An Investigation of the MaxCut Problem*

John W. Wheeler

Computer Science Department

The University of Iowa

Iowa City, IA 52242

wheeer@cs.uiowa.edu

April 27, 2004

Abstract

The MaxCut problem seeks to partition the vertices of a graph into two sets such that the weight of the edges joining those sets is maximized. The MaxCut problem has been of continued research interest and has developed an extensive literature. After reviewing of a small portion of that literature, this article discusses an approach to the solution of this problem based on a branch and bound algorithm composed with a divide and conquer preprocessing phase. Preliminary results are given and several directions for further research are outlined.

1 Introduction

Given a weighted undirected graph, the MaxCut problem seeks to partition the vertices of the graph into two sets such that the sum of the weights of the edges that join the two sets is maximized. Though simple to state, the MaxCut problem is among those that are known to be computationally intractable, i.e., it is an NP-complete problem. Despite this difficulty, the

*Partially supported by funding from the U.S. Army Materiel Systems Analysis Activity.

MaxCut problem has many applications and has been reformulated in a multitude of ways. That the MaxCut problem deserves further study is evidenced by the continuing active research in this and related areas.

Applications of MaxCut come from diverse areas, including layout of electronic circuitry, state problems in statistical physics, and combinatorial optimization. Several of these applications will be described in more detail in Section 3.3. Many methods have been applied to the solution of the MaxCut problem, ranging from random selection schemes to exhaustive search. These methods of attack vary in approach and rigor. A selection of these methods are categorized and discussed in Section 4.

This report is organized as follows. The next section provides some terminology and formally defines the MaxCut problem. Sections 3 and 4 then review some of the literature about the computational complexity, applications, and solution methods of the MaxCut problem. The remainder of the report discusses the approach we are developing for the MaxCut question, describes of our preliminary results, and outlines several directions for future work.

2 Preliminaries

We define an undirected simple graph $G = (V, E)$ to be composed of a finite set of n vertices, $V_n = \{1, 2, \ldots, n\}$, and a set of m edges each identified by the vertices it joins, $E_m = \{(i, j) : 1 \leq i < j \leq n\}$. To denote an individual edge we use e, e_{ij} or ij, alternatively. The degree of a vertex, d_v, is the number of edges connected to it. By the notation K_n we will denote the complete simple graph on n vertices, i.e. the graph containing an edge between every pair of vertices.

A *path* from vertex i to vertex j is a graph composed of a series of vertices and the edges that join them,

$$V(P) = \{x_0, x_1, \ldots, x_l\} \qquad E(P) = \{x_1x_2, x_2x_3, \ldots, x_{l-1}x_l\}.$$

A path is usually denoted by $x_0, x_1, \ldots, x_l$. When a path exists between two vertices, we say they

are connected. A *cycle* is a path in which $l \geq 3$, $x_0 = x_l$, and $x_i \neq x_j$ for all $i, j \in \{0, 1, \ldots, l-1\}$ such that $i \neq j$.

The *subgraph* induced by a subset $S \subseteq V$ is denoted G_S and consists of the vertices in S and only those edges of the graph G which join vertices in S. A *component* is the subgraph induced by a maximal set of connected vertices. Finally, a *biconnected component* is the subgraph induced by a maximal set of vertices such that every pair of vertices is in at least one cycle. The reader is referred to a reference text, such as Bollobás [3], for further information on graph theory.

Given a subset of the vertices of a graph, $S \subseteq V_n$, we define a *cut* induced by S as the set of edges with one end in S and the other in $V - S$. Equivalently, the cut, $\delta_G(S)$, is defined as,

$$\delta_G(S) := \{ij \mid |S \cap \{i, j\}| = 1\}.$$

When the graph G is understood from the context, we write simply $\delta(S)$. It is often convenient to speak of the set of edges that do not participate in a cut. We call these *dropped* edges and denote the set of dropped edges in a cut induced by set S as $D_G(S)$, or $D(S)$ when G is understood. This set is defined by,

$$D_G(S) := \{ij \mid |S \cap \{i, j\}| = 2\} \cup \{ij \mid |(V - S) \cap \{i, j\}| = 2\}.$$

When we discuss bounding functions, we will denote the set of edges dropped by partial assignment of vertices $i, \ldots, j$ to either S or $V - S$ by $D_i^j(S)$. Thus, $D_0^{i-1}(S)$ is the set of edges dropped by the assignment of the first $i - 1$ vertices, while $D_i^n(S)$ are the set of edges that will be dropped by assignment of the remaining vertices, $i, \ldots, n$.

The incidence vector of the cut $\delta(S)$, or cut vector, is that member of the $\binom{n}{2}$-dimensional Cartesian binary space $\{0, 1\}^{\binom{n}{2}}$ where $\delta(S)_{ij} = 1$ iff $|S \cap \{i, j\}| = 1$. The cut vector is also denoted $\delta(S)$, by an abuse of notation, and is used as a basis for developing the polyhedral theory of the MaxCut problem, a subject we will discuss briefly in Section 4.

Let weight c be a function assigning to each edge a non-negative real value, $c : E \to \mathbb{R}_+$, $c(e_{ij}) = c_{ij}$. Given a cut $\delta(S)$, the weight of the cut is the sum of the weights of the edges in the cut.

$$c(\delta(S)) := \sum_{ij \in \delta(S)} c_{ij}$$

The MaxCut problem has two forms: a general form and a simple form. The general MaxCut problem asks to find the maximum cut weight taken over all possible cuts of the weighted graph G.

$$mc(G) = max\{c(\delta(S)) \mid S \subseteq G\}$$

The simple MaxCut problem restricts the weighting function to take on only the values 0 or 1. Thus, the simple MaxCut problem asks only for the number of edges in the largest cut.

