we cannot make such a splitting, is a graph in which each vertex has exactly three neighbors, and there are no multiple edges. We prove that this situation cannot appear more than once on any path of the splitting tree.

2 Our result

2.1 General setting

To represent a graph with multiple edges between two vertices, we use a graph where every edge is assigned an integer weight. Let $W(u, v)$ denote the weight of edge between vertices u and v. By

$W(v)$ we denote the total weight of edges between v and its neighbors. The *degree* of vertex is the number of its neighbors.

The *weight of cut* for partition (V_1, V_2) of vertex set V is the sum of weights of those edges (u_1, u_2) for which $u_1 \in V_1$, $u_2 \in V_2$. A partition (V_1, V_2) is called *optimal* if it corresponds to the maximum possible weight of cut.

At the beginning of our algorithm the sets V_1 and V_2 are empty. At each step we move some vertices from V to one of the partition sets (when we remove a vertex from V, we remove also all incident edges from E). Therefore some edges are added to the cut. Let $CurrentCut$ be a variable denoting the weight of this cut.

Let Δ_1 and Δ_2 denote integer functions on the set of vertices. Intuitive concept of these functions is as follows: $\Delta_1(v)$ (or $\Delta_2(v)$) is a number that will be added to the current value of cut when we add v to the set V_1 (resp. V_2). These functions help us to save the information about the weight of removed edges as follows: Let the graph contains vertices u and v such that $(u, v) \in E$. Suppose that at some step of our algorithm we move v from V to V_1. In this case we need to remember that if we move u to V_2 then edge $W(u, v)$ will be added to the weight of cut. For saving this information we change $\Delta_2(u)$ as follows: $\Delta_2(u) = \Delta_2(u) + W(u, v)$.

2.2 Initialization procedure

We suppose that each graph G appearing in our algorithm consists of: vertex set $G.V$, edge set $G.E$ (each edge is assigned an integer weight), variable $G.CurrentCut$ and two functions $G.\Delta_1$ and $G.\Delta_2$. The weight of cut (V_1, V_2) for such a graph is equal to $\sum_{(u,v)\in E, u\in V_1, v\in V_2} W(u, v) + \sum_{u\in V_1} \Delta_1(u) + \sum_{u\in V_2} \Delta_2(u)$. Before applying our algorithm to the graph G we apply the following procedure:

Procedure *Init*(Graph G)

$G.CurrentCut = 0$

FOR each $u \in G.V$

$\Delta_1(u) = 0, \Delta_2(u) = 0$

end FOR

2.3 Splitting

By means of splitting we reduce the problem for a graph G to the problem for two graphs $G[v \in V_1]$ and $G[v \in V_2]$ (where $v \in V$). We make splittings using the following procedure:

Procedure *Split*(Graph G, Vertex v)

FOR $i = 1, 2$

(a) $G[v \in V_i].CurrentCut = G.CurrentCut + \Delta_i(v)$

(b) $G[v \in V_i].E = G.E$

(c) $G[v \in V_i].V = G.V - \{v\}$

(d) FOR each $u \in G[v \in V_i]$

$G[v \in V_i].\Delta_1(u) = G.\Delta_1(u)$,
$G[v \in V_i].\Delta_2(u) = G.\Delta_2(u)$

end FOR

(e) FOR each $u \in G.V$

IF $(u, v) \in G.E$

$G[v \in V_i].\Delta_{3-i}(u) = G[v \in V_i].\Delta_{3-i}(u) + W(u, v)$,
$G[v \in V_i].E = G[v \in V_i].E - \{(v, u)\}$

end IF

end FOR

end FOR

return $G[v \in V_1]$ and $G[v \in V_2]$

In short, splitting procedure removes vertex v and all its edges from graph and saves in $\Delta_i(u)$ the information that if vertices u and v belongs to different partition sets then edge (u, v) belongs to the cut. It is obvious that $MaxCut(G) = \max\{MaxCut(G[v \in V_1]), MaxCut(G[v \in V_2])\}$. Note that $W(G[v \in V_1]) = W(G[v \in V_2]) = W(G) - W(v)$.

2.4 Transformation Rules

To obtain the cut of maximum weight we must consider all partitions and choose the optimal one from them. Thus for every vertex v we must consider two cases: $v \in V_1$ and $v \in V_2$. But for some cases we can prove that at least in one optimal partition some vertex belongs to one of these sets. In this situation we do not need to consider the case when this vertex belongs to the other set. *Transformation rules* help us to handle such vertices.

