MAXIMUM CUT PROBLEM, MAX-CUT

CLAYTON W. COMMANDER

1. INTRODUCTION

The MAXIMUM CUT problem (MAX-CUT) is one of the simplest graph partitioning problems to conceptualize, and yet it is one of the most difficult combinatorial optimization problems to solve. The objective of MAX-CUT is to partition the set of vertices of a graph into two subsets, such that the sum of the weights of the edges having one endpoint in each of the subsets is maximum. This problem is known to be $\mathcal{NP}$-complete [18, 27]; however, it is interesting to note that the inverse problem, i.e., that of looking for the minimum cut in a graph is solvable in polynomial time using network flow techniques [1]. MAX-CUT is an important combinatorial problem and has applications in many fields including VLSI circuit design [9, 32] and statistical physics [5]. For other applications, see [16, 21]. For a detailed survey of MAX-CUT, the reader can refer to [33].

1.1. **Organization.** In this paper, we introduce the MAXIMUM CUT problem and review several heuristic methods which have been applied. In Subsection 3.2 we describe the implementation of a new heuristic based optimizing a quadratic over a hypercube. The heuristic is designed under the C-GRASP (Continuous Greedy Randomized Adaptive Search Procedure) framework. Proposed by Hirsch, Pardalos, and Resende [23], C-GRASP is a new stochastic metaheuristic for continuous global optimization problems. Numerical results are presented and compared with other heuristics from the literature.

1.2. **Idiosyncrasies.** We conclude this section by introducing the symbols and notations we will employ throughout this paper. Denote a graph $G = (V, E)$ as a pair consisting of a set of vertices V, and a set of edges E. Let the map $w : E \mapsto \mathbb{R}$ be a weight function defined on the set of edges. We will denote an edge-weighted graph as a pair (G, w). Thus we can easily generalize an un-weighted graph $G = (V, E)$ as an edge-weighted graph (G, w), by defining the weight function as

$$w_{ij} := \begin{cases} 1, \text{ if } (i,j) \in E, \\ 0, \text{ if } (i,j) \notin E. \end{cases} \tag{1}$$

We use the symbol "$b := a$" to mean "the expression a defines the (new) symbol b." Of course, this could be conveniently extended so that a statement like "$(1-\epsilon)/2 := 7$" means "define the symbol ϵ so that $(1-\epsilon)/2 = 7$ holds." We will employ the typical symbol S^c to denote the complement of the set S; further let $A \setminus B$ denote the set-difference, $A \cup B^c$. Agree to let the expression $x \leftarrow y$ mean that the value of the variable y is assigned to the variable x. Finally, to denote the cardinality of a set S, we use $|S|$. We will use **bold** for words which we define, *italics* for emphasis, and SMALL CAPS for problem names. Any other locally used terms and symbols will be defined in the sections in which they appear.

2. FORMULATION

Consider an undirected edge-weighted graph (G, w), where $G = (V, E)$ is the graph, and w is the weight function. A **cut** is defined as a partition of the vertex set into two

disjoint subsets S and $\bar{S} := V \setminus S$. The **weight** of the cut $(S, \bar{S})$ is given by the function $W : S \times \bar{S} \mapsto \mathbb{R}$ and is defined as

$$W(S, \bar{S}) := \sum_{i \in S, j \in \bar{S}} w_{ij}. \tag{2}$$

For an edge-weighted graph (G, w), a **maximum cut** is a cut of maximum weight and is defined as

$$MC(G, w) := \max_{\forall\ S \subseteq V} W(S, V \setminus S). \tag{3}$$

We can formulate MAX-CUT as the following integer quadratic programming problem:

$$\max\ \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j) \tag{4}$$

subject to:

$$y_i \in \{-1, 1\}, \qquad \forall\, i \in V. \tag{5}$$

To see this, notice that each subset $V \supseteq S := \{i \in V : y_i = 1\}$ induces a cut $(S, \bar{S})$ with corresponding weight equal to

$$W(S, \bar{S}) = \frac{1}{2} \sum_{1 \leq i < j \leq n} w_{ij}(1 - y_i y_j). \tag{6}$$

An alternative formulation of MAX-CUT based on the optimization of a quadratic over the unit hypercube was given by Deza and Laurent in [12].