3 A Few Facts about the MaxCut Problem

We list here a few useful facts about the complexity of the MaxCut problem and mention some of the many applications in which the problem has found use.

3.1 NP-completeness

The MaxCut problem, even in its restricted simple form, was shown by Karp [17] to be an NP-complete problem. He did this by reducing the Partition problem, one of a collection of problems which he proved be intractable in the same paper, to the MaxCut problem. Additional proofs of the NP-completeness of the MaxCut problem can be found in [20].

Given that the general problem is NP-complete, a polynomial time approximation algorithm becomes an attractive alternative. We will mention several positive results on such algorithms in Section 4. However, Håstad has shown there are also limits to this approach. In [14] he demonstrated that even approximating the value of the MaxCut to a performance guarantee of

any thing better than $16/17 = .94117$ is itself an NP-hard problem.

3.2 Classes of graphs for which polynomial time algorithms are known

Another approach to the MaxCut problem asks whether there are restricted classes of the general unweighted graphs for which the problem becomes tractable. Again, some progress has been made in this direction. Three related classes of graphs have been found that admit polynomial time solutions for the MaxCut problem. These are the planar graphs, the graphs not reducible to K_5, and the weakly bipartite graphs [6, 20]. While the planar graphs and the graphs not reducible to K_5 admit an intuitive understanding, the weakly bipartite graphs are not so easily characterized. Guenin, [13], defines a graph as weakly bipartite "if the polyhedron Q is integral (i.e., all its extreme points are integral):

$$Q = \{x \in \mathbb{R}_+^{|E|} \mid \textstyle\sum_{i \in C} x_i \geq 1, \text{ for all odd cycles } C \text{ of } G\}."$$

He notes that if $\hat{x}$ is a $\{0, 1\}$ extreme point of this polyhedron, then $\hat{x}$ is an incidence vector which intersects every odd cycle in G. This being the case, $\mathbf{1} - \hat{x}$, where $\mathbf{1}$ is the vector of all 1's, is the incidence vector of a bipartite subgraph of G. In particular, if $\hat{x}$ is the solution of

$$\min\{cx \mid x \in Q \cap \{0, 1\}^{|E|}\}$$

then $\mathbf{1} - \hat{x}$ is a solution to the general MaxCut problem. He notes that others have shown that both planar graphs and graphs not reducible to K_5 are weakly bipartite graphs. Poljak and Tuza [20] give other classes of graphs that are also weakly bipartite. For instance, a graph G with a vertex $v \in V$ such that the graph $G - \{v\}$ is bipartite is a weakly bipartite graph. Both sources warn, however, that the weakly bipartite graphs have not been completely characterized.

3.3 Applications of MaxCut

The MaxCut problems has many applications in diverse fields. One of the more commonly cited examples comes from the field of design of very large scale integrated (VLSI) circuits. This problem asks for the minimization of the number of vias, or cross-layer connections, in a circuit design subject to pin assignments and layout constraints. Another very commonly mentioned application is in statistical physics where the ground states of spin glasses in the presence of an external magnetic field are sought.

Other applications can be found in the area of combinatorial optimization where many geometric reformulations of the MaxCut problem are possible based on the incidence vector representation. Examples here include linear programming over the cut polytope and unconstrained quadratic 0-1 programming. The interested reader is referred to the extensive surveys of Deza and Laurent [5, 6] and Poljak and Tuza [20] for more information on the applications of the MaxCut problem.

4 Solution Methods

4.1 Classification of methods

Approaches to the solution of the MaxCut problem can be characterized in a number of ways. Two characteristics will be used in this article as a framework within which to discuss several of these approaches. These two characteristics are (1) whether the method admits an exact solution, an approximate solution with some performance guarantee, or simply a solution without performance guarantee and (2) whether the method attacks the problem directly, manipulating a representation of a graph, or indirectly through reformulating the problem into another problem domain.

4.2 Methods examples

4.2.1 Exact methods

The naive approach to an exact solution for the MaxCut problem enumerates the 2^n possible cuts of the graph, calculates the weight of each, and records the value of the cut that provides the greatest weight. While this approach will find the maximum weight cut, it will require time $O(m2^n)$ to examine all of the cuts.

The process of enumerating the cuts in the naive approach can be conceptualized as a complete traversal of the binary decision tree in which each branch is split based on the decision of whether to include each vertex in the cut set S. A natural way to reduce the time required to traverse this tree is to apply a heuristic function to estimate the value of the MaxCut that can be achieved based on the current state of the search. Such heuristics are of two types: upper bounds and lower bounds. Both are useful, although in different ways, to limit the traversal of the decision tree. In both cases, the best currently known cut weight is compared to the bound value. If the currently known value is greater than the upper bound achievable given the present state of the search, the current branch of the search tree can be abandoned. If, on the other hand, the currently known value is below the lower bound of that achievable given the current state of the search, the known value may be updated to the bound value. While in this case the remaining subtree must still be traversed, portions of that subtree may be avoided based on the improved known value. Examples of this approach and a variety of bounds used to limit search in this way are found in [22, 23].

Another way to reduce problem complexity is to divide the problem into smaller component problems, solve these smaller problems, and aggregate their solutions. Fedin and Kulikov [9] discuss one example of this approach. Their approach systematically decomposes the problem graph by removing vertices of low degree while preserving the implications of the removed vertices by modifying the characteristics of the remaining vertices. When no further simplifications are

applicable, an arbitrary vertex is selected and the problem is split on the assignment of that vertex to one or the other partition. These two subproblems are then each submitted to the full algorithm for solution and the best solution is returned. They report that their algorithm runs in time $poly(m) \cdot 2^{m/4}$ improving the running time of $poly(m) \cdot 2^{m/3}$ reported by Gramm, Hirsch, Niedermeier, and Rossmanith in [12].