A *correct* transformation rule replaces a graph G with a simpler graph G' such that $MaxCut(G) = MaxCut(G')$, i.e., a correct transformation rule preserves the weight of maximum cut. Each transformation rule is executed in polynomial time and does not increase the total weight of the edges.

We now list the transformation rules we use in our algorithm and prove their correctness.

Rule 1: Guaranteed value. This rule changes graph as follows:

FOR each $u \in G.V$

$G.CurrentCut = G.CurrentCut + min\{\Delta_1(u), \Delta_2(u)\}$
$G.\Delta_1(u) = G.\Delta_1(u) - min\{\Delta_1(u), \Delta_2(u)\}$
$G.\Delta_2(u) = G.\Delta_2(u) - min\{\Delta_1(u), \Delta_2(u)\}$

end FOR

Lemma 1. Rule 1 is correct.

Proof. Note that in a leaf of splitting tree each vertex must be in exactly one of the sets V_1 or V_2. So, when vertex v is added to one of the partition sets, at least $min\{\Delta_1(v), \Delta_2(v)\}$ is added to the $CurrentCut$. □

Rule 2: Isolated vertex. If graph contains a vertex v of weight 0, then **Rule 2** changes graph this way: $G.V = G.V - \{v\}$, $G.CurrentCut = G.CurrentCut + \max\{\Delta_1(v), \Delta_2(v)\}$. Clearly this rule is correct.

Rule 3: End vertex. Suppose that G contains vertices v and u such that $(u, v) \in E$ and $Deg(v) = 1$. Note that at least one of the functions $\Delta_1(v)$ and $\Delta_2(v)$ is equal to 0 (if not, apply **Rule 1**). Without lost of generality assume that $\Delta_2(v) = 0$.

Let us first assume that $\Delta_1(v) \geq W(u, v)$. Then **Rule 3** changes graph as follows: $\Delta_2(u) = \Delta_2(u) + W(u, v)$, $CurrentCut = CurrentCut + \Delta_1(v)$, $E = E - \{(u, v)\}$, $V = V - \{v\}$.

Now assume that $\Delta_1(v) < W(u, v)$. Then **Rule 3** changes graph as follows: $\Delta_2(u) = \Delta_2(u) + \Delta_1(v)$, $CurrentCut = CurrentCut + W(u, v)$, $E = E - \{(u, v)\}$, $V = V - \{v\}$.

Lemma 2. **Rule 3** is correct.

Proof. Consider the case $\Delta_1(v) \geq W(u, v)$. Suppose that in some optimal partitions $v \in V_2$. Let us then move v from V_2 to V_1. Clearly by doing this action we do not decrease the weight of cut. Thus, resulting partition is optimal too. It means that we do not need to consider the case $v \in V_2$. So, we just save in graph the information that the edge (v, u) will be added to the cut if $u \in V_2$. Then we add $\Delta_1(v)$ to the $CurrentCut$ and remove v from graph.

Now assume that $\Delta_1(v) < W(u, v)$. Note that if an optimal partition where u and v belong to same set V_i exists then by moving v from V_i to V_{3-i} we increase the weight of cut. Hence, every optimal cut includes the edge (u, v). So, we save the information that if in optimal partition u be in V_2 then v will be in V_1 and hence $\Delta_1(v)$ will be added to the $CurrentCut$. After that we add $W(u, v)$ to the $CurrentCut$ and remove v from graph. □

Rule 4: Small connected component. We can easily find an optimal cut for a small (say, with at most 4 edges) connected component of graph G. So, **Rule 4** removes such a component from G and adds the weight of maximum cut for this component to $G.CurrentCut$.

2.5 The algorithm

In this section, we present Algorithm MAX-CUT which solves MAX-CUT in time $poly(|E|) \cdot 2^{|E|/4}$, where $|E|$ is the total weight of edges of the input graph. We first present the algorithm and then show its correctness and estimate its running time.

Algorithm MAX-CUT

Input: A graph G after applying Procedure Init.
Output: The weight of maximum cut of G.