Theorem 1. *Given a graph $G = (V, E)$ with $|V| = n$, the optimal objective function value of the* MAXIMUM CUT *problem is given by*

$$\max_{x \in [0,1]^n} x^T W(e - x), \tag{7}$$

where $W = [w_{ij}]_{i,j=1}^n$ is the matrix of edge weights, and $e := [1, 1, \ldots, 1]^T$ is the unit vector.

Proof. Let

$$f(x) := x^T W(e - x) \tag{8}$$

denote the objective function from Equation (7). To begin with, notice that the matrix W has a zero diagonal, i.e., $w_{ii} = 0,\ \forall\, i \in 1, 2, \ldots, n$. This implies that $f(x)$ is linear with respect to each variable, and thus there always exists an optimal solution, x^* of (7) such that $x^* \in \{0, 1\}^n$. Therefore, we have shown that

$$\max_{x \in [0,1]^n} x^T W(e - x) = \max_{x \in \{0,1\}^n} x^T W(e - x). \tag{9}$$

The next step is to show that there is a bijection between binary vectors of length n and cuts in G. Consider *any* binary vector $\hat{x} \in \{0, 1\}^n$. Now suppose we partition the vertex set V into two disjoint subsets $V_1 := \{i | \hat{x}_i = 0\}$ and $V_2 := \{i | \hat{x}_i = 1\}$. Then, evaluating the objective function we have

$$f(\hat{x}) = \sum_{(i,j) \in V_1 \times V_2} w_{ij}, \tag{10}$$

which is equal to $W(V_1, V_2)$, the value of the cut defined by the partition of $V = V_1 \bigcup V_2$ (see Equation (2) above).

Alternatively, consider *any* partition of V into two disjoint subsets $V_1, V_2 \subseteq V$. That is

$$V = V_1 \bigcup V_2 \quad \text{and} \quad V_1 \bigcap V_2 = \emptyset.$$

Now, we can construct the vector $\hat{x}$ as follows:

$$\hat{x}_i \quad = \quad \begin{cases} 1, \text{ if } i \in V_1 \\ 0, \text{ if } i \in V_2. \end{cases} \tag{11}$$

Once again, evaluating the objective function on $\hat{x}$, we have

$$f(\hat{x}) = \sum_{(i,j)\in V_1\times V_2} w_{ij}. \tag{12}$$

Hence $f(\hat{x}) = W(V_1, V_2)$ and we have the result.[§] Alas, we have shown the bijection between binary n-vectors and cuts in G. In summary, we have

$$\max_{x\in[0,1]^n} x^T W(e-x) = \max_{x\in\{0,1\}^n} x^T W(e-x) = \max_{V=V_1\bigcup V_2, V_1\bigcap V_2=\emptyset} \sum_{(i,j)\in V_1\times V_2} w_{ij}.$$

□

There are several classes of graphs for which MAX-CUT is solvable in polynomial time [25]. These include planar graphs [11], weakly bipartite graphs with nonnegative edge weights [20], and graphs without K_5 minors [4]. The general problem however is known to be $\mathcal{APX}$-complete [31]. This implies that unless $\mathcal{P} = \mathcal{NP}$, MAX-CUT does not admit a polynomial time approximation scheme [30].

3. METHODS

The MAXIMUM CUT problem is one of the most well-studied discrete optimization problems [27]. Since the problem is $\mathcal{NP}$-hard in general, there has been an incredible amount of research done in which heuristic techniques have been applied. Before we present the new heuristic approach, we review some of the prior work that has been done.

3.1. **Review of Solution Approaches.**

There have been many semidefinite and continuous relaxations based on this formulation. This was first shown by Lovász in [28]. In 1995, Goemans and Williamson [19] used a semidefinite relaxation to achieve an approximation ratio of $.87856$. This implication of this work is significant for two reasons. The first is of course, the drastic improvement of the best known approximation ratio for MAX-CUT of 0.5 which had not been improved in over 20 years [36]. Secondly, and perhaps more significantly is that until 1995, research on approximation algorithms for nonlinear programming problems did not receive much attention. Motivated by the work of Goemans and Williamson, semidefinite programming techniques were applied to an assortment of combinatorial optimization problems successfully yielding the best known approximation algorithms for GRAPH COLORING [7, 26], BETWEENNESS [10], MAXIMUM SATISFIABILITY [13, 19], and MAXIMUM STABLE SET [2], to name a few [29].