4.2.2 Approximation methods with performance guarantees

Relaxing the requirement for an exact solution leads to the study of p-approximation algorithms, or polynomial time algorithms that provide a solution that is at least p times the optimal value. The constant p is called the the performance guarantee of the algorithm. Goemans and Williamson [10] cite five algorithms with improving performance guarantees before discussing their own algorithm. Here we only mention the first of these and the contribution of Goemans and Williamson. In 1976, Sahni and Gonzales [21] presented a .5-approximation algorithm for the MaxCut problem. Their algorithm iterates through the vertices and assigns each vertex to maximize the partial cut that has resulted from the assignment of the preceding vertices. Their algorithm essentially corresponds to a randomized algorithm in which each vertex is assigned to a partition based on the flip of an unbiased coin. In 1994, Goemans and Williamson became the first to apply semidefinite programming to the solution of the MaxCut problem. Their algorithm uses a randomized rounding of the solution to a non-linear relaxation of the MaxCut problem to give a .878-approximation.

4.2.3 Inexact methods without performance guarantees

Many of the solution methods that have been developed in the study of artificial intelligence have been also applied to the solution of the MaxCut problem. Such methods include random selection, restarting local search, simulated annealing, and genetic programming in its many forms. Most of these approaches rely on some form of random selection process and while the the solution they provide generally improves as the algorithm is allowed to continue to run, they cannot

guarantee any better performance than the .5-approximation algorithm described by Sahni and Gonzales. These approaches are not without appeal and continue to attract research attention. Two examples of studies involving these approaches are found in [11, 7].

In [11], Gosti, Nguren, Wan, and Zhou consider the value of performing a local search in the vicinity of the solution returned by the Goemans and Williamson algorithm as a method for improving that solution. They compared this composite approach with a Metropolis search, with Simulated annealing, and with local search alone. They concluded the Metropolis, the Simulating Annealing and the Goemans and Williamson algorithm followed by local search were all effective for graphs up to 100 vertices. For larger problems, however, they recommend simulated annealing as the method of choice.

In [7], Dolezal, Hofmeister, and Lefmann discuss an empirical study comparing a similar set of solution methods for the MaxCut problem. Specifically, they compared five methods: an application of the Goemans and Williamson semidefinite programming approach, a random strategy, a genetic algorithm, two combinatorial algorithms, and a divide and conquer strategy. Based on their sampling of randomly created graphs, they concluded the approach that randomly selected candidate cuts for a preset time and kept the maximum cut value gave the best trade off between runtime and accuracy of the result.

4.2.4 Reformulation into different problem space

The MaxCut problem can be reformulated in numerous ways. For example, it is well known that by the following theorem the MaxCut problem may be mapped into an equivalent Max-2-Sat problem. A Max-2-Sat problem takes a set of clauses in conjunctive normal form each containing at most 2 literals and asks for the maximum number of clause that can be simultaneously satisfied.

Theorem 1 ([4, 18]**).** *Given graph $G = (V, E)$, where $|V| = n$, $|E| = m$, assume weight $w(i, j) = 1$ for each $(i, j) \in E$. Construct an instance of MAX-2-SAT as follows: Let V be the set of propositional variables and for each edge $(i, j) \in E$ create exactly two binary clauses: $(i \vee j)$ and $(\overline{i} \vee \overline{j})$.*

Let F be the collection of such binary clauses, then the MaxCut problem on the graph G has a cut of weight k iff the constructed MAX-2-SAT problem has an assignment under which $m + k$ clauses are true.

It is interesting to note here that Alber, Gramm, and Niedermeier [1] have observed that while the MaxCut problem is thus theoretically equivalent to the Max-2-Sat problem, in practice, random MaxCut problems reformulated as Max-2-Sat problems are generally harder to solve than random Max-2-Sat problems owing to the special structure of the transformed problem.

Another example of a reformulation that can be applied to the MaxCut problem is that of transforming graphs into points in $|V|$-dimensional Cartesian space by the use of incidence vectors. This approach allows to apply the very large array of tools that have been developed in the areas of combinatorial optimization, convex programming, and polyhedral theory. The approximation method of Goemans and Williamson falls into this category. A more complete introduction to these areas is well beyond the scope of the current work. Again, the interested reader is referred to the articles [5, 6] and [20] for further information.

5 Current Work

Based on the recent success of Shen and Zhang with the Max-2-Sat problem [22, 23], we have mounted a direct attack on the MaxCut problem. Our approach combines a preprocessing phase with a branch and bound algorithm. The preprocessing step we have implemented identifies and separates certain subgraphs that are easily shown to be bipartite. In our case, the selected subgraphs are the graphs composed of only edges that participate in no cycle and the biconnected components that contain no cycle of odd length. Once identified, these subgraphs are removed from the problem graph and all edges in them are included in the MaxCut solution. The remainder of the problem graph is then submitted to a branch and bound algorithm that closely models that used by Shen and Zhang for the Max-2-Sat problem. In Figure 1, the function *bcc_max_cut*

illustrates the top level of a decision version of our program. In the following subsections, we address first the algorithm used in the preprocessing phase and then that used in the branch and bound phase.

5.1 Biconnected component algorithm

Using a divide and conquer approach one first seeks to divide a problem into smaller subproblems that are easier to solve. As all edges in any bipartite graph trivially participate in its MaxCut solution, we identify and characterize two classes of bipartite subgraphs of the problem graph. These subgraphs are the collection of edges that participate in no cycle and the biconnected components that contain no cycles of odd length.