Method

1. Apply **Rules 1–4** to G as long as at least one of them is applicable.

2. If G does not contain any edges, return $CurrentCut$.

3. If G contains a non-trivial connected component G_1, then return $MaxCut(G_1) + MaxCut(G - G_1)$.

4. If there is vertex v of weight at least 4, return[1] $\max\{MaxCut(G[v \in V_1]), MaxCut(G[v \in V_2])\}$.
5. If there are vertices u and v such that $W(u,v) = 2$, return $\max\{MaxCut(G_1), MaxCut(G_2)\}$, where G_1 and G_2 are the graphs resulting from graphs $G[v \in V_1]$ and $G[v \in V_2]$ respectively after applying transformation rules.
6. If each vertex has degree 2 then choose any vertex v and return $\max\{MaxCut(G_1), MaxCut(G_2)\}$, where G_1 and G_2 are the graphs resulting from graphs $G[v \in V_1]$ and $G[v \in V_2]$ respectively after applying transformation rules.
7. If there are vertices u and v such that $Deg(v) = 3$, $Deg(u) = 2$, and $(u,v) \in G.E$ return $\max\{MaxCut(G_1), MaxCut(G_2)\}$, where G_1 and G_2 are graphs resulting from graphs $G[v \in V_1]$ and $G[v \in V_2]$ respectively after applying transformation rules.
8. If each vertex has degree 3 and each edge has weight 1 then return $\max\{MaxCut(G[v \in V_1]), MaxCut(G[v \in V_2])\}$ for some vertex v.

Theorem 1. Given an undirected graph $G = (V,E)$ where edges are assigned integer weights, *Algorithm MAX-CUT* finds the weight of maximum cut of G in time $poly(|E|) \cdot 2^{|E|/4}$, where M is the total weight of edges of G.

Proof. Correctness. The correctness of transformation rules is shown in Section 2.4. The correctness of steps of *Algorithm MAX-CUT* is trivial. Let us show that every possible graph after applying transformation rules satisfies one of the conditions of steps 2–8.

Suppose that the graph G does not satisfy the conditions of steps 2–7. In this situation G is connected. G does not contain edges of weight at least 3, since there are no vertices of weight at least 4 and no end vertices in G. Also G does not contain edges of weight 2, because it does not satisfy the condition of step 5. So, each edge has weight 1 and each vertex has degree either 2 or 3. If all vertices has degree 2 then G satisfies the condition of step 6. If G consists of vertices of degree either 2 or 3, where both possibilities are realized, then there must be two vertices u and v such that $Deg(u) = 2, Deg(v) = 3, (u,v) \in E$ (since G is connected). In this case G satisfies condition of step 7. Thus, each vertex has degree exactly 3. It means that G satisfies the condition of step 8.

Running time. After applying transformation rules to G, *Algorithm MAX-CUT* makes two recursive calls for graphs with smaller total weight of edges (unless G becomes trivial) in one of the steps 3–8. Clearly, the total running time of the algorithm is the total running time of the two recursive calls plus time spent to make these calls. Therefore, the running time is within a polynomial factor of the number of nodes (or leaves) of the recursion tree.

Now we show that in steps 3–7 we can make a recursive call for a graphs G_1 and G_2 such that $|G.E| - |G_1.E| \geq 4, |G.E| - |G_2.E| \geq 4$ (it corresponds to the reccurence inequality $T(|E|) \leq 2 \cdot T(|E| - 4) + poly(|E|)$ on the running time). This statement is trivial for steps 3 and 4. If G satisfies the conditions of steps 5 or 7, then u is an end vertex in graphs $G[v \in V_1]$ and $G[v \in V_2]$ (since there are no vertices of weight at least 4 and no end vertices in G) and **Rule 2** removes it from them. If G satisfies the condition of step 6, then vertices of the graph make a cycle. So, after splitting we can remove all other vertices of this cycle by means of **Rule 2**.

[1] In the following, we denote by $G[v \in V_1]$ and $G[v \in V_2]$ the graphs constructed by Procedure Split.

Now let us prove that if G satisfies condition of step 8 and does not satisfy conditions of all other steps, then all following graphs will not satisfy this condition. Suppose that each vertex of graph G has degree 3 and after executing some transformation rules and steps of our algorithm, G is transformed into graph G_1 satisfying the same condition. Note that neither transformation rules nor steps of our algorithm increase degree of any vertex. Hence $G_1.V$ was a connected component in G, but this contradicts with the fact that G was connected.

As we have shown above each branch of splitting tree contains at most one graph satisfying condition of step 8. Therefore, the number of leaves is $O(2^{|E|/4})$.

$\square$

Acknowledgements

The authors thank Edward A. Hirsch for bringing the problem to their attention, and for valuable remarks.

References

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[BGJR98] F. Barahona, M. Grotschel, M. Junger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout desidn. *Operations Research*, 36:493–513, 1988.

[GHNR00] J. Gramm, E. A. Hirsch, R. Niedermeier, and P. Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. Technical Report 00-037, Electronic Colloquium on Computational Complexity, 2000. To appear in *Discrete Applied Mathematics*.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computations*, pages 85–103. Plenum Press, New York, NY, 1972.

[PT95] S. Poljak and Z. Tuza. Maximum cuts and large bipartite subgraphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 20:181–244, 1995.

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.