As noted in [16], the use of interior point methods for solving the semidefinite programming relaxation have proven to be very efficient. This is because methods such as the one proposed by Benson, Ye, and Zhang in [6] exploit the combinatorial structure of the relaxed problem. Other algorithms based on the nonlinear semidefinite relaxation include the work of Helmberg and Rendl [22] and Homer and Peinado [24].

The work of Burer et al. in [8] describes the implementation of a rank-2 relaxation heuristic dubbed `circut`. This software package was shown to compute better solutions than the randomized heuristic of Goemans and Williamson, in general [16]. In a recent paper dating from 2002, Festa, Pardalos, Resende, and Ribeiro [16] implement and test six randomized heuristics for MAX-CUT. These include variants of Greedy Randomized Adaptive Search Procedures (GRASP), Variable Neighborhood Search, and path-relinking algorithms [35]. Their efforts resulted in improving the best known solutions for several graphs and quickly producing solutions that compare favorably with the method of Goemans and Williamson [19] and `circut` [8]. For several sparse instances, the randomized heuristics presented in [16] outperformed `circut`.

[§]Notice that the result holds even if (without the loss of generality) $V_1 = V$ and $V_2 = \emptyset$. In this case, a cut induced by (V_1, V_2) will be a maximum cut if $w_{ij} \leq 0,\ \forall\, i, j \in V$.

In [25], Butenko et al. derive a "worst-out" heuristic having an approximation ratio of at least $1/3$ which they refer to as the *edge contraction method*. The also present a computational analysis of several greedy construction heuristics for MAX-CUT based on variations of the 0.5-approximation algorithm of Sahni and Gonzalez [36]. With this, we now move on and describe the implementation of a new heuristic for MAX-CUT based on the new metaheuristic Continuous GRASP [23].

3.2. **C-GRASP Heuristic.** The Continuous Greedy Randomized Adaptive Search Procedure (C-GRASP) is a new metaheuristic for continuous global optimization [23]. The method is an extension of the widely known discrete optimization algorithm Greedy Randomized Adaptive Search Procedure (GRASP) [15]. Preliminary results are quite promising, indicating that C-GRASP is able to quickly converge to the global optimum on standard benchmark test functions. The traditional GRASP is a two-phase procedure which

```
procedure GRASP(MaxIter, RandomSeed)
1    f* ← 0
2    X* ← ∅
3    for i = 1 to MaxIter do
4       X ← ConstructionSolution(G, g, X, α)
5       X ← LocalSearch(X, N(X))
6       if f(X) ≥ f(X*) then
7          X* ← X
8          f* ← f(X)
9       end
10   end
11   return X*
end procedure GRASP
```

FIGURE 1. GRASP for maximization

generates solutions through the controlled use of random sampling, greedy selection, and local search. For a given problem Π, let F be the set of feasible solutions for Π. Each solution $X \in F$ is composed of k discrete components $a_1, \ldots, a_k$. GRASP constructs a sequence $\{X\}_i$ of solutions for Π, such that each $X_i \in F$. The algorithm returns the best solution found after all iterations. The GRASP procedure can be described as in the pseudo-code provided in Figure 1. The *construction phase* receives as parameters an instance of the problem G, a ranking function $g : A(X) \mapsto \mathbb{R}$ (where $A(X)$ is the domain of feasible components $a_1, \ldots, a_k$ for a partial solution X), and a parameter $0 < \alpha < 1$. The construction phase begins with an empty partial solution X. Assuming that $|A(X)| = k$, the algorithm creates a list of the best ranked αk components in $A(X)$, and returns a uniformly chosen element x from this list. The current partial solution is augmented to include x, and the procedure is repeated until the solution is feasible, i.e., until $X \in F$.