The linear time algorithm we use to identify and characterize these subgraphs is similar to one attributed to Hopcroft and Tarjan [15, 24] in Atallah [2]. Their algorithm uses a single depth first pass over the problem graph to simply list the vertices contained in biconnected components. In contrast, our algorithm uses two depth first passes to both identity and characterize all edges in the graph. The algorithm, illustrated as BCC in Figure 1, is described in the following paragraphs.

The first depth first pass computes three values for each vertex: a discovery time $v.dis$, a value $v.low$, and a boolean value $v.odd$. The algorithm uses a global variable *time* that is incremented when each vertex is first visited. Time is initially 0 and is V when the first depth first pass is completed. The value $v.low$ depends on this vertex discovery time such that the following constraint holds.

$$v.low = \min(v.dis, w.dis \mid (u, w) \text{ is a back edge for some descendant } u \text{ of } v)$$

That is, $v.low$ is the discovery time of the vertex closest to the root that can be reached from the subtree rooted at v through at most one back edge. This value captures the concept of cycle within the graph and allows to identify biconnected components.

The value $v.odd$ for a vertex is determined by whether its depth in the search tree, i.e., the

Figure 1: Decision and biconnected component algorithms.

```
function bcc_max_cut ( G: graph, i: integer ) return boolean
    BCC ( G ); // identify and classify biconnected components in G
    CurrentCut := 0;
    foreach even biconnected component b or bridge b in G do
        CurrentCut := CurrentCut + edgesIn(b);
    end for
    if CurrentCut≥ i return true;
    else
        foreach odd biconnected component bcc in G do
            CurrentCut := CurrentCut + bcc_max_cut1(bcc);
            if CurrentCut ≥ i return true;
        end for
    end if
    return false;
end function


procedure BCC ( G: graph )
    // first pass
    depthfirstsearch ( G )
        foreach vertex v visited
            v.odd := odd( depth of v in search tree );
            v.dis := discovery time ;
            v.low := min(v.dis, w.dis :
                (u, w) is a back edge for some descendant u of v );
        end for
    end depthfirstsearch
    // second depth first search
    depthfirstsearch ( G )
        foreach vertex v visited
            foreach edge e = vu leaving v
                if v.odd = u.odd then mark vu as a member of an odd length cycle; end if
                if v.dis < u.low then mark vu as a bridge ;
                else if v.dis = u.low then vu starts new biconnected component;
                else vu continues current biconnected component ;
                end if
            end for
        end for
    end depthfirstsearch
end procedure
```

distance between the vertex and the root vertex, is an even or odd value. When a back edge is found during the first depth first pass, a comparison of the *odd* values of the vertices it joins allows to determine whether this edge completes a cycle of even or odd length.

The second pass uses the values *dis*, *low* and *odd* to characterize each edge in the graph into one of three categories. An edge that is not part of any cycle is identified as a bridge. All other edges participate in one or more cycles and are separated based on whether they are in a biconnected component that contains an odd length cycle or not. A biconnected component that contains an odd cycle is called itself labeled *odd*, otherwise it is labeled *even*.

This segregation of edges and components allows the top level of the program to rapidly dispose of the simple cases of bridge edges and even biconnected components before addressing the remaining parts of the problem.

5.2 Branch and bound algorithm

The second phase of our approach consists of the recursive branch and bound algorithm that is illustrated in Figure 2. The algorithm operates on three values that are initialized in the function *bcc_max_cut*1 prior to function *bcc_max_cut*2 being called to perform a depth first search of the solution space. This latter function uses several bounding heuristics to limit the solution space, as will be explained.

In this algorithm, the vertices of the problem graph are identified with the integers in the order they are considered for assignment by the algorithm. Thus vertex 1 is assigned to either S or $V - S$ before any higher numbered vertex and so on for the remaining vertices. For each vertex, v, the algorithm maintains three values. The value $v.N$ contains the set of neighbors of v that have not been assigned when v is considered for assignment. That is,

$$v.N = \{x \mid (v, x) \in E \wedge (v < x)\}$$

Figure 2: A decision algorithm for MaxCut of a biconnected component.

```
function bcc_max_cut1 ( G: graph ) return integer
    // initialization
    foreach vertex v in G do
        compute v.N from G;
        v.mb0 := v.mb1 := 0;
    end for
    return bcc_max_cut2( G, 1, |E_g|, 0 );
end function

function bcc_max_cut2 ( G: graph, i, d, g: integer ) return integer
1   if (i > |V_G|) return c(S); // end of branch in search tree
2   cut := 0;
3   if (lower_bound c(S_i) > g) then g := cut := lower_bound c(S_i); adjust(d); end if
4   if (upper_bound c(S_i) < g) return 0;
5   // decide if we want to assign vertex i to S
6   if (i.mb0 ≤ d) ∧ (i.mb0 < i.mb1 + |i.N|) then
7       record_assignment_implications(i, true);
8       new := bcc_max_cut2(G, i + 1, d − i.mb0, g)
9       if (new > cut) then cut := new; adjust(d); end if
10      undo_assignment_implications(i, true);
11  end if
12  // decide if we want to assign vertex i to V − S
13  if (i.mb1 ≤ d) ∧ (i.mb1 ≤ i.mb0 + |i.N|) then
14      record_assignment_implications(i, false);
15      new := bcc_max_cut2(G, i + 1, d − i.mb1, g)
16      if (new > cut) then cut := new; adjust(d); end if
17      undo_assignment_implications(i, false);
18  end if
19  return cut;
end function

procedure record_assignment_implications ( v: vertex, b: boolean )
    if b then for y ∈ N(x) do y.mb0 := y.mb0 + 1 end for;
    else for y ∈ N(x) do y.mb1 := y.mb1 + 1 end for; end if
end procedure

procedure undo_assignment_implications ( v: vertex, b: boolean )
    if b then for y ∈ v.N do y.mb0 := y.mb0 − 1 end for;
    else for y ∈ v.N do y.mb1 := y.mb1 − 1 end for; end if
end procedure
```

The value of $v.N$ is computed once on entering the algorithm and does not change. The values $v.mb0$ and $v.mb1$ are integers that give the number of neighbors of v that have been assigned to S and $V - S$, respectively. Thus, $v.mb0$ gives, based on the current assignment of vertices, the number of edges in the problem graph that will be dropped if vertex v is assigned to the set S. Similarly, $v.mb1$ gives the number of edges that will be dropped if vertex v is assigned to the set $V - S$. These values are maintained during the execution of the algorithms by the procedures record_assignment_implications and undo_assignment_implications.