The *intensification phase* consists of the implementation of a hill-climbing procedure. Given a solution $X \in F$, let $N(X)$ be the set of solutions that can found from X by changing one of the components $a \in X$. Then, $N(X)$ is called the neighborhood of X. The improvement algorithm consists of finding, at each step, the element X^* such that

$$X^* := \arg \max_{X' \in N(X)} f(X'),$$

where $f : F \mapsto \mathbb{R}$ is the objective function of the problem. At the end of each step we make the assignment $X^* \leftarrow X$ if $f(X) > f(X^*)$. The algorithm will eventually achieve a local optimum, in which case the solution X^* is such that $f(X^*) \geq f(X')$ for all $X' \in N(X^*)$. X^* is returned as the best solution from the iteration and the best solution from all iterations is returned as the overall GRASP solution. GRASP has been applied to

```
procedure C-GRASP(n, l, u, f(·), MaxIters, MaxNumIterNoImprov, NumTimesToRun, MaxDirToTry, α)
1   f* ← −∞
2   for j = 1 to NumTimesToRun do
3     x ← UnifRand([l, u))
4     h ← 1
5     NumIterNoImprov ← 0
6     for Iter = 1 to MaxIters do
7       x ← ConstructGreedyRandomized(x, f(·), n, h, l, u, α)
8       x ← LocalSearch(x, f(·), n, h, l, u, MaxDirToTry)
9       if f(x) ≥ f* then
10        x* ← x
11        f* ← f(x)
12        NumIterNoImprov ← 0
13      else
14        NumIterNoImprov ← NumIterNoImprov + 1
15      end if
16      if NumIterNoImprov ≥ MaxNumIterNoImprov then
17        h ← h/2
18        NumIterNoImprov ← 0
19      end if
20    end for
21  end for
22  return x*
end procedure C-GRASP
```

FIGURE 2. C-GRASP pseudo-code adapted from [23].

many discrete problems with excellent results. For an annotated bibliography of GRASP applications, the reader is referred to the work of Festa and Resende in [17].

Like GRASP, the C-GRASP framework is a multi-start procedure consisting of a construction phase and a local search [14]. Specifically, C-GRASP is designed to solve continuous problems subject to box constraints. The feasible domain is given as the n-dimensional rectangle $S := \{x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : l \leq x \leq u\}$, where $l, u \in \mathbb{R}^n$ are such that $l_i \leq u_i$, for $i = 1, 2, \ldots, n$. Pseudo-code for the basic C-GRASP is provided in Figure 2. Notice that the algorithm takes as input the dimension n, upper and lower bounds l and u, the objective function f, and parameters `MaxIters`, `MaxNumIterNoImprov`, `NumTimesToRun`, `MaxDirToTry`, and a number $\alpha \in (0, 1)$.

To begin with, the optimal objective function value f^* is initialized to $-\infty$. The procedure then enters the main body of the algorithm in the **for** loop from lines 2-21. The value `NumTimesToRun` is the total number of C-GRASP iterations that will be performed. To begin with, more initialization takes place as the current solution x is initialized as a random point inside the hyperrectangle, which is generated according to a function `UnifRand`$([l, u))$ which is uniform onto $[l, u)$[†]. Furthermore, the parameter which controls the discretization of the search space, h, is set to 1. Next, the construction phase and local search phases are entered. In line 9, the new solution is compared to the current best solution. If the objective function value corresponding to the current solution dominates the incumbent, then the current solution replaces the incumbent and `NumIterNoImprov` is set to 0. This parameter controls when the discretization measure h is reduced. That is, after a total of `MaxNumIterNoImprov` iterations occur in which no solution better than the current best solution is found, h is set to $h/2$ and the loop returns to line 6. By adjusting the value of h, the algorithm is able to locate general areas of the search space which contain

[†]This is the "typical" definition of a *Uniform* distribution. That is, $P : X \mapsto \mathbb{R}$ is uniform onto $[A, B)$, if, for any subinterval $I \subset [A, B)$, the measure of $P^{-1}(I)$ equals the length of I.

high quality solutions, and then narrow down the search in those particular regions. The best solution after a total of `NumTimesToRun` iterations is returned as the best solution.

The construction phase of the C-GRASP takes as input the randomly generated solution $x \in S$ (see Figure 2, line 3). Beginning with all coordinates unfixed, the method then performs a line search on each unfixed coordinate direction of x holding the other $n-1$ directions constant. The objective function values resulting from the line search solution for each coordinate direction are stored in a vector, say V. An element $v_i \in V$ is then selected uniformly at random from the maximum $(1-\alpha)100\%$ elements of V, and the v_i coordinate direction is fixed. This process repeats until all n coordinates of x have been fixed. The resulting solution is returned as the C-GRASP solution from the current iteration. For a slightly more detailed explanation of this procedure, the reader is referred to [23].