Once these values are initialized, function *bcc_max_cut2* is called to search the solution space. The parameters to this function, i and g, are the vertex to be considered for assignment and the goal cut value, respectively. The parameter d is the number of edges that may be dropped by the remaining vertex assignments before no further solution on this branch of the search tree will improve the best currently known solution.

The core of the algorithm is expressed in lines 1, 2, 7–10, 14–17, and 19 of the function *bcc_max_cut2* in Figure 2. Together, these lines simply enumerate the cuts of the problem graph and record the best solution. The following theorem concerning this core algorithm is easily proven:

Theorem 2. *The time complexity of the core bcc_max_cut2 algorithm is* $O(n2^n)$.

Because $m = O(n^2)$, this result is better in the worst case than the latest result of Fedin and Kulikov [9] who proposed an algorithm of complexity $poly(m) \cdot 2^{m/4}$.

The remaining lines in the algorithm apply bounding heuristics to improve this performance. In line 3, the following value gives a lower bound on the MaxCut for the current branch prior to the assignment of vertex i,

$$|E| - (D_0^{i-1}(S) + \sum_{j=i}^{n} \max(j.mb0, j.mb1) + \sum_{j=i}^{n} |j.N|).$$

In words, the bound is the the total number edges in the graph less 1) those edges already dropped,

2) the maximum number edges for which one vertex has been assigned that could be dropped by a continuation of the current assignment, and 3) all other remaining edges. As no continuation of the current assignment can ever cut fewer edges than this value, it is a valid lower bound on the value of the MaxCut. This bound is used to improve the known cut value as discussed in Section 4.2.1.

The bounds in lines 6 and 13, e.g., $(i.mb0 \leq d)$ and $(i.mb0 < j.mb1 + |i.N|)$, have a scope which is local to the current vertex and prevent the algorithm from proceeding further with an assignment if that assignment will drop more edges than the limit d, or will drop more edges than it could possibly add, respectively. Shen and Zhang have attributed the second of these bounds to Niedermeier and Rossmanith [19]. The remaining bound is applied in line 4 in Figure 3.

Inspired by the Max-2-Sat lower bounds discussed in Shen and Zhang [22], we have developed several similar bounds for use in line 4 of our algorithm. While these bounds are used as upper bounds on the achievable MaxCut value, they are computed as lower bounds on the number of edges that must be dropped in any continuation of the current solution. For this reason, we refer to these bounds as lower bounds LB0, LB1, and LB2.

The first two of these are defined by the following relations,

$$\text{LB0} = D_0^{i-1}(S) \qquad\qquad \text{LB1} = D_0^{i-1}(S) + \sum_{j=i}^{n} \min(j.mb0, j.mb1)$$

The first, LB0, simply counts the number of edges dropped by the current partial assignment. The second, LB1, adds to that the minimum number of edges that will be dropped by any assignment that completes the current partial assignment. A lower bound is said to be *admissible* if it never exceeds the value it bounds. To ensure the completeness of a branch and bound algorithm, any lower bound must be admissible. It is easy to see that LB0 is admissible as, clearly, no continuation of the current assignment will drop fewer edges than have already been dropped.

To see that LB1 is also admissible, consider the set of edges, $E(i)$, that could still be dropped

by the assignment of vertices $i, \ldots, n$. Taking the vertex i as a boundary vertex between the assigned and unassigned vertices, we segregate the set $E(i)$ into three groups: edges that join i to an assigned vertex, edges that join i to an unassigned vertex, and edges that join unassigned vertices. The number of edges in the first of these groups is given by the values $i.mb0$ and $i.mb1$. Let $\overline{i.mb0}$ represent the edges from i to vertices in S and $\overline{i.mb1}$ represent the edges from i to vertices in $V - S$. The second group of edges consists of those neighbors of i that are contained in $i.N$. We identify the third group of edges as $E(i + i)$. Finally, let $D^*(E(i))$ be the number of edges in $E(i)$ that will be dropped in the continuation of the current assignment; $D^*(E(i)_{i \in S})$ be the number of edges that will be dropped in any continuation based on the assignment of i to set S; and $D^*(E(i)_{i \in (V-S)})$ be the number of edges that will be dropped in any continuation based on the assignment of i to set $V - S$.

With these definitions, we can calculate a relation that supports the claim that LB1 is admissible. First note that,

$$\begin{aligned} E(i) &= \overline{i.mb0} \cup \overline{i.mb1} \cup i.N \cup E(i+1) \qquad \text{and} \\ D^*(E(i)) &= D^*(\overline{i.mb0} \cup \overline{i.mb1} \cup i.N \cup E(i+1)) \end{aligned}$$

For any MaxCut assignment, the relation $D^*(E(i)) = \min(D^*(E(i)_{i \in S}), D^*(E(i)_{i \in (V-S)})$ must hold. Continuing, we have,

$$\begin{aligned} D^*(E(i)_{i \in S}) &= i.mb0 + D^*(i.N \cup E(i+1)) \\ D^*(E(i)_{i \in (V-S)}) &= i.mb1 + D^*(i.N \cup E(i+1)) \\ D^*(E(i)) &= min(i.mb0 + D^*(i.N \cup E(i+1)), i.mb1 + D^*(i.N \cup E(i+1)) \\ D^*(E(i)) &= min(i.mb0, i.mb1) + D^*(i.N \cup E(i+1)) \end{aligned}$$

Note that, for any set X_i,

$$D^*(E(i+1)) \leq D^*(X_i \cup E(i+1)). \tag{1}$$

From this, we have the following result, which may be used to show by induction on i that lower bound LB1 is admissible.

$$min(i.mb0, i.mb1) + D^*(E(i+1)) \quad \leq \quad D^*(E(i)) \tag{2}$$

Notice that lower bound LB0 accounts for all edges where both of the joined vertices have been assigned and that LB1 further accounts for all edges for which only one of the joined vertices have been assigned. Consideration of whether some portion of the set X_i in equation (1) that was omitted from LB1 could, in fact, be incorporated in the lower bound requires to reason about the edges for which neither of the joined vertex has been assigned. For this the following lemmata are useful.

Lemma 3. *In a partial cut assignment S on graph G, if there is an edge ij such that $i.mb0 < i.mb1$ and $j.mb0 < j.mb1$, then* $\text{LB1} + 1 \leq D_G(S)$.

Proof. Given an edge ij for which $i.mb0 < i.mb1$ and $j.mb0 < j.mb1$, first note that LB1 includes both $\min(i.mb0, i.mb1) = i.mb0$ and $\min(j.mb0, j.mb1) = j.mb0$. Now, if vertex i is assigned to set S, then $j.mb0$, the smallest number of edges to be dropped by the assignment of vertex j will increase by 1. On the other hand, if vertex i is assigned to set $V - S$, then $i.mb1$ edges will be dropped by the assignment. Finally, recall that $i.mb1$ is at least 1 larger than $i.mb0$. A similar argument holds for the assignment of vertex j. □

A mirror of this lemma for an edge ij where $i.mb1 < i.mb0$ and $j.mb1 < j.mb0$ can be proven by a similar argument. Lemma 3 and its mirror serve to identify edges joining as yet unassigned vertices for which the lower bound LB1 may be incremented. The following lemma limits number

of such edges that may be used to improve this lower bound.

Lemma 4. *For any subset $X_i \subseteq i.N$, if $|X_i| \leq (i.mb0 - i.mb1)$, then*

$$\min(i.mb0, i.mb1) + D^*(X_i \cup E(i+1)) \leq D^*(E(i)) \tag{3}$$

Proof. The case in which $|X_i| = 0$ has already been shown in (2) above. When $|X_i| > 0$, $(i.mb0 > i.mb1)$ and (3) becomes,

$$i.mb1 + D^*(X_i \cup E(i+1)) \leq D^*(E(i)) \tag{4}$$

There are two sub-cases to consider: i is either assigned to S, or to $(V - S)$ in the final MaxCut assignment.

Case 1: $(i \in S)$. If $i \in S$ in the final MaxCut assignment, then

$$D^*(E(i)_{i \in S}) \leq D^*(E(i)_{i \in (V-S)}).$$

Therefore we have,

$$D^*(E(i)) = D^*(E(i)_{i \in S}) = i.mb0 + D^*(i.N \cup E(i+1))$$

Now, by the condition of the lemma $|X_i| \leq (i.mb0 - i.mb1)$, so $i.mb1 + |X_i| \leq i.mb0$. Also, by Equation (1) above, $D^*(X_i \cup E(i+1)) - |X_i| \leq D^*(E(i+1)) \leq D^*(i.N \cup E(i+1))$. Then,

$$i.mb1 + D^*(X_i \cup E(i+1)) = i.mb1 + |X_i| + D^*(X_i \cup E(i+1)) - |X_i| \leq D^*(E(i)).$$

Case 2: $(i \in (V - S))$. If $i \in (V - S)$ in the final MaxCut assignment, then

$$D^*(E(i)_{i \in (V-S)}) \leq D^*(E(i)_{i \in S}).$$

Therefore we have,

$$D^*(E(i)) = D^*(E(i)_{i\in(V-S)}) = i.mb1 + D^*(i.N \cup E(i+1)).$$

As X_i is a subset of $i.N$, $D^*(X_i \cup E(i+1)) \leq D^*(i.N \cup E(i+1))$. So we have,

$$i.mb1 + D^*(X_i \cup E(i+1)) \leq i.mb1 + D^*(i.N \cup E(i+1)) \leq D^*(E(i)).$$

In either case, the lemma holds. A mirror of this lemma for the case in which $|X_i| \leq (i.mb1 - i.mb0)$ can be proven by a similar argument . □

Figure 3 shows an algorithm that exploits the preceding results to compute our lower bound LB2. The algorithm functions as follows: Lines 1–4 create a local copy of the $mb0$ and $mb1$ values for the unassigned vertices. The **for** loop in lines 6, 7, and 24 alone would simply accumulate LB1. For each iteration of this outer **for** loop, however, the inner **foreach** loops, lines 10–15 and 17–23, increase the values of $mb1$ or $mb0$ of the neighbors of the current vertex, i. Following the logic of the the case where $i.mb0s > i.mb1s$, lines 10–15 increase $v.mb1s$ for each neighbor v of i. This effectively records the first half of the condition required by Lemma 3 for LB1 to be incremented by 1. When a later iteration of the outer loop reaches vertex v, if $v.mb0s > v.mb1s$, line 7 selects the incremented $v.mb1s$. This selection completes the recognition of the Lemma 3 condition and accumulates the incremented lower bound value. At each iteration, only the first $n = \text{abs}(i.mb0s - i.mb1s)$ neighbors are marked as Lemma 4 limits to this number the edges that may increase LB1. Because the algorithm marks the first n neighbors and not necessarily the n neighbors that will be dropped in a MaxCut solution, it will always compute a lower bound that is less than or equal to the bound allowed by Lemma 4. It is simple to prove the following theorem on the time complexity of this algorithm:

Theorem 5. *The time complexity for the lower bound LB2 algorithm is* $O((n-i) + \sum_{j=i}^{n} |j.N|)$.