As for the local search phase, this procedure *simulates* the role of calculating the gradient of the objective function $f(\cdot)$. As mentioned earlier, gradients are not used in C-GRASP because oftentimes, they are difficult to compute and result in slow computation times. Therefore, the gradient is approximated as follows. Given the construction phase solution x, the local search generates a set of directions and determines in which direction (if any) the objective function improves.

The directions are calculated according to a bijective function T which maps the interval of integers $[1, 3^n) \cap \mathbb{Z}$ onto their balanced ternary representation. Recall that n is the dimension of the problem under consideration. That is, $T : [1, 3^n) \cap \mathbb{Z} \mapsto \{-1, 0, 1\}^{\mathbb{N}}$. Clearly, as $n \to \infty$, the number of search directions grows exponentially. Therefore, only `MaxDirToTry` directions are generated[‡] and tested on the current solution. For each direction d, the point $\hat{x} := x + hd$ is constructed and $f(\hat{x})$ is computed. Recall that h is the parameter which controls the density of the search space discretization. If the constructed point $\hat{x} \in S$ has a more favorable objective value than the current point x, then $\hat{x}$ replaces x, and the process continues. The phase terminates when a locally optimal point $x^* \in S$ is found. The point x^* is said to be locally optimal if $f(x^*) \geq f(x^* + hd)\ \forall\ d \in \{1, 2, \ldots, \texttt{MaxDirToTry}\}$. Again, for a slightly more in depth description of this procedure, the reader should see the paper by Hirsch et al. [23].

$\alpha = 0.4$	$\texttt{MaxDirToTry} = 20$
$\texttt{NumTimesToRun} = 20$	$\texttt{MaxIters} = 1000$
$\texttt{MaxNumIterNoImrpov} = 1$	

TABLE 1. Parameters used for C-GRASP.

3.2.1. *Computational Results.* The proposed procedure was implemented in the `C++` programming language and complied using Microsoft® Visual C++ 6.0. It was tested on a `PC` equipped with a 1700MHz Intel® Pentium® M processor and 1GB of RAM operating under the Microsoft® Windows® XP environment. The C-GRASP parameters used are provided in Table 1. First, we tested the C-GRASP on 10 instances produced by the Balasundarm-Butenko problem generator in [3]. Though these problems are relatively small, they have proven themselves to be quite formidable against the Multilevel Coordinate Search (MCS) black-box optimization algorithm. We also tested the C-GRASP on 12 instances from the TSPLIB [34] collection of test problems for the TRAVELING SALESMAN PROBLEM. These problems are also used as benchmark problems for testing MAX-CUT heuristics [19].

For further comparison, all instances were tested using the rank-2 relaxation heuristic `circut` [8], as well as with a simple 2-exchange local search heuristic which is outlined in the pseudo-code provided in Figure 3. The method receives as input a parameter `MaxIter`

[‡]uniformly at random

Name	$\|V\|$	Opt	C-GRASP	MCS	`circut`	LS
G5-1	5	126	126	125	125	126
G5-2	5	40	40	39	40	40
G8-1	8	1987	1802	1802	1987	1987
G8-2	8	1688	1631	1671	1688	1688
G10-1	10	1585	1585	1513	1585	1585
G10-2	10	1377	1373	1373	1377	1377
G15-1	15	399	389	389	399	399
G15-2	15	594	594	593	594	594
G20-1	20	273	267	273	273	273
G20-2	20	285	285	282	285	285

TABLE 2. Comparative results from the Balasundaram-Butenko instances from [3].