Figure 3: The algorithm for lower bound LB2.

```
function LB2 ( k: integer) return integer
1    for i = k to |V_G| do
2        i.mb0s = i.mb0;
3        i.mb1s = i.mb1;
4    end for
5    lower_bound := 0;
6    for i = k to |V_G| do
7        lower_bound := lower_bound + min(i.mb0s, i.mb1s);
8        t = abs(i.mb0s − i.mb1s);
9        if (i.mb0s > i.mb1s) then
10           foreach v ∈ i.N do
11               if (t > 0) then
12                   t := t − 1;
13                   v.mb1s := v.mb1s + 1;
14               end if
15           end for
16       else
17           foreach v ∈ i.N do
18               if (t > 0) then
19                   t := t − 1;
20                   v.mb0s := v.mb0s + 1;
21               end if
22           end for
23       end if
24   end for
25   return lower_bound;
end function
```

To show that the lower bound LB2 computed by this algorithm is admissible, we use Lemma 4 or its mirror at each step of the iteration on i giving the following theorem.

Theorem 6. *Lower bound LB2 is admissible, that is,* $LB2(i) \leq D^*(i)$.

6 Experimental results

The current implementation grew out of a project that originated simply as a demonstration of a branch and bound algorithm for the MaxCut problem. The original demonstration was conducted

to compare the performance of a variety of algorithms on randomly constructed MaxCut problems. As a result of this heritage, testing of the current approach has been limited to the same class of random problems as used in the original study.

Many methods have been reported in the literature for generating random graphs with a variety of characteristics. In the current case, we adopt the method used by Erdös and Rényi [8] in which an undirected, simple graph of n vertices is generated by randomly selecting, without replacement, m edges from the $n(n-1)$ possible edges.

Four configurations of our implementation were run on problems sets each composed of 50 randomly generated graphs containing a varying number of vertices and edges as shown in the table. The four configurations of the algorithm are identified here as BB, BB_LB2, BCC, and BCC_LB2. Configuration BB served as a control. It did not include the preprocessing phase and used our lower bound LB1. Configurations BB_LB2 and and BCC added our bound LB2 and the preprocessing phase to the base configuration, respectively. Configuration BCC_LB2 added both of these changes. The average number of branches taken during problem solution was selected as the measure on which to compare the performance of these configurations. This is a valid comparison method as all configurations were built on the same core branch and bound algorithm.

Table 1 shows the results of this preliminary comparison. Each row reports the average branches taken by each configuration as it was run on the same set of problems. The best entry in each row, i.e., the lowest average number of branches taken, is displayed in bold font.

The data shown supports several interesting observations. A comparison of the results for the BB and BB_LB2 configurations clearly shows the superiority of the lower bound LB2 over our lower bound LB1. A comparison of the results for the BB_LB2 and BCC configurations shows that for graphs of low edge to vertex ratio the preprocessing step alone dominates the improved lower bound. However, as that ratio rises the superiority rapidly declines. Finally, the improvements demonstrated by the LB2 lower bound and the addition of the preprocessing phase

Table 1: Performance comparison for four configurations (average branches taken).

Problem		BB	BB_LB2	BCC	BCC_LB2
#var	#edges	#branches	#branches	#branches	#branches
20	18	1.02E+02	4.66E+01	4.20E+00	**3.32E+00**
	20	1.87E+02	9.06E+01	4.98E+00	**3.94E+00**
	25	3.90E+02	1.28E+02	6.41E+01	**4.75E+01**
	50	2.04E+03	**4.84E+02**	1.66E+03	5.11E+02
	75	4.63E+03	9.92E+02	3.92E+03	**9.32E+02**
	100	8.87E+03	**1.92E+03**	8.85E+03	1.98E+03
	125	1.56E+04	**3.25E+03**	1.64E+04	3.66E+03
	150	3.05E+04	**6.66E+03**	3.05E+04	6.92E+03
	325	7.84E+04	1.92E+04	7.80E+04	**1.91E+04**
50	49	3.34E+04	4.31E+03	3.30E+01	**2.42E+01**
	50	3.67E+04	3.52E+03	4.02E+01	**3.17E+01**
	75	1.49E+06	6.80E+04	7.34E+04	**1.80E+04**
	100	5.22E+06	2.31E+05	1.25E+06	**1.36E+05**
	150	3.03E+07	**1.01E+06**	1.70E+07	1.10E+06
100	100	1.19E+08	5.17E+06	1.43E+03	**7.35E+02**
	125	2.99E+08	3.14E+07	2.65E+06	**9.42E+05**

appear additive. Again, as the edge to vertex ratio of the problem graph rises, the preprocessing step appears to become ineffective in reducing the search space. Further tests will be required to clarify these observations.

We note that the effectiveness of our preprocessing phase conforms to current knowledge about the differences in the structure of random graphs that depend on the ratio of graph edges to vertices. Janson [16] has noted that, for fixed $\delta > 0$, that a unique giant component appears when $m = (1 - \delta)n/2$. Our results confirm there is a rapidly increasing probability that this unique component contains an odd length cycle as the ratio of edges to vertices increase past this point.

7 Conclusion and Directions for Future Work

While strong conclusions can not be drawn from the preliminary results we have presented, these results do support two observations. First, we have added some support to the claim that our lower bound bound LB2 is an effective bounding mechanism in this application. Second, we have begun to demonstrate that the effectiveness of our preprocessing mirrors current knowledge about the structure of random graphs as it depends on the ratio of edges to vertices in the graph. That our preprocessing step appears effective only for graphs with low edge to vertex ratio recommends more robust simplification methods be sought.

Though it is very likely unreasonable to hope for a fast general solution to the MaxCut problem, the search for improved methods of attack should continue. Several areas related to the current work that appear promising are developing tighter heuristic bounding functions, identifying better problem-simplifying graph transformations, and extending the current algorithm to handle the more general weighted form of the MaxCut problem.

Taken to the extreme, the best bounding algorithm would solve the original problem. On the one hand, Håstad's results remind that this extreme will be as difficult to solve as that original problem. On the other hand, the application of LB2 to the MaxCut problem has demonstrated that progress in this area is still being made and points out that further improvements may be still found.

The graph simplifying preprocessing phase of the current algorithm removes an easily identifiable class of bipartite graphs from the problem search space. The success of this approach suggests that further improvements might be available by developing fast methods for identifying larger classes of graphs that admit polynomial-time solution. Another facet of the same "simply first" approach, is the search for substructure in the problem graph that can be removed and accounted for, as is seen in Fedin and Kulikov's algorithm [9]. Further work in this area may also prove fruitful.

As mentioned earlier, the current work grew out of a simple demonstration of a branch and bound algorithm. As such, it was developed only to address the simple unweighted version of the MaxCut problem. Extending this approach to address the more general weighted problem would increase the utility of the algorithm and may allow further improvement in the preceding two areas to be achieved.

In this report, we have introduced and defined the MaxCut problem, reviewed some of the extensive literature associated with this problem, mentioned a few of its application, and discussed some of the methods that have been applied to its solution. We then reviewed an approach we have taken to its solution that incorporates a preprocessing step which separated the problem graph into biconnected components with a branch and bound algorithm for the core problem. We sketched our preliminary experimental results which begin to support a claim for the effectiveness of our lower bound LB2 and to demonstrate the range of effectiveness of the preprocessing step we implemented. Finally, we have suggested several areas in which further work appears promising.

8 Acknowledgments

We would like to thank Hantao Zhang and Haiou Shen for their continued interest and support of our work in this area. We also wish to acknowledge the U.S. Army Materiel Systems Analysis Activity for their generous funding of this work.

References

[1] J. Alber, J. Gramm, and R. Niedermeier. Faster exact algorithms for hard problems: a parameterized point of view. *Discrete Mathematics*, 229(1):3–27, 2001.

[2] Mikhail J. Atallah, editor. *Algorithms and theory of computation handbook*, chapter 6, pages 6.6–6.8. CRC Press, Boca Raton, 1999.

[3] Béla Bollobás. *Modern Graph Theory.* Number 184 in Graduate Texts in Mathematics. Springer-Verlag, New York, NY, 2000.

[4] J. Cheriyan, W. H. Cunningnham, L. Tuncel, and Y. Wang. A linear programming and rounding approach to Max-2-Sat. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:395–414, 1996.

[5] Michel Deza and Monique Laurent. Applications of cut polyhedra I. *J. Comput. Appl. Math.*, 55(2):191–216, 1994.

[6] Michel Deza and Monique Laurent. Applications of cut polyhedra II. *J. Comput. Appl. Math.*, 55(2):217–247, 1994.

[7] Oliver Dolezal, Thomas Hofmeister, and Hanno Lefmann. A comparison of approximation algorithms for the MaxCut-problem. Technical Report CI–57/99, Universität Dortmund, Dortmund, Germany, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, 1999.

[8] P. Erdös and A. Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[9] Sergey S. Fedin and Alexander S. Kulikov. A $2^{|E|/4}$-time algorithm for Max-Cut. *Zapiski nauchnyh seminarov POMI*, (293):129–138, 2002.

[10] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, November 1995.

[11] Wilsin Gosti, Giao Nguyen, Marlene Wan, and Min Zhou. Approximation algorithm for the Max-Cut problem. URL: http://citeseer.ist.psu.edu/489604.html.

[12] Jens Gramm, Edward A. Hirsch, Rolf Niedermeier, and Peter Rossmanith. New worst-case upper bounds for MAX-2-SAT with application to MAX-CUT. *Electronic Colloquium on Computational Complexity (ECCC)*, 037, 2000.

[13] Bertrand Guenin. A characterization of weakly bipartite graphs. Preprint, Submitted to Elsevier Preprint, May 2000.

[14] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[15] John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation. *Commnications of the ACM*, 16:372–378, 1973.

[16] Svante Janson. Festschrift in honour of Lennart Carleson and Yngve Domar (Uppsala 1993). *Acta Univ. Upsaliensis Skr. Uppsala Univ. C Organ Hist. 58*, pages 185–190, 1995.

[17] Richard Manning Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, N.Y., 1972.

[18] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31:335–354, 1999.

[19] Rolf Niedermeier and Peter Rossmanith. New upper bounds for MaxSat. *Lecture Notes in Computer Science*, 1644:575–??, 1999.

[20] Svatopluk Poljak and Ssolt Tuza. Maximum cuts and largest bipartite subgraphs. *Combinatorial Optimization, DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 20(1052-1798):181–244, 1995. American Mathematical Society, Providence, R.I.

[21] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

[22] Haiou Shen and Hantao Zhang. Improving exact algorithms for MAX-2-SAT. Proc. of International Symposium on AI and Math (AIM'04), Ft. Lauderdale, FL, April 2004.

[23] Haiou Shen and Hantao Zhang. Study of lower bound functions for MAX-2-SAT. Proceedings of the American Association for Artificial Intelligence, April 2004.

[24] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160, Jun 1972.