indicating the maximum number of iterations to be performed and $G = (V, E)$ the instance of the problem whereupon a maximum spanning tree is found using Kruskal's algorithm [1]. The spanning tree, due to its natural bipartite structure provides a feasible solution to which a swap-based local improvement method is applied in line 5. The local improvement works as follows. For all pairs of vertices (u, v) such that $u \in S$ and $v \in \bar{S}$, a swap is performed. That is, we place $u \in \bar{S}$ and $v \in S$. If the objection function is improved, the swap is kept; otherwise, we undo the swap and examine the next (u, v) pair. The local search was tested on the same PC as the C-GRASP. The `circut` heuristic was compiled using Compaq® Visual Fortran on a PC equipped with a 3.60GHz Intel® Xeon® processor and 3.0GB of RAM operating under the Windows® XP environement.

```
procedure LocalSearch(G, MaxIter)
1    f* ← −∞
2    x* ← ∅
3    for j = 1 to MaxIter do
4       x ← KruskalMST(x, G)
5       x ← LocalImprove(x, G)
6       if f(x) ≥ f* then
7          x* ← x
8          f* ← f(x)
9       end if
10   end for
11   return x*
end procedure LocalSearch
```

FIGURE 3. The 2-exchange local search routine.

Table 2 provides computational results of the algorithms on the 10 Balasundarum-Butenko instances from [3]. The first three columns provide the instance name, the number of vertices and the optimal solution. The solutions from the heuristics are provided next. The solutions from the Multilevel Coordinate Search algorithm were provided in [3]. For all of these instances, the time required by the C-GRASP, `circut`, and the local search to find their best solutions was fractions of a second. Computing times were not listed for the MCS algorithm in [3]. Notice that the 2-exchange local search computed optimal solutions for each of these instances, followed closely by `circut` which found optimal cuts for all but one problem. As for the continuous heuristics, the C-GRASP found optimal solutions

for 5 of the 10 instances while the MCS procedure produced optimal cuts for only 1 instance. For the 5 instances where C-GRASP produced suboptimal solutions, the average deviation from the optimum was 3.54%.

Table 3 shows results of the C-GRASP, local search, and `circut` heuristics when applied to 12 instances from the TSPLIB collection of test problems for the TRAVELING SALESMAN problem [34]. The first two columns provide the instance name and the size of the vertex set $|V|$. Next the solutions are provided along with the associated computing time required by the respective heuristic. Notice that for all 12 instances, the three heuristics all found the same solutions. Notice that in terms of computation time, the simplest heuristic, the 2-exchange local search seems to be the best performing of the three methods tested. The rank-2 relaxation algorithm `circut` is also very fast requiring only 2.99 seconds on average to compute the solution. On the other hand, the C-GRASP method did not scale as well as the others. We see that there is a drastic increase in the solution time as the number of vertices increases beyond 48.

This is not particularly surprising. The philosophical reasoning behind the slow computation time of the C-GRASP relative to the discrete heuristics being that the C-GRASP is a *black-box* method and does not take into account any information about the problem other than the objective function. To the contrary, the local search and `circut` specifically exploit the combinatorial structure of the underlying problem. This allows them to quickly calculate high quality solutions.

Name	$\|V\|$	C-GRASP	Time (s)	LS	Time (s)	`circut`	Time (s)
burma14	14	283	0.120	283	0.00	283	.046
gr17	17	24986	0.19	24986	0.00	24986	.047
bays29	29	53990	0.701	53990	0.01	53990	1.109
dantzig42	42	42638	1.832	42638	0.01	42638	1.75
gr48	48	320277	4.216	320277	0.00	320277	3.672
hk48	48	771712	2.804	771712	0.00	771712	2.516
gr96	96	105328	52.425	105328	0.01	105328	14.250
kroA100	100	5897368	66.445	5897368	0.01	5897368	2.359
kroB100	100	5763020	94.175	5763020	0.01	5763020	2.531
kroC100	100	5890745	66.545	5890745	0.01	5890745	2.500
kroD100	100	5463250	94.155	5463250	0.03	5463250	2.547
kroE100	100	5986587	69.64	5986587	0.03	5986587	2.500

TABLE 3. Comparative results from TRAVELING SALESMAN PROBLEM instances [34].

4. CONCLUSIONS

In this paper, we implemented a new metaheuristic for the MAXIMUM CUT problem. In particular, we proposed the use of a continuous greedy randomized adaptive search procedure (C-GRASP) [23], for a continuous formulation of the problem. To our knowledge, this is the first application of C-GRASP to continuous formulations of discrete optimization problems. Numerical results indicate that the procedure is able to compute optimal solutions for problems of relatively small size. However, the method becomes inefficient on problems approaching 100 nodes. The main reason for this is the fact that C-GRASP is a black-box method, in that it does not take advantage of any information about the problem structure. Recall that the only input to the method is some mechanism to compute the objective function. A natural extension of the work presented here is to *enhance* the C-GRASP framework to take advantage of the structure of the problem at hand. Using a priori information about the problem being considered, one could modify the algorithm to include these properties which would presumably reduce the required computation time.

5. Cross References

See also: **Combinatorial test problems and problem generators; Continuous global optimization: Models, algorithms and software; Derivative-free optimization; Greedy randomized adaptive search procedure, GRASP; Heuristic search; Integer programming; NP-complete problems and proof methodology; Quadratic integer programming: Complexity and equivalent forms; Random search methods; Semidefinite programming: Optimality conditions and stability; Semidefinite programming and the sensor network localization problem, SNLP; Solving large scale and sparse semidefinite programs; Variable neighborhood search methods.**

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] N. Alon and N. Kahale. Approximating the independence number via the θ-function. Technical report, Tel Aviv University, Tel Aviv, Israel, 1995.

[3] B. Balasundaram and S. Butenko. Constructing test functions for global optimization using continuous formulations of graph problems. *Journal of Optimization Methods and Software*, 20(4-5):439–452, 2005.

[4] F. Barahona. The max-cut problem in graphs is not contractible to k_5. *Operations Research Letters*, 2:107–111, 1983.

[5] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statisical physics and circuit layout design. *Operations Research*, 36:493–513, 1998.

[6] S. Benso, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10:443–461, 2000.

[7] A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470–516, May 1994.

[8] S. Burer, R.D.C. Monteiro, and Y. Zhang. Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM Journal on Optimization*, 12:503–521, 2001.

[9] K.C. Chang and D.-Z. Du. Efficient algorithms for layer assignment problems. *IEEE Transactions on Computer-Aided Design*, 6:67–78, 1987.

[10] B. Chor and M. SUdan. A geometric approach to betweenness. In *Proceedings of the Third Annual European Symposium Algorithms*, pages 227–237, September 1995.

[11] W. Fernandez de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms*, 8(3):187–198, 1996.

[12] M. Deza and M. Laurent. *Geometry of Cuts and Metrics*. Springer-Verlag, 1997.

[13] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[14] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationaly difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[15] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[16] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

[17] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C. Ribeiro and P.Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[18] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[19] M. Goemans and D.P. Williamson. Improved approximation algorithms for MAX-CUT and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[20] M. Grotschel and W.R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1:23–27, 1981.

[21] W.W. Hager and Y. Krylyuk. Graph partitioning and continuous quadratic programming. *SIAM Journal on Discrete Mathematics*, 12:500–523, 1999.

[22] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2000.

[23] M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende. Global optimization by continuous GRASP. *Optimization Letters*, To appear, 2006.

[24] S. Homer and M. Peinado. Two distributed memory parallel approximation algorithms for Max-Cut. *Journal of Parallel and Distributed Computing*, 1:48–61, 1997.

[25] S. Kahruman-Anderoglu, E. Kolotoglu, S. Butenko, and I.V. Hicks. On greedy construction heuristics for the MAX-CUT problem. *International Journal on Computatonal Science and Engineering*, to appear, 2007.
[26] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *35th Annual Symposium on Foundations of Computer Science*, pages 2–13. IEEE, 1994.
[27] R.M. Karp. Reducability among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
[28] L. Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information Theory*, IT-25(1):1–7, 1979.
[29] H-I. Lu. *Efficient Approximation Algorithms for Some Semidefinite Programs*. Ph.d. dissertation, Brown University, 1996.
[30] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, 1994.
[31] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer System Science*, 43(3):425–440, 1991.
[32] R.Y. Pinter. Optimal layer assignment for interconnect. *Journal of VLSI Computational Systems*, 1:123–137, 1984.
[33] S. Poljak and Z. Tuza. The max-cut problem: A survey. In W. Cook, L. Lovász, and P. Seymour, editors, *Special Year in Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Computer Science. American Mathematical Society, 1995.
[34] G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.
[35] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
[36] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, July 1976.

(C.W. COMMANDER) AIR FORCE RESEARCH LABORATORY, MUNITIONS DIRECTORATE, AND, DEPT. OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, GAINESVILLE, FL USA.
E-mail address: clayton.commander@eglin.af.mil